Addressing Computing's Energy Problem via Minimization of Energy Waste in Computing Systems

A DISSERTATION
SUBMITTED TO THE FACULTY OF THE GRADUATE SCHOOL
OF THE UNIVERSITY OF MINNESOTA
BY

S. Karen Khatamifard

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
Doctor of Philosophy

Ulya R. Karpuzcu

March, 2019

Acknowledgements

There are many people that have earned my gratitude for their contribution to my time in graduate school. Foremost, I would like to express my sincere gratitude to my advisor, Prof. Ulya R. Karpuzcu, for her continuous support, motivation, and encouragement through my PhD. Her guidance was always a crucial help in my research, and how to move forward. I would like to extend my sincerest appreciation for her patience in teaching me how to be self-critical, which helped me in becoming a more successful researcher. I could not have imagined having a better advisor for my PhD.

Besides, I would like to thank my defense committee members, Prof. Lilja, Prof. Sapatnekar, Prof. Chandra, and Prof. Kose, for their valuable feedback and encouragement. I want to specially thank Prof. Lilja and Prof. Kose, for their exceptionally kind support and guidance as research collaborators.

I am incredibly fortunate to know and work with my colleagues in ALTAI Lab: Dr. Ismail Akturk, Zamshed Chowdhury, Mike Resch. Ismail abi has always been a great friend and a mentor to me as a junior researcher. I am truly thankful of him for being incredibly supportive. I would like to thank Zamshed and Mike for being incredible collaborators, and great friends.

My warmest thanks go to my family, for being always supportive. First, I want to thank my wife, Sepideh, for always being there for me, carrying me through the lows and celebrating the highs. I cannot imagine being able to go through this journey without her endless love and kind support. Finally, I am sincerely grateful of my family in Iran, Jamileh and Hashem, Tara, Reza, and Mani, for enduring being apart from me, and their encouragement and love.

Dedication

To Sepideh

Abstract

Gordon Moore's famous prediction, known as the *Moore's Law*, projected that the most cost-effective number of transistors per integrated circuit (IC) doubles every year. This observation has stayed almost true for many decades, until today. Thanks to this trend, computing with ICs has become more powerful and cheaper year-by-year, to a point that almost all electronic devices of today have a computer in it. Robert H. Dennard formulated another trend, ICs getting faster and more energy-efficient exponentially year-by-year as well, known as *Dennard Scaling*. More specifically, Dennard predicted that while we can add exponentially more number of smaller and faster transistors on ICs every year, power density stays nearly the same. This trend was crucial, since power budget of ICs is severely bounded due to cooling limits, and cannot scale well. Unfortunately, Dennard Scaling does not hold any more, which leads to severe growth in total power consumption of newer ICs, bringing systems to a point where large number of transistors of ICs become unusable due to overheating, resulting in *the dark silicon* problem.

The total power consumption of a computing system equals to the product of energy per operation, a proxy for energy-efficiency, and operations per second, computing system's performance, i. e., how fast we can compute. Hence, in a limited power budget, the only way to increase performance is by decreasing energy per operation, i. e., by improving energy-efficiency. This has motivated a diverse set of approaches to form this thesis, all trying to improve the performance even after the end of Dennard scaling by *Addressing Computing's Energy Problem via Minimization of Energy Waste in Computing Systems*, including reliability concerns of reducing energy loss in power conversion of on-chip regulators, a novel covert channel induced by aggressive power management in modern computing systems to optimize energy-efficiency, an accelerator based on processing near-memory for a bioinformatics application, read mapping, to reduce energy waste in data transfer, a modular architecture-level model of parametric variation for Thin-Channel switches (with lower energy loss compared to conventional switches), spatio-temporal omission of synchronization points in noise-tolerant applications to

improve energy-efficiency of execution, and rethinking memory system of stochastic computing systems to reduce energy loss in data conversion.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

In 1965, Gordon Moore's famous prediction, known as the *Moore's Law* [1], projected that the most cost-effective number of transistors per integrated circuit (IC) grows with an exponential rate (doubles every year), as shown in Fig. 1.1. While he originally predicted this trend for at least 10 years (until 1975), this observation stayed true for many more decades, until today. Thanks to this trend, computing with integrated circuits has become more powerful and cheaper year-by-year, to a point that almost all electronic devices of today have a computer in it.

While Moore's Law projects cheaper and more powerful ICs, near a decade later, Robert H. Dennard formulated another trend, ICs getting faster and more energy-efficient exponentially year-by-year as well, known as *Dennard Scaling*[2]. More specifically, Dennard predicted that while we can add exponentially more number of smaller and faster transistors on ICs every year, power density, power consumption per unit of area, stays nearly the same. In other words, while more number of transistors can be integrated into the same size ICs each year, total power consumption stays the same. This trend was crucial, since power budget of ICs is severely bounded due to cooling limits, and cannot scale well. Unfortunately, Dennard Scaling does not hold any more as the operating voltage does not scale in the last generations of new ICs, as shown in Fig. 1.2 [3, 4]. Consequently, power density of newer ICs is growing per new generation, as well, which leads to severe growth in total power consumption of ICs, bringing systems to a point where large number of transistors of ICs become unusable due to overheating. In other words, today ICs have more transistors than the power budget limit, resulting

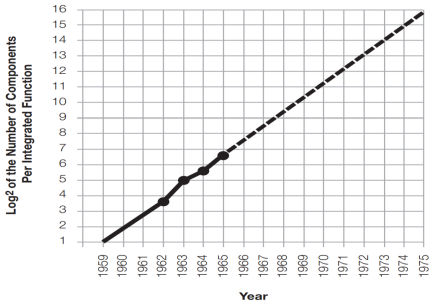


Figure 1.1: Moore's Law.

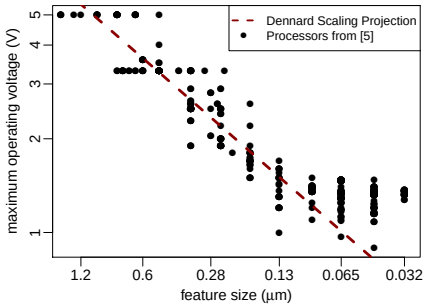


Figure 1.2: Dennard Scaling. (Data from [8])

in *the dark silicon* problem[5, 6, 7, 4].

We can derive total power consumption of a computing system, P, from

$$P = \frac{Energy}{OP} \times \frac{OPs}{second},$$

where energy per operation (OP) is a proxy for energy-efficiency, and operations per second is computing system's performance, i. e., how fast we can compute [3]. As the formula shows, in a limited power budget, the only way to increase performance is by decreasing energy per operation, in other words, by improving energy-efficiency. This has motivated a diverse set of approaches to form this thesis, all trying to improve the performance even after the end of Dennard scaling, by *Addressing Computing's Energy Problem via Minimization of Energy Waste in Computing Systems* at different levels of the system stack, from devices to architecture. In the following, we briefly describe each, and how they reduce energy waste of computing systems.

Tailoring the operating voltage to fine-grain temporal changes in the power and performance needs of the workload can effectively decrease energy waste, and consequently, enhance energy efficiency. Therefore, power-limited computing platforms of today widely deploy integrated (i.e., on-chip) voltage regulation which enables fast fine-grain voltage control. Voltage regulators convert and distribute power from an external energy source to the processor. Unfortunately, power conversion loss is inevitable and projected integrated regulator designs are unlikely to eliminate this loss even asymptotically. Conversion loss is sizable, and as it gets dissipated as heat, on-chip regulators can easily cause thermal emergencies due to their small footprint [9]. In Chapter 2, we introduce

ThermoGater[10], an architectural governor for a collection of practical, thermally-aware regulator gating policies minimizing energy waste in conversion while mitigating (if not preventing) regulator-induced thermal emergencies, also considering potential implications for voltage noise.

Besides, to be able to meet demanding application performance requirements within a tight power budget, runtime power management must track hardware activity at a very fine granularity in both space and time, to minimize energy waste as much as possible, by tuning operating voltage and frequency of different voltage domains, controlled by on-chip voltage regulators. This gives rise to sophisticated power management algorithms, which need the underlying system to be both highly observable (to be able to sense changes in instantaneous power demand timely) and controllable (to be able to react to changes in instantaneous power demand timely). The end goal is allocating the power budget, which itself represents a very critical shared resource, in a fair way among active tasks of execution. Fundamentally, if not carefully orchestrated, any system-wide shared resource can give rise to covert communication. Power budget does not represent an exception, particularly as systems are becoming more and more observable and controllable. In Chapter 3, we demonstrate how power management vulnerabilities can enable covert communication over a previously unexplored, novel class of covert channels [11, 12].

Furthermore, a major source of energy waste of computing systems in the era of exploding data volumes is transferring data from memory to processor (which can easily waste more than half of an application's execution time [13]), in order to do the actual computations on data. In other words, energy wasted in data transfer dominates the actual energy spent on computation itself. This has motivated new computing paradigms which try to minimize energy waste in data transfer, by moving computation units to near or inside memory, known as *Processing In/Near Memory* (PIM or PNM). This way, data transfer becomes unnecessary or less energy consuming, leading to spending most of the total energy in computations, which significantly improves energy-efficiency of computing, and performance consequently. As an example, Bioinformatics applications usually have small amount of computing on a large amount of data. As explained above, they are suffering from wasting most of energy in data transfer. As semiconductor technology revolutionized computing, modern DNA sequencing technology revolutionized genomic research, and as a result, modern sequencing platforms can sequence hundreds

of millions of short DNA fragments in parallel, termed reads. An important application, mapping each of the reads to (the most similar portion of) a reference genome of the same species, i.e., read mapping, is a common critical first step in a diverse set of emerging bioinformatics applications. Mapping represents a search-heavy memory-intensive similarity matching problem, therefore, can greatly benefit from near-memory processing. In Chapter 4, we demonstrate how obtaining a processing near-memory technology, TCAMs[14], can significantly improve energy-efficiency of read mapping (by up to 219×), and consequently performance (by up to 113×) [15].

Another way to minimize energy waste in computing systems is by minimization of energy waste in electrical switches themselves. This has led to the advent of Thin Channel (TC) devices such as FinFETs, in which leakage current, a major source of energy waste, is significantly reduced compared to conventional switches [16]. Under aggressive miniaturization, unconventional digital switches such as TCs rapidly come to light, which introduce new sources of variation in design parameters, and hence challenge the manufacturing process further. As a result, performance and power of manufactured hardware becomes greatly unpredictable. Characterizing variation-incurred unpredictability at early stages of the design necessitates dependable architecture-level models of variation, which distill device- and circuit-level details to accurately evaluate system-level implications. In Chapter 5, we introduce a modular architecture-level model of parametric variation to address this challenge [17]. As a case study, we refine our discussion to a representative class of emerging TC switches, FinFETs.

Moreover, energy is wasted in waiting for other processes, in each synchronization point of parallel applications, which represents a point of serialization, and thereby can easily hurt performance. As demonstrated by recent studies, approximating, i.e., relaxing synchronization by eliminating a subset of synchronization points spatio-temporally can help reduce the amount of energy wasted in waiting, and consequently improve performance, as long as approximation incurred violations of basic execution semantics remain predictable and controllable. Even if the divergence from fully-synchronized execution renders lower computation accuracy rather than catastrophic program termination, for approximation to be viable, the accuracy loss must be bounded. In Chapter 6, we assess the viability of approximate synchronization using Speculative Lock Elision (SLE), which was adopted by hardware transactional memory implementations from industry, as a

baseline for comparison. Specifically, we investigate the efficacy of exploiting semantic and temporal characteristics of critical sections in preventing excessive loss in computation accuracy, and devise a light-weight, proof-of-concept Approximate Speculative Lock Elision (ASLE) implementation, which exploits existing hardware support for SLE [18].

Finally, growing uncertainty in design parameters (and therefore, in design functionality) renders stochastic computing particularly promising, which represents and processes data as quantized probabilities. However, due to the difference in data representation, integrating conventional memory (designed and optimized for non-stochastic computing) in stochastic computing systems inevitably incurs a significant waste of energy in data conversion (up to 80% of system's overall energy consumption [19, 20]). Barely any stochastic computing proposal to-date covers the memory impact. In Chapter 7, as the first study of its kind to the best of our knowledge, we rethink the memory system design for stochastic computing. The result is a seamless stochastic system, StochMem[21], which features analog memory to trade the energy and area overhead of data conversion for computation accuracy. In this manner StochMem can reduce the energy overhead of data conversion by up-to 52.8% at the cost of at most 0.7% loss in computation accuracy.

The remainder of this dissertation is organized as follows:

- Chapter 2 addresses reliability concerns of minimizing energy wasted as conversion loss in on-chip voltage regulators;
- Chapter 3 demonstrates a novel covert channel exploiting vulnerabilities of power management in modern computing systems, where to minimize energy waste, total power budget is shared among all system units based on performance requirements;
- Chapter 4 explores energy waste minimization via designing application-specific accelerators for *read mapping*, a critical step in a diverse set of emerging bioinformatics applications using a near-memory computing technology, TCAMs;
- Chapter 5 introduces a modular architecture-level model of parametric variation for Thin-Channel switches, replacing conventional planar switches to reduce energy waste;
- Chapter 6 explores how spatio-temporal omission of synchronization points in error-tolerant parallel applications can reduce the energy loss due to synchronization, and consequently improve performance, while trading off acceptable levels of accuracy;

- Chapter 7 depicts how using analog memories instead of digital memories in Stochastic Computing systems can significantly cut energy wasted in data conversion;
- Finally, Chapter 8 summarizes our contributions and concludes the discussion.

Besides, to be able to meet demanding application performance requirements within a tight power budget, runtime power management must track hardware activity at a very fine granularity in both space and time, to minimize energy waste as much as possible, by tuning operating voltage and frequency of different voltage domains, controlled by on-chip voltage regulators.

Chapter 2

ThermoGater: Thermally-Aware On-Chip Voltage Regulation

2.1 Motivation

Due to gradual (if not stagnated) voltage scaling, chip power density (power per chip area) has been growing over technology generations [22]. At the same time, cooling limitations prevent a proportional expansion of the chip power budget. In this power-limited environment, the only way to avoid performance degradation becomes to enhance the power efficiency, i.e., performance improvement per unit power consumed. Both, power and performance strongly depend on the operating voltage V_{dd}. Therefore, tailoring V_{dd} to fine-grain temporal changes in the power and performance needs of the workload can effectively enhance power efficiency. As a result, to enable fast fine-grain voltage control, power-limited computing platforms of today widely deploy *integrated* – be it partially *on-package* [23, 24] or entirely *on-chip* [25] – voltage regulation.

In supplying a fixed or time-varying V_{dd} to logic and memory blocks, voltage regulators convert and distribute power from an external energy source to the processor. Regulator *power conversion efficiency*, η, is defined as the ratio of output power to the input power (of the regulator). Due to inevitable losses in power conversion, the best-known and projected integrated regulators fail to reach 100% efficiency even asymptotically. Conversion efficiency η changes as a function of the load current at the regulator output as depicted in Fig. 2.1 for a representative subset of highly-optimized, recent regulator

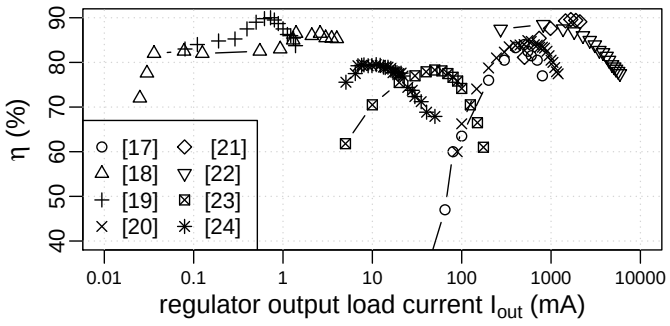


Figure 2.1: Reported power conversion efficiency η of a representative subset of recent, highly optimized regulators from ISSCC 2015 [26, 27, 28, 29, 30, 31, 32, 33].

designs presented in ISSCC 2015 [26, 27, 28, 29, 30, 31, 32, 33]. Output load current I_{out} (as captured by the x-axis) represents a proxy for microarchitectural activity.

Under stringent (voltage) noise requirements, conventional integrated regulators are calibrated to deliver the peak efficiency, η_{peak}, at a specific I_{out} which typically matches maximum microarchitectural activity. However, once deployed, regulators spend most of the time under much lighter load conditions [34]. This is the primary reason behind further power conversion losses.

An off-chip voltage converter supplies the input to on-chip voltage regulators over the *global* power grid. Each on-chip regulator, in turn, delivers power over a *local* grid to its V_{dd}*-domain*, i.e., logic or memory blocks connected to the output of the voltage regulator over the local power grid. Power managers can control the V_{dd} of each domain separately.

An emerging practice is distributing many small regulators (which can be homogeneous or heterogeneous in their topology and electrical characteristics), across each V_{dd}-domain. Connected in parallel, these small regulators can cumulatively supply an I_{out} corresponding to the sum of the $I_{out,r}$ of each component regulator r. In this setting, each component regulator can operate at its peak efficiency $\eta_{peak,r}$ at a specific value (usually maximum) of $I_{out,r}$. If each component regulator is always enforced to operate at its respective $\eta_{peak,r}$, by modulating the number of active component regulators within the V_{dd}-domain, n_{on}, a wide range of I_{out} can be supplied to the respective V_{dd}-domain at the minimum possible power conversion loss. Such reconfigurable power

delivery by selective shut-down, i.e., *gating*, of distributed on-chip regulators in response to spatio-temporal changes in I_{out} can sustain operation at $\eta_{peak(,r)}$ throughout the execution.

However, even at η_{peak}, a notable power conversion loss of around 10% is the case for the best-known industrial integrated regulators of today [24, 25]. At the same time, integrated regulators are miniaturized to minimize the area overhead. As power conversion loss gets dissipated as heat, integrated regulators can easily cause thermal emergencies due to their small footprint. Deviation from the peak efficiency η_{peak} as a result of fluctuations in microarchitectural activity can only exacerbate the thermal profile by generating even more heat due to higher conversion loss.

Integrated regulators close to high-activity functional blocks can increase the span and the temperature of hotspots, or give rise to new hotspots. Integrated regulators close to low-activity regions such as caches, on the other hand, can raise the local temperature. Consequently, the *maximum temperature* observed across chip, T_{max}, can rise, which in turn can lead to a higher *maximum spatial difference in temperature*, i.e., *thermal gradient*. A higher T_{max} may exceed the permissible maximum and degrade performance by triggering thermal throttling. A higher thermal gradient may enforce operation at a lower speed to mask potential timing violations due to temperature-induced spatial timing variations [35]. Mean time to failure (MTTF) for silicon wear-out mechanisms can also decrease notably with increasing temperature [36]. Worse, static power consumption skyrockets at higher temperatures.

Although reconfigurable distributed on-chip power delivery is promising as a new design paradigm to enforce sustained operation at η_{peak} throughout the execution, thermal implications have been overlooked at the architectural level. *How to address thermal emergencies due to integrated voltage regulation* hence forms the focus of this study, considering multiple V_{dd}-domains (each featuring many small regulators) dispersed across chip. This study

- investigates thermal implications of distributed integrated voltage regulation;
- provides an architectural exploration of how spatio-temporal selective gating of regulators can mitigate regulator-induced thermal emergencies;
- introduces *ThermoGater*, a collection of practical runtime policies to orchestrate

thermally-aware regulator gating.

Thermally-aware regulator gating policies from ThermoGater spatio-temporally manage a parallel network of many small regulators dispersed across a V_{dd}-domain, by closely tracking microarchitectural activity. Circuit blocks serviced by a hot regulator do not starve if the respective regulator is gated. Rather, cooler regulators in the same V_{dd}-domain take over the load of the gated regulator. Therefore, spatio-temporal regulator gating can effectively spread the heat concentrated around a hot regulator to neighboring circuit blocks and cap local peak temperatures [37] without compromising performance, as opposed to the vast majority of conventional dynamic thermal management techniques [36]. The end effect is a reduction in both, T_{max} and the thermal gradient across chip.

Be it thermally-aware or not, the goal of regulator gating is to sustain operation at η_{peak} over a wide I_{out} range, which imposes a stringent constraint on the maximum number of regulators that can be gated at a given time. Hence, ThermoGater first determines the number of active regulators within a V_{dd}-domain, n_{on}, required to sustain operation at η_{peak}. ThermoGater next identifies a subset of the regulators of size n_{on} to turn on. Therefore, under thermally-aware regulator gating, circuit blocks may not always be supplied by the closest regulator. The potential adverse affect is higher voltage noise, but ThermoGater takes the implications into account.

To our knowledge, this study represents the first architectural study of thermally-aware regulator gating, which has been only very recently explored at the circuit-level [9] in broad terms where quite generic guidelines are provided for the voltage regulator physical placement to mitigate thermal emergencies. In the following, Section 2.2 details thermal implications of integrated voltage regulation; Section 2.3 provides the background; Section 2.4 covers how regulator gating can help mitigate (if not avoid) regulator-induced thermal emergencies; and Sections 2.5 and 2.6 provide the evaluation.

2.2 Integrated Voltage Regulation: Thermal Impact

The primary design objectives for integrated voltage regulators are i) to maximize the power conversion efficiency η, ii) to minimize the on-chip area overhead, and iii) to minimize the response time to transient changes in I_{out} due to microarchitectural activity.

Regulator power conversion efficiency is defined as $\eta = P_{out}/P_{in}$ where P_{in} and P_{out} are, respectively, the input and output power of the regulator. As depicted in Fig. 2.1, η may degrade significantly off the peak (which typically corresponds to maximum anticipated microarchitectural activity).

Voltage regulators convert and distribute power from an external energy source to the processor. The power dissipated during conversion, P_{loss}, is the difference $P_{in} - P_{out}$. $P_{in} = V_{in} \times I_{in}$ and $P_{out} = V_{out} \times I_{out}$ where V_{in}, I_{in}, V_{out}, and I_{out} are, input voltage, input current, output voltage, and output (load) current of the regulator, respectively. Hence,

$$P_{loss} = P_{out} \times (1/\eta - 1) = V_{out} \times I_{out} \times (1/\eta - 1) \tag{2.1}$$

applies. As Fig. 2.1 reveals, η is a function of I_{out}.

In a reconfigurable distributed power delivery network, the η_r of each component regulator r typically monotonically increases with $I_{out,r}$, reaches a peak, and degrades past the peak, similar to the the curves from Fig. 2.1. However, modulating the number of active component regulators (as explained in Section 2.1, as a function of microarchitectural activity) can make the effective cumulative η barely change with I_{out} (which represents the sum of component $I_{out,r}$) over a wide I_{out} range [24].

Coming back to Eqn. 2.1, at a given V_{out} ($=V_{dd}$), how P_{loss} changes with I_{out} closely tracks how η evolves with I_{out}. P_{loss} is dissipated as heat, and therefore can increase the local temperature by the regulator significantly, as a function of the physical footprint of the regulator. The physical footprint of regulators from Fig. 2.1 varies, as well, due to differences in regulator topologies. From different regulator designs featuring similar η_{peak}, the smaller ones are more likely to cause thermal problems. For each regulator design from Fig. 2.1, P_{loss} per area tends to dip at η_{peak} which incurs the minimum P_{loss} by construction.

A Motivating Case Study: The reported regulator P_{out} per area in Intel's Haswell processor is 33.6W/mm^2 [23], with η_{peak}=90%. In this case, according to Eqn. 2.1, P_{loss} per area becomes 3.7W/mm^2 at η_{peak} [24]. As air (microchannel) cooling limit is around 1.5W/mm^2 (7.9W/mm^2) [38], this P_{loss} per area of 3.7W/mm^2 can result in thermal emergencies, depending on where a regulator resides on-chip[1].

[1] In fact, IBM POWER8 sensors table from [39] features a "VRHOT" signal, the description of which points to "regulator overheating".

2.3 Background & Related Work

2.3.1 Distributed Voltage Regulation

Spatio-temporal selective shutdown of integrated regulators, *regulator gating*, applies to any distributed power delivery network, where a number of regulators are connected in parallel and dispersed across a V_{dd}-domain to maximize the physical proximity to their respective load (circuit blocks). The increased proximity provides very fast response time in tailoring V_{dd} to varying load conditions, i.e., microarchitectural activity. At the same time, regulating V_{dd} at close proximity to the load (circuit blocks) minimizes voltage noise [40, 41]. The component regulators can be *homogeneous* or *heterogeneous* [42] in terms of circuit topology and other electrical characteristics.

Modern processors widely deploy three types of integrated regulators: buck, switched capacitor (SC), and linear low-dropout (LDO). The conversion efficiency of buck regulators can reach over 90%, but usually at a high area penalty. Intel Haswell's fully integrated voltage regulator (FIVR) represents a buck regulator, which mitigates the area overhead due to bulky inductors by keeping these on package while the remaining components of the regulator are placed on chip [24]. While regulator components are distributed between the package and the chip, the regulation happens on-chip in this case. η_{peak}=90% applies where the reported P_{out} per area is 33.6W/mm^2 [23]. For SC regulators, η can reach over 90%, as well, and the footprint is usually smaller than comparable buck regulators [43]. LDO regulators are also area efficient, but both buck and SC regulators can deliver a higher η over a wide V_{out} range when compared to LDO regulators. This is because η of LDO regulators closely tracks V_{out}/V_{in}, and hence degrades significantly as the difference between V_{in} and V_{out} increases (e.g., while supplying a low V_{out} =V_{dd} to the processor). IBM's POWER8 features LDO regulators with a P_{out} per area of 34.5W/mm^2, and η_{peak} of 90.5% [25, 44].

All three types of regulators can represent component regulators in a distributed power delivery network. In this setting, close physical proximity of component regulators to their respective circuit blocks enables sub-nanosecond response times while tracking microarchitectural activity [40, 41]. For example, POWER8 features a network of 64 uniformly distributed LDO regulators per V_{dd}-domain [25, 44]. Each regulator has a bypass mode (i.e., the equivalent of gating) to eliminate P_{loss} when the respective output

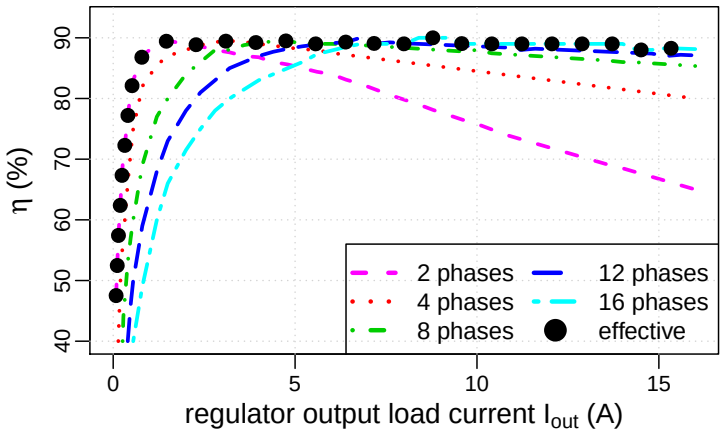


Figure 2.2: η of a 16-phase regulator from Intel [23].

load is low. POWER8 design corresponds to a *homogeneous* distributed network as component LDO regulators (i.e., *microregulators* in IBM terminology) are electrically identical.

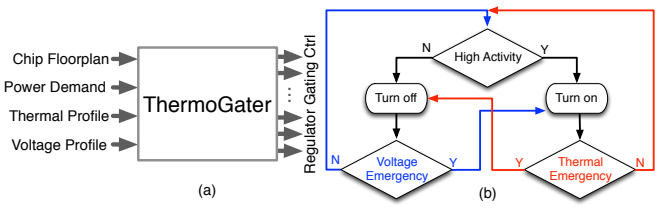
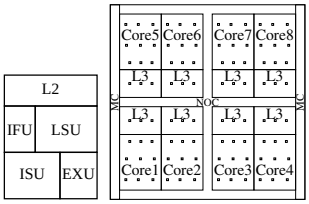


Figure 2.3: Thermally-aware regulator gating: macroscopic (a) and microscopic (b) view.



(a) core (b) chip

Figure 2.4: Simplified floorplan.

Multi-phase regulators are among the most efficient buck and SC regulators. In this case, the regulator itself comprises a parallel network of multiple component regulators which are electrically identical by design, but operate at different phases [26, 29, 32, 45, 46]. Throughout execution, a varying number of the phase-shifted component regulators (i.e., *phases*) are activated to feed the load circuit blocks. For example, Haswell's fully integrated voltage regulator (FIVR) design features phase interleaved buck converters [23, 24, 47]. Different numbers of active phases give rise to different

η vs. I_{out} characteristics, each reaching η_{peak} at a different I_{out} (Fig. 2.2). Hence, by tailoring the number of active phases to the instantaneous I_{out} demand (as dictated by microarchitectural activity), a multi-phase regulator can effectively sustain η_{peak} throughout execution. In this case, distributing phases across a V_{dd}-domain gives rise to an alternative *homogeneous* distributed network design [40, 41].

Although reconfigurable distributed power delivery is emerging as a new design paradigm to increase the performance [48], η [49], and battery life [23, 24], to reduce the energy consumption [34], or to minimize voltage noise [25, 44], thermal implications have been overlooked at the (micro)architectural level.

2.3.2 (Thermally-Oblivious) Regulator Gating

Regulator power conversion efficiency η strongly depends on the output load current I_{out}, as shown in Fig. 2.1. Any time I_{out} deviates from $I_{out,peak}$, at which the regulator operates at peak power conversion efficiency η_{peak}, conversion efficiency degrades. It is possible to reconfigure regulators in response to varying I_{out} demand, such that η vs. I_{out} curve shifts to enforce peak efficiency η_{peak} at the instantaneous I_{out} [34, 50]. In a distributed power delivery network (Section 2.3.1), modulating the number of active regulators can alter the *effective* η vs. I_{out} characteristics. *Regulator gating* refers to such demand-driven turning-on/off of individual component regulators with the goal of sustained operation at η_{peak}. Regulator-gating applies to distributed multi-phase SC [51], distributed multi-phase buck [24], or distributed LDO regulators [25]. Recent representative examples include modulation of bypass modes in IBM POWER8 [25, 44]; and active phases, in Intel Haswell [47, 24, 23] (Section 2.3.1).

As an example, Fig. 2.2 illustrates how gating component regulators (i.e., phases in this case) on demand can sustain operation at the peak efficiency over a wide I_{out} range for the 16-phase Intel buck regulator [23]. Each curve corresponds to a different regulator configuration, as dictated by the number of active phases. Therefore, each curve reaches the peak efficiency at a different I_{out}. Consequently, adaptive gating of active phases can shift the curve to match the instantaneous I_{out} demand (as governed by instantaneous microarchitectural activity) at the peak efficiency, for the entire duration of execution. The *effective* curve, as a result, takes the form of the black dotted trend-line in Fig. 2.2: a practically constant conversion efficiency over a wide I_{out} window, closely tracking

the peak η_{peak}. When the I_{out} demand increases under high microarchitectural activity, additional phases become active to deliver a higher total output current. When the I_{out} demand decreases under low microarchitectural activity, gating applies to a subset of the phases.

2.4 ThermoGater: Thermally-Aware Regulator Gating

As shown in Section 2.2, due to inevitable power conversion losses and the small physical footprint, integrated voltage regulators can easily challenge cooling limits. Regulators may become hotspots themselves or cause significant temperature increase in their close proximity, possibly giving rise to new hotspots or raising the thermal gradient. Both the regulator area and physical placement [9] determine the severity of the regulator incurred thermal problems.

In its thermally-oblivious form as covered in Section 2.3.2, regulator gating can effectively sustain operation at the peak efficiency (Fig. 2.2). However, even the peak efficiency η_{peak} remains around 90% for recent, highly optimized designs, hence incurs a notable power conversion loss. Therefore, sustained operation at the peak efficiency cannot eliminate regulator induced thermal emergencies.

In a distributed power delivery network, regulator gating can be leveraged to mitigate regulator induced thermal problems. Selectively turning off *hot* regulators can keep the local temperature under control. Gating a regulator does not imply (power-)gating its respective load circuit. Cooler regulators (within the same V_{dd}-domain) in close proximity take over the load of the gated regulator. This does not necessarily translate into operation at a lower power conversion efficiency than the peak, i.e., into more conversion efficiency loss. This is because, *be it thermally-aware or not, the goal of regulator gating is to sustain a practically constant peak efficiency over a wide I_{out} range.*

By *spatio-temporally* modulating the location of the active regulators, regulator-gating can help reduce the number and temperature of thermal hotspots, and smoothen the thermal gradient. Spatially changing (the location of) active regulators spreads the heat concentrated around a hot regulator to neighboring circuit blocks. Temporally changing (the set of) active regulators can reduce the average power dissipated by each

regulator (i.e., P_{loss}) over time, and therefore, cap the maximum local temperature.

Be it thermally-aware or not, the goal of regulator gating is to sustain operation at η_{peak} over a wide I_{out} range, which imposes a stringent constraint on the number of active regulators at any point in time during execution. This is because, under regulator gating, only a specific number of active regulators (within a V_{dd}-domain), n_{on}, can supply the demanded I_{out} while operating at η_{peak}. Therefore, a viable regulator gating policy first needs to determine the number of active regulators (within a V_{dd}-domain), n_{on}, required to sustain operation at η_{peak}. Thermal awareness comes only at the next step, in identifying a subset of the regulators of size n_{on} to turn on.

Under thermally-aware regulator gating, logic or memory blocks may not always be supplied by the regulator in closest proximity. The potential adverse affect is higher voltage noise – particularly IR drop, as the effective impedance observed by the load circuit block increases with distance to the supplying regulator. Accordingly, thermally-aware regulator gating is subject to the potential onset of voltage emergencies.

Putting it all together: Three critical factors drive thermally-aware regulator gating decisions: the first factor **(I)** sets the number of active regulators, n_{on}, while the second **(II)** and third **(III)**, determine which n_{on} from the entire set of component regulators within a V_{dd}-domain to activate:

(I) The instantaneous I_{out} demand, as determined by microarchitectural activity: Thermally-aware regulator gating is only legal if n_{on} active regulators can collectively supply the required I_{out} at η_{peak}. As the maximum current a component regulator can supply is limited, the instantaneous I_{out} demand can easily restrict thermally-aware regulator gating.

(II) Thermal emergencies: The n_{on} regulators selected to be turned on should not trigger thermal emergencies.

(III) Voltage emergencies: The n_{on} regulators selected to be turned on should not trigger voltage emergencies.

Fig. 2.3 provides the macroscopic (a) and microscopic (b) view of ThermoGater, an architectural framework which orchestrates thermally-aware regulator gating subject to the constraints imposed by **(I)**, **(II)**, and **(III)**. As depicted in Fig. 2.3(a), in achieving

this, ThermoGater has to monitor the instantaneous power demand along with the thermal and voltage profiles per V_{dd}-domain across the chip. To this end, ThermoGater can deploy thermal or voltage sensors [52, 53, 54] and/or emergency predictors [55] to proactively alter thermally-aware regulator gating decisions. The resulting ThermoGater control loop is depicted in Fig. 2.3(b). When the I_{out} demand increases (decreases) under high (low) microarchitectural activity, ThermoGater turns on (off) the required number of component regulators to sustain operation at η_{peak}, in a thermally- and voltage-noise-aware manner. We will next introduce and evaluate a collection of practical policies to implement ThermoGater's control loop from Fig. 2.3(b).

2.5 Evaluation Setup

In the rest of this study, we will refer to *phases* (in Intel terminology) or *microregulators* (in IBM terminology) – as covered in Section 2.3.1 – as *(component) voltage regulators (VR).*

Benchmarks: We experiment with all benchmarks from SPLASH2x
[56] to cover a representative range of application domains and characteristics. We restrict our analysis to the region-of-interest (ROI) of the benchmarks where the actual computation takes place. Our simulations involve 8 threads.

Architectural, thermal, power simulation: To quantitatively characterize thermally-aware regulator gating, without loss of generality, we model an 8-core processor similar to IBM POWER8 [44]. Table 2.1 captures the technology and architecture parameters. Fig. 2.4a depicts the floorplan of a core which comprises an IFU (instruction fetch unit), an ISU (instruction scheduling unit), an EXU (execution unit), an LSU (load store unit) and a private L2. L1 data cache (not shown in the figure) resides inside LSU; L1 instruction cache, inside IFU. Fig. 2.4b demonstrates the floorplan for the entire chip of 8 cores, including the memory controller (MC), network-on-chip (NOC), and 96 integrated voltage regulators, shown as squares.

We experiment with 16 V_{dd}-domains: a separate V_{dd}-domain for each core (+ private L2), and for each L3 bank, mimicking the IBM POWER8 design [44]. Each per core V_{dd}-domain incorporates 9; each per L3 bank V_{dd}-domain, 3 (component) VRs. Over 8 cores and 8 L3 banks (Fig. 2.4b), this totals up to 96 on-chip VRs distributed among 16

V_{dd}-domains[2].

We integrated MR2 [57] version of McPAT [58] into SNIPER6.0 [59] microarchitectural simulator to collect power traces. The model is calibrated such that the share of static power does not exceed 30% of the total consumption (of the entire chip) at 80°C. We use Hotspot6.0 [60] to model temperature. Temperature (the output of Hotspot) is used to calculate the static power (an input to Hotspot), therefore Hotspot is invoked each time in a closed feedback loop until convergence. We adapt HotSpot6.0's default cooling package (which mimics POWER7+). We expect our observations to generally hold under better cooling, because: (i) cooling solutions usually uniformly affect the chip; (ii) on-chip regulators have much smaller footprint than logic or memory blocks; (iii) regulator power efficiency loss is inevitable.

Technology Parameters
Technology node: 22nm, Frequency: 4.0GHz
TDP: 150W, Area: $441mm^2$, Vdd: 1.03V
Architecture Parameters
cores: 8, issue width: 8
architectural floating point registers: 64
architectural integer registers: 32
L1-I cache: 32KB, 8-way, 64B, LRU, 1-cycle hit
L1-D cache: 64KB, 8-way, 64B, LRU, 1-cycle hit
L2 cache: 512KB, 8-way, 128B, LRU, 11-cycle hit
L3 cache: 64MB, 8-way, 128B, LRU, 30-cycle hit

Table 2.1: Technology and architecture parameters.

Voltage noise simulation: We deploy an extended version of Volt-Spot [61] to capture the impact of thermally aware regulator gating on voltage noise. The extended version, validated against SPICE, models all critical (component) VR parameters. VoltSpot requires cycle-accurate power traces. Since generating them for the entire duration of execution is too expensive, we rely on sampling, following the suggested methodology for VoltSpot [62]: 200 equally distant samples are captured from the entire execution of each application. Each sample contains 2K cycles. The first 1K is used to warm up

[2]The high computational complexity of our thermal and voltage noise simulators prevented experimentation with larger number of component on-chip regulators. A lower regulator count worsens both the thermal and the voltage noise profile, therefore, we selected the maximum possible (component) regulator count permissible by our simulation infrastructure. IBM design features a similar number of V_{dd}-domains, however, many more (component) regulators per domain.

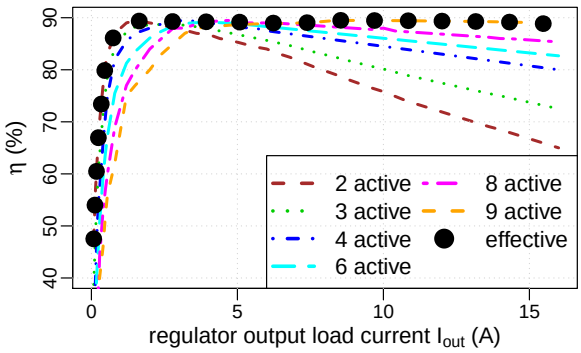


Figure 2.5: η vs. I_{out} used for calibration.

VoltSpot; the rest, for actual analysis.

Voltage regulator properties: In the simulations, we use 96 voltage regulators (VRs) distributed across the chip over 16 V_{dd}-domains, as captured by the little squares in Fig 2.4b. Area of each VR is 0.04mm^2. Without loss of generality, we calibrate these VRs to match the conversion efficiency η vs. I_{out} characteristics of Intel's Haswell design [47]. Recall that this design keeps the inductors off-chip (on the package), however, the actual voltage regulation is performed on-chip. We picked these curves just for calibration purposes, as this design represents one of the most efficient regulators from industry. Fig. 2.5 captures the η vs. I_{out} characteristics for the (component) VRs in each per-core V_{dd}-domain. In this case, each component VR (i.e., *phase*) provides around I_{out}=1.5A load current at the highest conversion efficiency η_{peak}=90%. In each per-core V_{dd}-domain, while all 9 (component) VRs should be active at the highest performance point, lower number of active VRs can still provide operation at η_{peak} at lower processor utilization (Section 2.3.2).

Voltage regulator placement: On-chip regulator placement can affect the voltage noise profile significantly. In order to eliminate any adverse bias from our analysis, we obtain the (voltage-noise) optimal placement following the methodology from [63], which finds the optimal C4[3] pad locations in order to minimize the maximum (both transient and steady-state) voltage noise. We mimic the devised algorithm (i.e., *Deep*

[3]Controlled Collapse Chip Connection (C4) pads connect the off-chip voltage converter to the global power grid.

Optimization) to find the optimal location of on-chip component VRs since similar to C4 pads, they act as inputs to the power delivery network[4]. Starting with the VRs in immediate vicinity of where the voltage noise peaks, we attempt to move VRs away one by one in each iteration. We accept a change of position only if it decreases the maximum voltage noise. We continue until the placement converges. The resulting optimized placement slightly deviates from the uniformly distributed placement from Fig. 2.4b (which results in an increase in the maximum voltage noise by less than 0.4%). We find that the voltage noise profiles of the two placements are very similar otherwise. The uniform placement is more convenient to model due to its regularity. Therefore, in the following, we will stick to the more regular uniform placement.

2.6 Evaluation

Any viable thermally-aware regulator gating policy needs to carefully determine two critical parameters: *number* of active regulators and their respective *location*, on a per V_{dd}-domain basis. Section 2.6.1 focuses on the first; Section 2.6.2, on the second, parameter. In Section 2.6.3 we devise practical ThermoGater policies which can effectively orchestrate thermally-aware spatio-temporal regulator gating. Section 2.6.4 concludes the evaluation with a design space exploration.

2.6.1 Setting the Number of Active Regulators

Regulator power conversion efficiency η is a strong function of the load current I_{out} (Section 2.3.1). To be able to sustain operation at the peak efficiency η_{peak} throughout the execution, only as many VRs should be active as necessary to supply the instantaneous current demand I_{out} at η_{peak} as demonstrated in Fig. 2.5. If more or less VRs remain active than needed to operate at η_{peak}, the degradation in power conversion efficiency causes higher power conversion loss, P_{loss} (Eqn. 2.1), to be dissipated as heat, which can easily exacerbate the thermal profile and cause thermal emergencies.

As a representative example, Fig. 2.6 shows how the cumulative number of active regulators over all V_{dd}-domains to enforce operation at η_{peak} (i.e., cumulative n_{on}) changes over the execution time for an 8-threaded run of *lu_ncb*. Time is shown on the

[4]C4 pads feed the global; on-chip VRs, the local power grids, respectively.

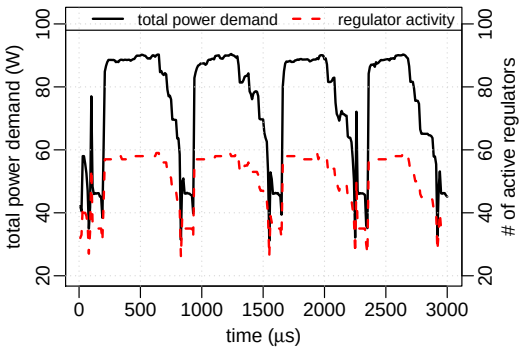


Figure 2.6: Evolution of #active regulators with time.

x-axis while the left y-axis represents the total power demand and the right one shows the active regulator count. As Fig. 2.5 reveals, operation at η_{peak} under higher (lower) I_{out} demand requires more (less) active regulators. Accordingly, we observe in Fig. 2.6 that regulator activity closely tracks temporal changes in total power demand, which represents $V_{dd} \times I_{out}$ with V_{dd} being constant. Recall that thermally-aware regulator gating is only viable if the set of active regulators can collectively supply the required I_{out} at η_{peak} (Section 2.4).

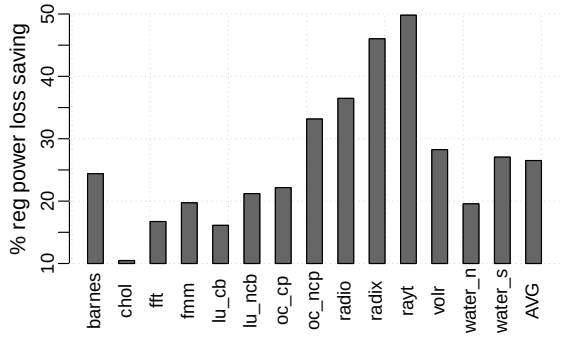


Figure 2.7: P_{loss} improvement under optimal gating.

We next analyze the impact of active regulator count on the power conversion loss, P_{loss}. We compare two cases: keeping all 96 regulators active (*all-on*) vs. keeping only as many regulators active as necessary to sustain η_{peak} (i.e., cumulative n_{on} over all V_{dd}-domains, as it was the case for Fig. 2.6, by regulator gating). We find the average (over

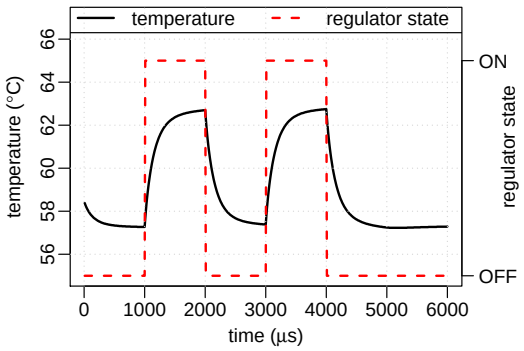


Figure 2.8: A representative thermal profile under Naïve.

time and space) P_{loss} of the regulators for each case, and report the difference in Fig. 2.7. The y-axis captures % P_{loss} saving in comparison to *all-on* under regulator gating. We observe a wide range of savings from 10.4% for *cholesky* up to 49.8% for *raytrace*. % saving strongly depends on the total power consumption of the application: If power consumption stays high (*cholesky*) throughout execution, many more active regulators are required to operate at η_{peak} (as Figs. 2.5 and 2.6 reveal), hence the difference to *all-on* becomes much less pronounced where we keep all regulators active all the time. As a result, regulator gating (if at all) can only save a very little fraction of P_{loss}. On the other hand, for low power applications (*raytrace*), savings become significant as only a notably lower number of active regulators is necessary to operate at η_{peak}. Overall, we observe that for most of the applications, regulator gating can significantly reduce P_{loss}, by ≈26.5% on average.

2.6.2 Setting the Location of Active Regulators

Be it thermally-aware or not, the goal of regulator gating is to sustain operation at η_{peak} over a wide I_{out} range. Once we determine the number of active regulators, n_{on} (on a per V_{dd}-domain basis), required to sustain operation at η_{peak}, the question becomes *which subset of the regulators of size n_{on} to select to turn on.* Thermally-oblivious regulator gating policies (Section 2.3.2) typically focus on n_{on} calculation only and do not consider potential thermal implications at all, as opposed to ThermoGater.

In selecting n_{on} regulators (within a V_{dd}-domain) to turn on, omitting the hottest

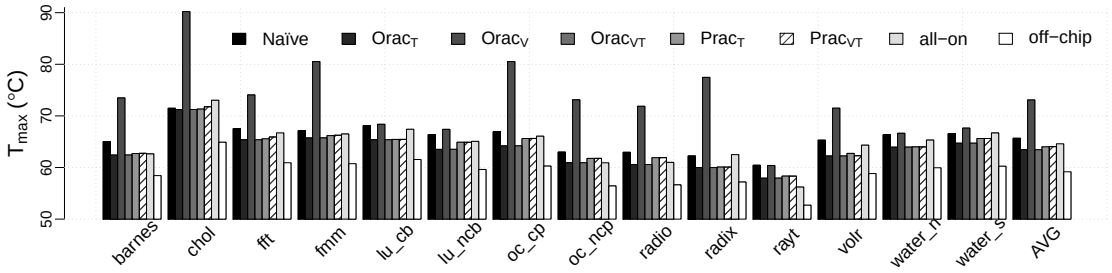


Figure 2.9: Maximum chip-wide temperature under different regulator gating policies.

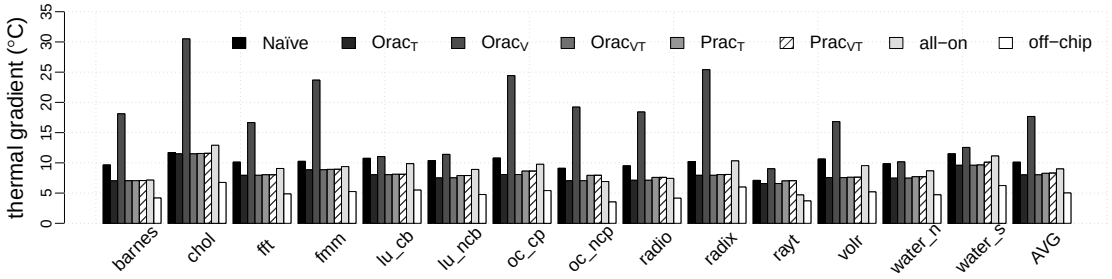


Figure 2.10: Maximum thermal gradient under different regulator gating policies.

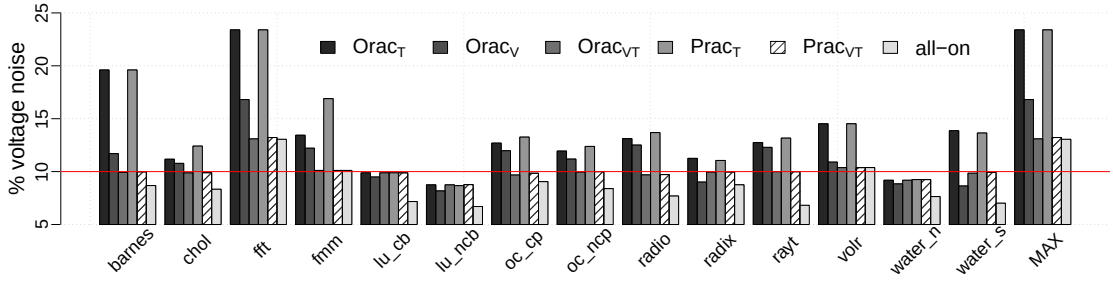


Figure 2.11: Maximum voltage noise under different regulator gating policies.

regulators can decrease the local temperature, which in turn can help reduce the number or temperature of thermal hotspots, and keep the thermal gradient under control. Such a selection, however, may cause on-chip logic and memory blocks not to be supplied by the regulators in closest physical proximity, and hence, may accelerate the onset of voltage emergencies. In the following, we analyze these adverse effects by starting with a thermally-aware greedy gating policy (Section 2.6.2) and continuing with predictive voltage-noise oblivious thermally-aware (Section 2.6.2) and thermally-oblivious voltage-noise-aware (Section 2.6.2) gating policies, before concluding with an oracular policy which considers the thermal and voltage noise implications together (Section 2.6.2). These policies only differ in the selection of n_{on} regulators on a per V_{dd}-domain basis.

Thermally-Aware *Naïve* Gating

Spatio-temporal thermal information can reveal which regulators are more likely to become potential hotspots. Based on this information, we designate a greedy gating policy, *Naïve*, which picks the n_{on} coolest of the regulators to turn on (in each V_{dd}-domain). This gives the hottest regulators time (until the next gating decision takes place) to cool down. Practically, *Naïve* keeps the maximum possible number of hottest regulators off at each decision point. All of the thermally-aware gating policies in this study, including *Naïve* draw gating decisions every 1ms[5].

Fig. 2.8 depicts how the temperature T around a representative regulator changes under *Naïve* during the execution of *lu_ncb*. The x-axis captures the time; the y-axis on left (right), T in °C (regulator state). A similar trend applies to other applications or regulators across different V_{dd}-domains. At the first decision point at 1ms, this specific regulator is not one of the hottest regulators in its respective V_{dd}-domain, so *Naïve* turns it on. However, at the second decision point at 2ms, the regulator becomes hot enough (with respect to the rest of the regulators) that *Naïve* turns it off and lets it cool down at least until the next decision point. Regulator T changes by more than 5°C during this process.

Considering implications for reliability, keeping the thermal gradient (the maximum spatial difference in T) under control can become more critical than capping the maximum

[5]According to our experiments, choosing a 100× shorter period improves the accuracy only by less than 1%.

temperature across chip, T_{max} [35]. Throughout this study, we will report the impact of different gating policies on both, the thermal gradient and T_{max}. Specifically, we will report the *temporal maximum* of both, the thermal gradient and T_{max}.

Fig. 2.10 compares the maximum thermal gradient under different gating policies with two baselines: *all-on* where all 96 on-chip regulators are active all the time, and *off-chip* which excludes on-chip regulation. When compared to *off-chip*, *all-on* increases the thermal gradient significantly, by 79.4% on average. *Naïve* enforces η_{peak} throughout the execution, however, exacerbates the thermal gradient further, by 12.5% on average, over *all-on*. Similarly, as Fig. 2.9 reveals, *all-on* increases T_{max} by 5.4°C over *off-chip*; and *Naïve* by 1.1°C over *all-on*. Clearly, *Naïve* does not represent a feasible thermally-aware gating policy.

Thermally-Aware Oracular Gating

At each temporal decision point, *Naïve* swaps a hot regulator (from each V_{dd}-domain) which was on, say *hot*, with a cooler one which was off, say *cool*. At the decision point t, the temperature of *hot*, $T_{t,hot}$ is higher than $T_{t,cool}$. However, depending on the instantaneous power demand, once *cool* is turned on, T_{cool} can easily exceed the temperature *hot* would assume if *hot* was kept on. Therefore, keeping *hot* on until the next decision point can turn out to be a better option, although there are cooler regulators such as *cool* to turn on. *Naïve*'s poor performance hence stems from not considering the impact of gating to the future thermal profile. This motivates a policy of predictive nature. We start with an oracular policy, $Orac_T$, which turns off the *hottest-to-be* (as opposed to the instantaneous *hottest* under *Naïve*) regulators at each decision point in selecting n_{on} regulators to turn on within each V_{dd}-domain. We assume that $Orac_T$ has full-fledged oracular knowledge about the output power demand and temperature of each regulator at each decision point in time under all possible gating decisions. Later in Section 2.6.3 we will factor in practical limitations.

Fig. 2.10 reveals that $Orac_T$ can improve the maximum thermal gradient by 10.9% on average over *all-on*. At the same time, recall that $Orac_T$ enforces operation at η_{peak} throughout execution (as the rest of the gating policies we study), while power conversion loss is inevitable under *all-on* (Fig. 2.5). The decrease in maximum thermal gradient over *Naïve* is over 20.7%. Similarly, as Fig. 2.9 reveals, $Orac_T$ decreases T_{max} by 1.2°C

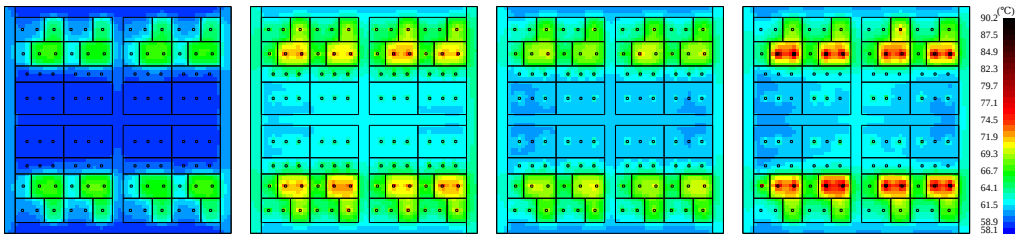
over *all-on*; and by 2.2°C over *Naïve*.

We next analyze the spatial impact of regulator gating using heat maps. Fig. 2.12 shows a representative thermal frame of *cholesky* without loss of generality, where T_{max} peaks during execution (under different gating policies we report, the frames provided differ by less than 100μs). If we do not use on-chip regulators at all (*off-chip*), T_{max} does not exceed 66°C (Fig. 2.12a). Keeping all on-chip regulators on all the time (*all-on*) triggers hotspots on some of the LSUs and EXUs, increasing T_{max} to 73°C (Fig. 2.12b). As Fig. 2.12c reveals, by predictive thermally-aware gating, $Orac_T$ can eliminate these hotspots and decrease the instantaneous T_{max} to nearly 71.2°C, while operating at η_{peak} as opposed to *all-on*.

On-chip memory blocks are usually cooler and less power hungry than logic units. In picking n_{on} regulators to sustain operation at η_{peak} (on a per V_{dd}-domain basis), $Orac_T$ effectively moves active regulators farther from logic units and closer to the memory blocks. This reduces the maximum thermal gradient and T_{max}, however, keeping supplying regulators further away from logic units worsens the voltage noise profile. Fig. 2.11 captures the impact of different gating policies on voltage noise. $Orac_T$ exacerbates the voltage noise significantly for all applications: On average, the maximum voltage noise becomes 23.4% of the nominal V_{dd}, which is 79.3% larger than *all-on*. Safe operation under $Orac_T$ therefore would demand an excessive V_{dd} guard-band, which implies operation at a much higher nominal V_{dd}. This in turn can increase the power consumption noticeably and is more than likely to wipe out the savings in P_{loss} under regulator gating. Accordingly, a feasible thermally-aware gating policy cannot be voltage-noise-oblivious.

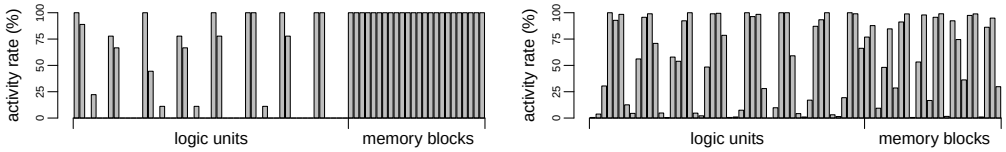
Voltage-Noise-Aware Oracular Gating

We next analyze the voltage-noise-aware dual policy of $Orac_T$: $Orac_V$. In picking n_{on} regulators to sustain operation at η_{peak} (on a per V_{dd}-domain basis), $Orac_V$ chooses regulators based on the spatial voltage noise profile only, in a thermally-oblivious way. Therefore, $Orac_V$ tends to keep the regulators physically closest to high voltage noise regions on. Similar to $Orac_T$, we assume that $Orac_V$ has full-fledged oracular knowledge about the output power demand per regulator and domain-wide voltage noise profile across chip at each decision point in time under all possible gating decisions. Later in



(a) off-chip (b) all-on (c) $Orac_T$ (d) $Orac_V$

Figure 2.12: Representative heat maps under different regulator gating policies.



(a) $Orac_T$ (b) $Orac_V$

Figure 2.13: Regulator activity under $Orac_T$ vs. $Orac_V$.

Section 2.6.3 we will factor in practical limitations.

Typically, voltage noise is worse near logic units since they are more power hungry than on-chip memory blocks. Fig. 2.13 compares spatial regulator activity under $Orac_T$ (a) and $Orac_V$ (b) for *lu_ncb* as an example, without loss of generality. Y-axis represents % of execution time during which a regulator stays on. On the x-axis, we have 72 bars, each representing a regulator from a (per-core) V_{dd}-domain (over all domains), binned into two groups according to the location within the V_{dd}-domain: regulators supplying logic units on the left; on-chip memory blocks, on the right. Recall that each per core V_{dd}-domain incorporates a core along with its private L1 and private L2 (Section 2.5). We observe that while $Orac_T$ tends to turn more regulators off in the immediate proximity of logic units, $Orac_V$ does the opposite to keep the voltage noise under control.

Fig. 2.14 shows a critical spatio-temporal voltage noise sample from *fft* which causes the worst voltage noise profile under $Orac_T$ across all the applications as Fig. 2.11 reveals. Gating according to spatial voltage noise information helps a lot in this case: $Orac_V$

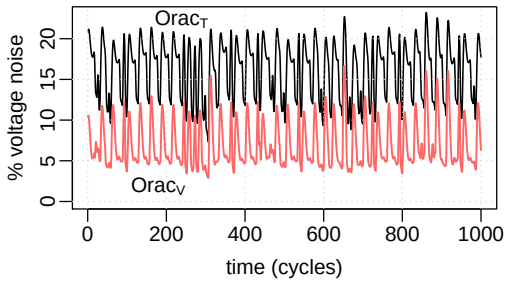


Figure 2.14: Impact on voltage noise: $Orac_T$ vs. $Orac_V$.

decreases the maximum voltage noise significantly by 28.2% over $Orac_T$.

Fig. 2.11 reports the maximum voltage noise for all applications under different gating policies. We report the maximum across all V_{dd}-domains. We observe that under $Orac_V$ maximum voltage noise remains within 28.4% of the maximum voltage noise under *all-on* (which represents the best case for voltage noise, as each logic or memory block is guaranteed to be supplied by the regulator in closest physical proximity). However, $Orac_V$ can sustain operation at η_{peak}, while the effective conversion efficiency η under *all-on* ($\leq \eta_{peak}$) fluctuates throughout the execution. Voltage noise profile under $Orac_V$ is worse than under *all-on* for most of the applications. For these cases, turning on more regulators than n_{on} can help at the expense of operating at a degraded η much below η_{peak}, but maximizing conversion efficiency criterion prevents $Orac_V$ from turning on more than n_{on} regulators (as necessary to operate at η_{peak}) on a per V_{dd}-domain basis.

By turning on the regulators mostly near logic units (Fig. 2.13b), $Orac_V$ can severely worsen the thermal profile as the heat map from Fig. 2.12d reveals: T_{max} increases to more than 90° in this case. The poor thermal profile under $Orac_V$ is summarized for all applications in Fig.s 2.10 and 2.9. $Orac_V$ increases the maximum thermal gradient by nearly 96.3% (120.9%); T_{max}, by 8.5°C (9.6°C), in comparison to *all-on* ($Orac_T$), respectively.

Putting It All Together: $Orac_{VT}$

We next devise a thermally-aware gating policy which also takes the implications for voltage-noise into account: $Orac_{VT}$. Similar to $Orac_T$ and $Orac_V$, $Orac_{VT}$ is of oracular nature (in Section 2.6.3 we will factor in practical limitations). $Orac_{VT}$ captures the

App. names	barnes	chol	fft	fmm	oc_cp	oc_ncp	radio	radix	rayt	volr	water_s	**AVG**
% exec. time	0.67	0.001	0.49	0.024	0.50	0.002	0.008	0.06	0.032	0.002	0.11	**0.13**

Table 2.2: % execution time spent in voltage emergencies under $Orac_T$ (reported are non-zero values only).

impact on voltage noise by tracking the frequency (of occurrence) of voltage emergencies. We define a *voltage emergency* as the maximum voltage noise exceeding a threshold of 10% (of the nominal V_{dd}), on a per V_{dd}-domain basis. The horizontal line in Fig. 2.11 marks this threshold.

Table 2.2 summarizes % of cycles spent in voltage emergencies throughout the execution across all benchmarks under $Orac_T$. We observe that for all of the applications % cycles spend in voltage emergencies stays under 1%. Moreover, changes in temperature take much longer than the duration of a voltage emergency (which is in the order of cycles). In other words, the time constant for temperature changes is higher, therefore, a change in gating schedule of very short duration is usually not long enough to affect the temperature.

Based on these observations, $Orac_{VT}$ mimics $Orac_T$ by default and switches to *all-on*, on a per V_{dd}-domain basis, only upon the prediction of a voltage emergency (the prediction accuracy is perfect under $Orac_{VT}$ due to the oracular nature). Therefore, for applications which do not experience any voltage emergencies per our definition (such as *lu_ncb*), $Orac_{VT}$ becomes equivalent to $Orac_T$. Fig.s 2.10, 2.9, and 2.11 reveal the maximum thermal gradient, T_{max}, and maximum voltage noise under $Orac_{VT}$. We observe that under $Orac_{VT}$ the thermal profile effectively converges to the thermal profile under $Orac_T$; and the voltage noise profile, to *all-on*. Higher temperature under $Orac_V$ can increase the static power significantly, and thereby, the overall power consumption. This is how $Orac_V$ can render higher voltage noise than the less power-hungry $Orac_{VT}$ for some benchmarks.

2.6.3 Practical ThermoGater Policies

We next devise practical ThermoGater policies to orchestrate thermally aware regulator gating following the closed control loop from Fig. 2.3. We closely mimic oracular policies from Section 2.6.2 and factor in the practical limitations.

We start with $Prac_T$, which corresponds to $Orac_T$ and gates regulators based on spatial thermal information only. Mimicking $Orac_T$, in selecting n_{on} regulators to ensure operation at η_{peak} (on a per V_{dd}-domain basis), $Prac_T$ ranks regulators by their anticipated temperature, turns on n_{on} of the regulators with coolest anticipated temperature, and turns the rest off. To this end, $Prac_T$ should be able to predict the anticipated temperature of each regulator as a function of changes in the power demand throughout execution.

In predicting the anticipated temperature of each regulator, $Prac_T$ assumes a linear relationship between changes in power dissipation (ΔP) and changes in temperature (ΔT) between two decision points, on a per regulator basis:

$$\Delta T_i = \theta_i \Delta P_i \qquad\qquad i = 1, 2, ..., 96 \tag{2.2}$$

where θ_i represents a constant of proportionality (or model fitting parameter) that we extract for each regulator i using (temporal) power and thermal traces from a profiling pass. As pointed out in Skadron et. al. [64] such linear models often fail short of accurately capturing the on-chip thermal profile. However, using HotSpot, we validated that *confined* deployment of this model only to predict the anticipated temperature of on-chip regulators (considering their small footprint) provides highly accurate results. ΔP_i captures the anticipated change (between two decision points) in the power dissipation (i.e., in $P_{loss,i}$) of regulator i, which is dictated by the anticipated change in the power demand of load circuit blocks (i.e., $P_{out,i}$) per Eqn. 2.1. Accordingly, $Prac_T$ tracks $\Delta P_{out,i}$ in determining ΔP_i.

To find the accuracy of prediction for Eqn. 2.2, we use *coefficient of determination*, R^2, a common statistical metric to quantify predictability [65]:

$$R^2 = 1 - \frac{\sum_i (T_{i,HotSpot} - T_{i,Prediction})^2}{\sum_i (T_{i,HotSpot} - T_{avg,HotSpot})^2} \tag{2.3}$$

applies where $T_{i,HotSpot}$ and $T_{i,Prediction}$ denote the T of regulator i obtained from Hotspot simulation and predicted using Eqn. 2.2, respectively. $T_{avg,HotSpot}$ captures

the average regulator T obtained from HotSpot. If the prediction error is absolutely zero, the numerator of Eqn. 2.3 goes to zero, leading to $R^2 = 1$. So, a more accurate prediction implies a closer-to-one R^2. We calibrate θ_i values to keep R^2 around 0.99.

Thermal sensors are widely used in modern commercial processors. For instance, IBM POWER7 employs 44 digital thermal sensors to detect chip wide hotspots [52]. Thermal sensors capable of providing up to 10K thermal readings per second exist [53]. For this type of sensor, in the worst case, thermal readings $Prac_T$ uses at each decision point would be 100μs old. We place such a thermal sensor at immediate vicinity of each regulator, and assume that the overhead of gathering and sorting sensor readings is comparable to the sensor delay (100μs in this case) by relying on microcontroller firmware such as POWER8 On-Chip Controller (OCC) [44].

To be able to predict the anticipated temperature by using Eqn. 2.2, $Prac_T$ needs a prediction for the anticipated power demand of the load circuit blocks (i.e., P_{out}), as well. To this end, we use the Weighted Moving Average (WMA) based model from [66], which can derive the anticipated power consumption from the power consumption history spanning the last three decision points.

Putting it all together: $Prac_T$ deploys Eqn. 2.2 at each decision point as follows: θ_i values are extracted from a profiling pass and do not change if the floorplan is fixed. $Prac_T$ first collects the instantaneous readings from the thermal sensors. Let the reading be $T_{t,i}$ for regulator i at decision point t. $Prac_T$ also estimates the anticipated power demand from the power demand history of the last three decision points, and calculates ΔP_i from the difference between the anticipated demand and the demand at the previous decision point. The next step is deploying Eqn. 2.2 to derive the anticipated temperature for each regulator i from $T_{t,i} + \theta_i \Delta P_i$. Recall that the anticipated temperature corresponds to the temperature the regulator would assume if it was turned on (until the next decision point). Finally, $Prac_T$ sorts the anticipated temperatures and picks the n_{on} of the regulators with the lowest anticipated temperatures to turn on, on a per V_{dd}-domain basis.

Factoring in all sensor and control related overheads, Fig.s 2.10 and 2.9 report the maximum thermal gradient and T_{max} under $Prac_T$. We observe that the thermal profile under $Prac_T$ slightly degrades in comparison to $Orac_T$ mainly due to the thermal sensing delay and prediction error from Eqn. 2.2: on average, the maximum thermal gradient

increases by ≈3%; T_{max}, by 0.5°C over $Orac_T$. As it is the case for $Orac_T$, gating only based on thermal information renders a poor voltage noise profile under $Prac_T$, as Fig. 2.11 reveals, increasing the overall maximum by 79.9% in comparison to *all-on*.

We add voltage-noise awareness to $Prac_T$ by mimicking $Orac_{VT}$, and call the resulting policy $Prac_{VT}$, which needs to predict the onset of voltage emergencies. As demonstrated in [55], voltage emergencies are predictable with more than 90% accuracy. $Prac_{VT}$ deploys a voltage emergency detector per core similar to [55] instead of alternative spatial voltage emergency sensors [54]. Further, $Prac_{VT}$ turns all regulators of the affected domain on upon a voltage emergency alert. This policy relaxes the power conversion efficiency constraint by possibly turning on more than n_{on} regulators, where n_{on} regulators would be necessary to operate at η_{peak} (on a per V_{dd}-domain basis). However, since emergency events are rare (Table 2.2), and $Prac_{VT}$ only turns on all regulators within a few critical domains upon an alert, the power conversion efficiency η degrades negligibly, by less than 0.1% on average, with the maximum degradation reaching 0.5% for *barnes*. When compared to $Prac_T$, $Prac_{VT}$ improves the voltage noise profile significantly as Fig. 2.11 reveals: overall maximum voltage noise stays at 13.22% of the nominal V_{dd} under $Prac_{VT}$; at 13.05%, under *all-on*.

In conclusion, $Prac_{VT}$ can effectively sustain operation within 0.5% of the peak efficiency η_{peak}, while degrading the maximum thermal gradient by around 3%; T_{max}, by 0.5°C over $Orac_T$, and the maximum voltage noise by less than 1.3% over *all-on*, subject to the accuracy of sensors and the predictive model from Eqn. 2.2, including calibration. Parametric variation due to manufacturing imperfections may render per-chip calibration necessary and can increase manufacturing testing overhead, but $Prac_{VT}$ is ranking-based and can tolerate calibration errors as long as inaccuracies keep relative ranking intact (where absolute parameter values may fluctuate significantly).

2.6.4 Design Space Exploration

The evaluation so far assumed an Intel FIVR [24, 47, 23] like regulator design in calibrating the regulator power conversion efficiency curves according to Fig 2.5. In this case, each distributed component regulator (as depicted by small squares, 96 in total, in Fig. 2.4b) corresponds to a phase. Our observations and ThermoGater policies, however, are equally applicable to different types of regulators, as well.

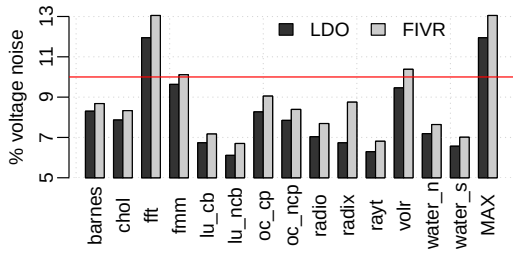


Figure 2.15: Maximum voltage noise: LDO vs. FIVR.

We repeat our analysis for an LDO based design similar to IBM's POWER8 [25, 44]. In this case, each distributed component regulator represents a digital LDO (micro)regulator. The reported η_{peak} and P_{out} per area assume very similar values in comparison to FIVR[6]. For an apples-to-apples comparison, we calibrate this LDO based design to follow the efficiency curves from Fig. 2.5 ([25, 44] do not report the efficiency curves).

Due to very similar P_{out} per area values (and because we calibrate both designs to render the very same power conversion efficiency curves under gating), the LDO based design's thermal profile closely tracks the FIVR based design. The main difference comes from the faster response time of LDO regulators, which is anticipated to lead to lower voltage noise.

Fig. 2.15 compares the maximum voltage noise for both of these designs across all benchmarks, if all component regulators stay on all the time (*all-on*). The LDO based design decreases the voltage noise by around 0.7% on average, while the overall maximum (for *ftt*) decreases by around 1.1% over the FIVR based design. This small improvement in voltage noise did not render any notable deviation from our results or observations in Section 2.6.

For FIVR, the power loss on the on-package inductor does not directly contribute to on-chip heating. Recall that we rely on FIVR only in calibrating the regulator power conversion efficiency curves. The only reason why we used FIVR for calibration was the public disclosure of the corresponding efficiency curves, as best-known representatives

[6] η_{peak}=90.5%, P_{out} per area = 34.5W/mm^2 for this case; η_{peak}=90%, P_{out} per area = 33.6W/mm^2 for FIVR.

from industry. Otherwise, ThermoGater decisions are governed by the *effective* curve from Fig. 2.5 which takes a very similar form for both (FIVR like) buck- and LDO-based designs. Note that a given (η, I_{out}) point on this curve can result in a different number of active regulators for buck- vs. LDO-based designs. This is where the inaccuracy in our modeling comes from, which is unlikely to change our fundamental observations.

Chapter 3

POWERT Channels: A Novel Class of Covert Communication Exploiting Power Management Vulnerabilities

3.1 Introduction

Modern computing platforms are fundamentally power limited [22]. This gives rise to sophisticated runtime power management – spanning several software and hardware layers of the system stack – in order to meet diverse and demanding runtime performance needs within the stringent power budget. Effective power management requires a highly *observable* and *controllable* system, at a very fine granularity in both space and time. Observability is necessary to be able to timely *sense*; controllability, to be able to timely *react*, to changes in the instantaneous power consumption of the overall system. Activity monitors in the form of performance counters or sensors dispersed across chip serve the purpose. Exposing fine grain hardware knobs for power management to the software layers of the system stack can help, as well, as it is the case for Intel's Running Average Power Limit (RAPL) interface [67].

By distributing the power budget carefully among active tasks of execution, runtime

power management has to guarantee that the system-wide power consumption never exceeds the system-wide power budget. The power budget itself represents a very critical shared resource. If not carefully orchestrated, any shared (hardware or software) resource can easily enable information leakage via covert communication [68, 69, 70]. As a fundamental shared resource, power budget unfortunately does not represent an exception. The abundance of specialized activity monitors, their exposure to software layers, and the need for tight global control to avoid power budget overshoots, just exacerbate the situation.

In this study, we introduce, demonstrate, and characterize a novel class of covert communication over previously unexplored channels triggered by power management vulnerabilities. In the following, we will refer to this novel class as POWERT channels. Key contributions of this study include:

- Detailed analysis of covert communication over POWERT channels; a novel, previously unexplored class of covert channels induced by power management vulnerabilities;
- Demonstration of POWERT communication on two commercial systems;
- Comprehensive characterization of the POWERT channel capacity.

In the following, Section 3.2 covers the basics of POWERT communication; Section 3.3, channel capacity; Sections 3.4 and 3.5, the evaluation; Section 3.6, countermeasures; and Section 3.7, a compare and contrast to related work.

3.2 POWERT Communication Basics

We next characterize POWERT communication enabled by power management vulnerabilities. We start with basic definitions and the threat model in Section 3.2.1, and continue in Section 3.2.2 with inevitable power management practices that enable POWERT channels. Section 3.2.3 then provides a conceptual explanation of how such power management practices facilitate covert channels. We conclude by demonstrating a proof-of-concept POWERT attack on a commercial system in Section 3.2.4.

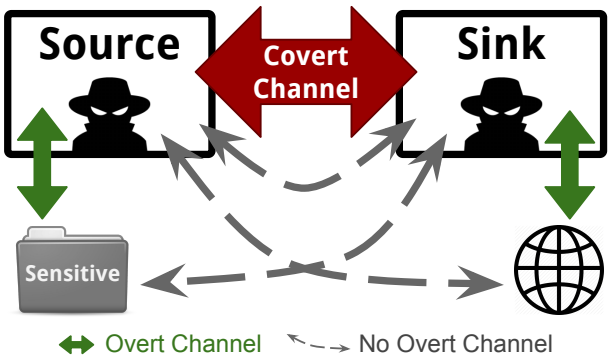


Figure 3.1: Threat model.

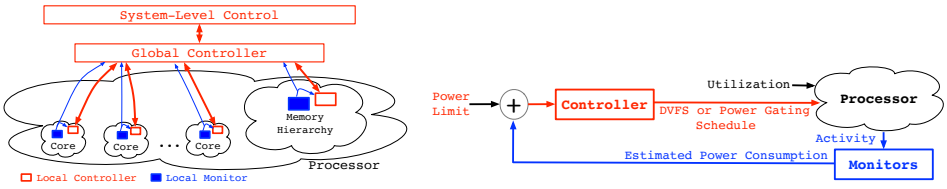
3.2.1 Threat Model

Fig. 3.1 depicts two malicious entities (the *source* and the *sink*, respectively), not permitted to communicate legitimately through *overt* channels, performing a *covert* channel attack. Without loss of generality, these entities may correspond to hardware (such as cores or functional units), or software sharing hardware resources [71]. The transmitting end, the *source*, has access to sensitive information (such as a secret key), however, not to any overt channel for data communication. The receiving end, the *sink*, on the other hand, has access to an overt channel for potential data communication, but not to sensitive information. By communicating with the source over the *covert* channel, the sink can not only acquire access to sensitive information, but also send it to third parties subsequently over the overt channel. As a representative example, the source can be a contacts manager, and the sink, a weather application, in a mobile system [71]. By construction, such covert communication is hidden from other hardware or software entities sharing the same system.

3.2.2 Power Budget: A Critical Shared Resource

Modern power-limited computing platforms benefit from sophisticated power management in two distinct ways:

(i) Prevention of serious power budget overshoots, which can physically damage the system;



(a) Macroscopic view (b) Microscopic view

Figure 3.2: Hierarchical on-chip power management and control, adapted from [72].

(ii) Optimal distribution of the shared power budget among active tasks of execution, to satisfy possibly conflicting performance requirements in a fair and efficient manner.

Fig. 3.2a illustrates an overview of inevitable power management practices in modern systems. System-level control at the hardware-software interface directs a global controller, which in turn orchestrates local controllers dispersed across chip. These local controllers periodically evaluate monitored local activity to adjust the operating point (i.e., the operating voltage or frequency). Such distributed control is becoming more and more common, as local controllers can react to local changes in the instantaneous power demand much faster. If a reallocation of the system level power budget becomes necessary, system-level control alerts the global controller, which in turn makes the local controllers adjust the local operating voltage and frequencies accordingly.

Fig. 3.2b depicts a generic control loop, which is equally applicable to both global and local controllers in Fig. 3.2a. In this case, be it local or global, the controller modulates the operating point as a function of the power limit provided at its input. The goal always is delivering the maximum possible performance without violating this power limit. Various options exist for operating point modulation, including adjustments to the operating voltage and frequency (via DVFS, dynamic voltage and frequency scaling), selective shut-down of idle resources (via power gating) or both.

To summarize, system-level control imposes an instantaneous power budget, which the global controller has to meet via orchestrating local controllers. Local controllers in turn enforce the necessary adjustments to local operating points in their respective control domain. At the same time, the impact of such adjustments on the instantaneous power consumption gets instantly monitored to prevent power budget overshoots.

3.2.3 Power Headroom Modulation (PHM)

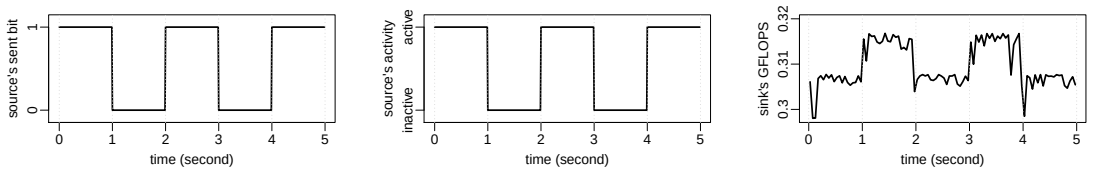
Following the threat model from Fig. 3.1, let us assume that a source application, which has access to sensitive information, shares processor resources with a sink application. For example, the source and the sink may run on a mobile chip, possibly along with other applications.

Controllers usually modulate the operating point (by, e.g., voltage or frequency scaling) periodically. This is because activity monitors time-sample the system at regular intervals. The period of such adjustments is a function of the temporal monitoring granularity of activity monitors along with the latency across the power/clock distribution networks and of voltage regulators [72]. The period t_{PM} is usually in the order of several processor clock cycles. By construction, the source and the sink are very well aware of this period. Moreover, typical power management algorithms are of predictive nature and extrapolate predictions from a history of monitored activity which are t_{PM} apart in time from each other.

If no other application resides in the system, but the source and the sink, the source can easily modulate the share of the power budget available to the sink, to encode binary information. In the following, we will refer to the sink's share of the power budget as the *power headroom*. This is possible simply because the chip-wide power budget, as well, is a shared resource. The source can easily control its own activity, and thereby, its own power consumption. Then, what is left to the sink is the power budget minus the source's power consumption.

The procedure is straight-forward: To encode a logic 1, the source can run a very power hungry virus. In this manner, the source can reach its own power budget limit, and taint its local activity history to trick the global controller. The sink can distinguish this case from no activity (hence, low power consumption) at the source, as the controllers will very likely trigger emergency power throttling at the sink – to prevent the instantaneous (system-wide) power consumption exceed the available budget as a result of excessive consumption at the source. Under no activity at the source, on the other hand, such throttling events at the sink become much less likely. In this particular example, the sink decodes an almost constant, sizable positive power headroom on its side, as a logic 0; and any power headroom change (caused by source's activity) as a logic 1.

Other applications sharing the same processor resources can challenge this type of



(a) bits sent by source (b) source's activity (c) sink's performance (GFLOPS)

Figure 3.3: POWERT attack.

covert communication between the source and the sink. Inevitably, sharing induces noise in the covert channel. This is because the sink quantifies its available power headroom, i.e., decodes the source's message bit by bit, by tracking the throttling events.

3.2.4 Anatomy of POWERT Attacks

In the following, we will refer to the entire hierarchy of the controllers in charge of the power management, as depicted in Fig. 3.2, as the *Power Manager* (PM). In a typical multi/many-core, the power budget cannot accommodate all cores operating at the peak performance point at the same time. Hence, when multiple cores run compute-intensive workloads simultaneously, PM has to assign a lower operating frequency to them than the rated peak frequency, in order to meet the power budget. We will next characterize a POWERT attack exploiting this inevitable behavior of PM.

Usually, when only one compute-intensive application is running on one of the cores, PM lets that core run at the rated maximum frequency. The common outcome for two compute-intensive applications running at the same time on two different cores is a lower frequency than the rated maximum enforced on both cores. Thereby, an application's activity pattern can directly affect the performance of other applications running on the system. Applications like the source and the sink from Fig. 3.1 can hence rely on this phenomenon to communicate with each other covertly, by affecting the operating frequency and/or voltage, hence the performance, of each other.

Let us next take a closer look into a POWERT attack: The source and the sink are both compute-intensive applications. The source sends a "1" through the covert channel by running a compute-intensive workload, and a "0", by going into sleep. In order to

capture source's activity pattern, the sink constantly runs a compute-intensive workload. As a result, PM slows down the sink when the source is sending a "'1" (i.e., running a compute-intensive workload), compared to when the source is sending a "0" (i.e., going into sleep mode). The sink therefore can retrieve bits sent by the source by just tracking its own performance. To measure its own performance, the sink can simply periodically check its own progress. Neither the sink, nor the source does need any system level privilege to this end, which challenges capturing (and potentially blocking) the attack.

Fig. 3.3 demonstrates a POWERT attack, where the source sends 5 bits through the covert channel at a communication rate of 1bit/s[1], as shown in Fig. 3.3a. The source becomes active when sending a "1", and goes to sleep otherwise. Fig. 3.3b captures source's activity pattern corresponding to the sent bits. Finally, Fig. 3.3c depicts sink's performance, as measured by the sink itself. Y-axis represents GFLOPS (Giga floating point operations per second); the x-axis, time. In this particular example the compute-intensive sink application comprises a floating point heavy loop, therefore, GLOPS is a good proxy for checking the rate of forward-progress at the sink. We observe that the GFLOPS rate of the sink decreases by around 2.5% on average when the source is active (sending "1"), compared to when the source is sleeping (sending "0"). The sink can therefore retrieve information from this covert channel by simply checking its GLOPS rate. We fully characterize such POWERT attacks on different commercial platforms in Section 3.5.

3.3 Channel Capacity

Power Manager, PM, constituting the entire hierarchy of controllers from Fig. 3.2, orchestrates power consumption of different entities, where each entity (e.g., cores, caches or routers) can operate at a different voltage and frequency. Let us assume that PM is in charge of N different entities, where P_i depicts the maximum possible power consumption for entity i, i ranging from 1 to N. All entities cannot have operating points to reach their peak power consumption simultaneously, without overshooting the

[1]We pick a relatively low communication frequency here to ease illustration, and explore higher frequency ranges in Section 3.5.

power budget (Section 3.2.4). Therefore,

$$\sum_{i=1}^{N} P_i > power\ budget$$

applies. However, when the system is not highly utilized, most of the entities become idle and can be power-gated. In this case, PM can let the few active entities operate at their peak power consumption, P_i, as long as the overall power consumption stays below the power budget. As utilization increases, more entities require to be active at the same time, in order to meet performance goals. Even under less than 100% utilization, the overall power consumption with all active entities running at P_i may violate the power budget. To avoid budget violation, PM has to force all active entities to a lower-power operating point, L_i, in this case, with $L_i < P_i$.

As an example, let us assume that a PM has to manage the power consumption of four cores in a multi-core environment, each consuming 15 Watts at peak. Now, let us assume that the overall chip power budget is only 40 Watts. In this case, if only two of the cores are active, the PM still can let them run at their peak power consumption, since the overall power consumption does not reach the power budget. However, consider a scenario in which all four cores are active. This time, they would consume 60 Watts in total, if they were running at peak at the same time, which violates the power budget of 40 Watts. Therefore, the PM can let each core consume at most 10 Watts, which leads to a relative slow-down (with respect to the peak rated performance point) in all four cores.

It is this type of inevitable PM decisions that give rise to POWERT channels via power headroom modulation (Section 3.2.3). A malware (such as the source from Fig. 3.1) can send information covertly to another malware (such as the sink from Fig. 3.1) by modulating the receiving side's share of the power budget, i.e., power headroom, and consequently, performance. The source can also activate a number of entities, enough to violate the power budget, in order to send a "1". In this case, PM has to lower the share of the power budget of all entities, including the sink at the receiving end. The sink in turn can retrieve the sent bit (a "1"), by sensing a slow-down in its own performance. Similarly, to send a "0", the source can put multiple entities to sleep to minimize the chance of violating the power budget (hence, of the sink being throttled).

At the receiving end, the sink does not sense a slow-down in this case, and translates this to having received a "0".

POWERT communication via power headroom modulation has four phases:

- The first phase spans the time window t_{Util}, over which the source enforces changes in the activity, i.e., utilization, by modifying the number of active entities.
- The second phase covers the time window $t_{Monitor}$, over which activity monitors sense the corresponding change in the power consumption of the affected entities.
- In the third phase, PM senses the change in the activity by reading monitors and makes a decision about throttling. This phase usually takes place periodically (Section 3.2.3), with a period of t_{PM}.
- The last phase comprises two steps: The first step is the time it takes for the affected entities to adjust their operating voltage and frequency to meet the enforced power budget, t_{Adjust}. The second step is the time it takes for the sink to sense the changes in its own performance, t_{Sense}.

These four phases together span the duration of communication, starting from sending a single bit until sensing it at the receiving end. We can also overlap, i.e., pipeline, these phases: For example, after PM reads the monitors (phase two) and makes a new decision (phase three), the source can start sending the next bit (phase one). This is because the sink has enough time to sense the changes (phase four) until the next PM decision takes place. Therefore, the first two phases together, the third phase, and the fourth phase can form three distinct stages of a pipeline, to accelerate POWERT communication. In Section 3.5, we will assume such pipelined POWERT communication in deriving an upper-bound for the communication rate, F_{MAX} as

$$(max(t_{Util} + t_{Monitor}, t_{PM}, t_{Adjust} + t_{Sense}))^{-1}.$$

PM manages N different entities, each with variable voltage and frequency. As finer grain on-chip voltage regulation is becoming more common, the number of entities (i.e., N) that PM can manage independently keeps increasing. As an example, N becomes 48 for POWER8 processors [73]. This also necessitates hierarchical PMs as depicted in Fig. 3.2, as a single centralized PM is very likely to result in sub-optimal power management – both due to the increasing latency of collecting data from all monitors and the complexity of solving a larger optimization problem with increasing N.

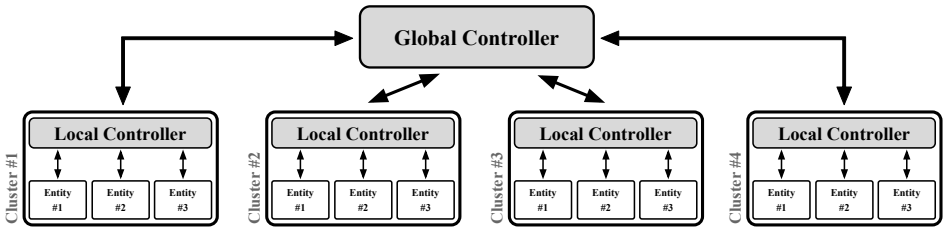


Figure 3.4: Two layer hierarchical power management.

Fig. 3.4 depicts a two-layer example, which closely mimics the general structure from Fig. 3.2. A global controller sets the power budget of 4 different clusters, each containing 3 entities, where the respective local controller of each cluster distributes the assigned power budget among the three entities.

The sink and the source can reside in the same cluster. Under *intra-cluster* covert communication, the source forces the local controller of the cluster to throttle the performance of other entities within the cluster according to the local power budget. The upper-bound of intra-cluster covert communication rate, $F_{MAX,intra\_cluster}$, hence becomes

$$(max(t_{Util} + t_{Monitor}, t_{PM_{local}}, t_{Adjust} + t_{Sense}))^{-1}$$

where $t_{PM_{local}}$ is the local controller's decision period.

The sink and the source can be in separate clusters, as well. Under *inter-cluster* covert communication, the source should increase the corresponding cluster's power consumption (or even, the power consumption of multiple clusters) to the point where the global controller has to limit the power budget of all active clusters, including the one containing the sink. Then, the local controller of sink's cluster adjusts the share of the power budget of each entity of the cluster accordingly, which inevitably leads to noticeable performance degradation at sink. Clearly, inter-cluster covert communication is slower than intra-cluster. An upper-bound for inter-cluster covert communication rate, $F_{MAX,inter\_cluster}$ hence becomes

$$(max(t_{Util} + t_{Monitor} + t_{Comm}, t_{PM_{global}},$$

$$t_{Adjust} + t_{Sense} + t_{Comm}))^{-1}$$

	Intel Xeon E3-1505M v5	Samsung Exynos-5422	
μarchitecture	Skylake family	Cortex-A15 (big)	Cortex-A7(little)
# of cores (threads)	4 (8)	4 (4)	4 (4)
technology node	14 nm	28 nm	
frequency	(0.8-2.80) GHz	(0.2-2.0) GHz	(0.2-1.4) GHz
L1 Inst.	32KB 8-way	32KB 2-way	32KB 2-way
L1 Data	32KB 8-way	32KB 2-way	32KB 2-way
L2	256KB 4-way	2MB 16-way	512KB 8-way
L3	8MB 16-way		

Table 3.1: Evaluated systems.

where t_{Comm} is the local to global controller communication latency; and $t_{PM_{global}}$, global controller's decision period.

3.4 Evaluation Setup

3.4.1 Evaluated Systems

As a proof of concept, we characterize POWERT communication on two commercial platforms: a laptop machine featuring an Intel Xeon E3-1505M v5 and Ubuntu 14.04.5 and an ODROID-XU4 board, featuring a Samsung Exynos-5422 with a processor based on ARM's big.LITTLE architecture [74] and Ubuntu 16.04.3 LTS (Table 3.1). Both source and sink represent floating-point heavy applications (Section 3.4.2). We compile the sink and source application using GNU GCC version 4.8.4 on the laptop platform, and GNU GCC version 5.4.0 on the ODROID board, with all optimizations disabled. To maximize energy efficiency, the default mode of the operating system's power manager, *on demand*, allocates (by asking hardware PM) the maximum possible frequency to the source (during active phases) and the sink. Hence, to make sure that operating system's PM does not affect our experiments, and indeed hardware PM is making the throttling decisions (which is harder to block, as explained in Section 3.6), we do not change the default PM of the operating systems on both machines. Besides, to minimize the impact of background noise (in order to better characterize the channel capacity), we turn off unnecessary OS services.

```
// gets called when timer overflows.
sigalrm_handler ( ) {
   Print_to_file ( Loop_Counter );
   setitimer ( t_Sample ); //sets interrupt timer.
}

Main ( ) {
   setitimer ( t_Sample ); //sets interrupt timer.
   //infinite while loop
   while (1) {
        Loop_Counter++;
        Run_Float (); //runs multiple fp instructions.
   }
}
```

Figure 3.5: Sink application's code.

```
// gets called when timer overflows.
sigalrm_handler ( ) {
   Data_Index++;
   // activates MPrime, to send a 1
   if Data[Data_Index]=1 && Active=0 {
        system ( "kill -CONT MPrime_PID" );
        Active = 1;
   }
   // stops MPrime, to send a 0
   if Data[Data_Index]=0 && Active=1 {
        system ( "kill -TSTP MPrime_PID" );
        Active = 0;
   }
   setitimer ( t_Covert ); //sets interrupt timer.
}

Main ( ) {
   setitimer ( t_Covert ); //sets interrupt timer.
   while (1); //infinite while loop
}
```

Figure 3.6: Source application's code.

3.4.2 Malware Codes

Fig. 3.5 depicts the sink code, which resides on the receiving end of the covert channel. The sink constitutes an infinite floating-point heavy loop. The sink has full-fledged control over the mix and count of the executed floating-point instructions within `Run_Float()`. The sink samples the channel every `t_Sample` seconds and calls `sigalrm_handler` function. We set `t_Sample` to be 20 times smaller than the (known-to-both-sides) communication period `t_Covert`, which leads to 20 samples per bit. Periodic interrupt timer overflows invoke `sigalrm_handler` function, with a period of `t_Sample`. Inside `sigalrm_handler` the sink dumps the loop counter variable, `Loop_Counter`, to an output file which serves as a proxy for the rate of forward progress. The sink extracts its GFLOPS rate periodically from `Loop_Counter`. A third-party application after receiving this file can also retrieve the data communicated over the POWERT channel.

On the other end of the POWERT channel, as explained in Sect. 3.2.4, the source runs a highly power hungry application. On the Intel-based platform, we use the latest version of MPrime [75] (v2810) to this end, specifically, the `Torture Test` mode to maximize the power consumption in active phases. We use cpuburn-a7 [76] to maximize power consumption on ARM-based platform. Fig. 3.6 depicts the code of the source application. Similar to the sink, the source sets an interrupt timer for accurate timing of communication. Every `t_Covert` seconds, which represents the communication period (known to both sides), `sigalrm_handler` function gets invoked. Inside `sigalrm_handler` the source changes highly power hungry application's status from running to idle or vice versa, according to the value of the next bit to be sent.

3.4.3 Communication Protocol

The source sends data in 100-bit long packets, at a rate known to the sink – every `t_Covert` seconds following Section 3.4.2. At the same time, each packet starts with a 5-bit preamble which is known to both sides, and the third party receiving data from the sink, as well. The preamble specifically is used by the third party application to synchronize with the sink and to de-noise retrieved information (Section 3.4.4). To synchronize with the sink at the start of communication, the source, on the other hand, sends a long bit-stream of interleaved ones and zeros followed by a short bit-stream of

zeros only (both of known lengths). The sink periodically probes the channel with a shorter period than the duration of the ones and zeros in this header. During probing, when the sink receives these ones and zeros, it remains active and waits for the short bit-stream of zeros to arrive. At the end of the short bit stream of zeros, the sink starts to dump loop counter information to its file (according to Fig. 3.5), as the following bits sent by the source are the actual data packets.

3.4.4 Communication with Third Party

As explained in Section 3.2.1, the sink has access to the network to send the retrieved information to a third party application. The third party application is responsible for decoding the information sent by the sink, i. e., 100-bit long data packets, from a signal similar to the one shown in Fig. 3.3c. As explained in Section 3.4.2, the sink records 20 loop count samples per each bit sent by the source. The third party application therefore can use the median of every 20 samples to represent each bit, in order to remove noise from the data, and subsequently compare each median value to a threshold, Bit_{THR}, to find the actual bit value.

The third party application can extract retrieved information from the sink bit by bit, even if the sampling rate deviates from the expected 20 samples per bit. To this end, the third party application can simply try a few potential sample rates to decode the known preamble (the first 5 bits of every 100 bit data package) and record the corresponding error, to settle at the sample rate which results in the minimum error. A similar method applies for extracting the Bit_{THR}. The third party in turn can use this sample rate and Bit_{THR} to decode the actual information. We implement the third party application in R version 3.4.0.

3.4.5 Design Space Exploration

We explore different POWERT attack scenarios to find the maximum achievable covert communication rate on the evaluated systems. To minimize simulation noise, we pin the source and the sink applications to specific cores during the entire execution. We also experiment with different number of source and sink applications, and characterize the channel capacity for each case.

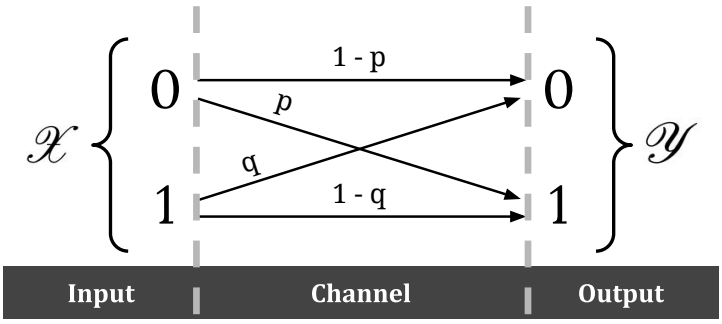


Figure 3.7: Binary asymmetric channel model.

3.4.6 Channel Capacity by Shannon Theorem

We next discuss how to quantify the actual channel capacity from measurements on a given system. Following the methodology from [77], we quantify channel capacity as the theoretical maximum possible communication bit-rate according to Shannon's theorem [78]. We model the POWERT channel as a memoryless binary asymmetric channel, as shown in Fig. 3.7. $\mathcal{X} = \{0, 1\}$ represents the input alphabet; $\mathcal{Y} = \{0, 1\}$, the output alphabet, respectively. From actual measurements we can find p, the probability of sending a 0 and receiving a 1, and q, the probability of sending a 1 and receiving a 0.

Shannon's well-known theorem defines the theoretical maximum for channel capacity (per channel use), C, as

$$C = max(I(X;Y)) \qquad \forall p(x) \qquad x \in \mathcal{X}$$

where the maximum of I, mutual information given X (input) and Y (output) is taken over all possible input distributions, $p(x)$. For instance, $p(x) = (0.5, 0.5)$ when the probability of sending a 0 or a 1 is the same. Knowing the number of bits we can send per each channel use, C, we can find the maximum number of bits per second that can be transmitted via covert communication using $C \times f$, where f denotes the communication frequency (rate).

3.5 Evaluation

We will next examine POWERT channel characteristics on the two commercial platforms.

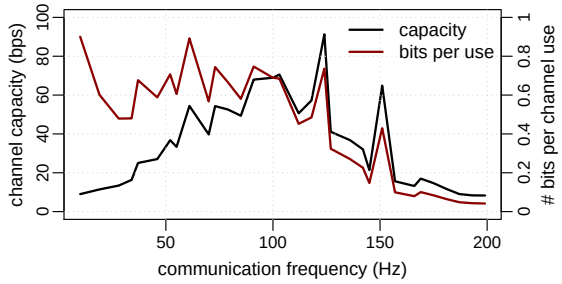


Figure 3.8: Single source to single sink communication.

3.5.1 Case Study I

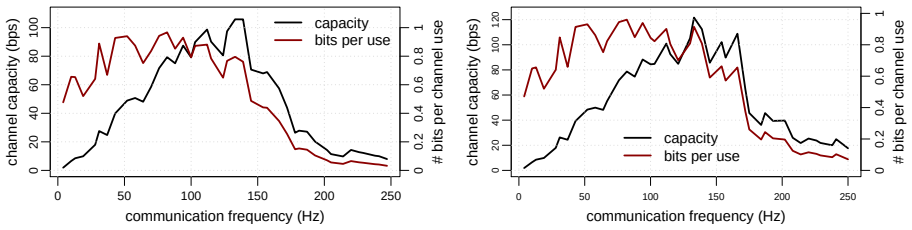
In this section, we evaluate POWERT communication on the Intel Xeon system (Section 3.4.1).

Analytical Upper Bound (Section 3.3): We can directly apply the model from Section 3.3 to this 4-core system to derive an upper bound for channel capacity, as follows (we will revisit this upper-bound using actual measurements in Section 3.5.1): First we extract t_{Util} of the evaluated system. We find t_{Util} by altering the activity status of MPrime from running to idle and the other way round continuously for a fixed period of time, t_{test}. Then, we count how many status changes happened during the t_{test} period, N_{Util}. We extract t_{Util} from t_{test}/N_{Util}, which is around 709.2μs for the evaluated system. We estimate $t_{Monitor}$ to be around 250μs [79]; and t_{PM}, around 1ms [80]. This system features on-chip voltage regulation with a t_{Adjust} of around 100-200ns [45]. Finally, t_{Sense} depends on the sink application's timer precision, which for the evaluated system is 1μs. Based on these parameters, the upper-bound for POWERT communication rate becomes

$$(max(709.2\mu s + 250\mu s, 1ms, 200ns + 1\mu s))^{-1} = 1\ Kbps.$$

Measurement-based Characterization: In this section we characterize covert communication through POWERT channels using the methodology from Section 3.4.5. We start with a scenario in which, a single source application communicates information covertly to a single sink application, each running on a separate core.

Fig. 3.8 shows channel capacity on left y-axis for different communication frequencies (x-axis). Besides, right y-axis represents C (Section 3.4.6), which captures the average number of bits we can send per each bit transfer attempt through the channel. For



(a) First Sink (b) Second Sink

Figure 3.9: Single source to two sink communication.

instance, C of 0.5 means that we can send only 0.5 bit of information on average, per each bit transfer attempt. Therefore, channel capacity (left y-axis) simply corresponds to C (right y-axis) multiplied by per bit communication frequency (x-axis).

As Fig. 3.8 depicts, for lower than around 100Hz frequencies, we see a relatively constant C of around 0.65 on average. However for larger than 100Hz, C starts to drop since many of the OS background tasks, the main source of the background noise for POWERT channels, have similar activity rates. The peak communication rate we observe in this scenario is about 91.3 bps (which, as expected, is less than the communication rate cap of 1Kbps we derived previously) at a communication frequency of around 124.1 Hz. In this case, we observe a difference of 2.56% in sink's GFLOPS rate when the source has sent a zero (idle source) vs. a one (active source).

One Source, Multiple Sinks: A higher number of active cores decreases the available power headroom, and consequently, increases the likelihood of throttling. To quantify this effect, we increase the number of active cores by instantiating multiple copies of the sink application. Each of the sink applications stays *constantly* active, running the same floating point heavy loop. Therefore, after initialization, GFLOPS rate of each sink instance can only primarily evolve as a function of source's activity, and not of other sinks'.

Fig. 3.9 depicts channel characterization curves for the case where two instances of the sink application receive information from the same source. Each of the sinks and the source are running on three separate cores of the evaluated system. Fig.s 3.9a and 3.9b characterize the two different sinks separately. We observe that C of both sink

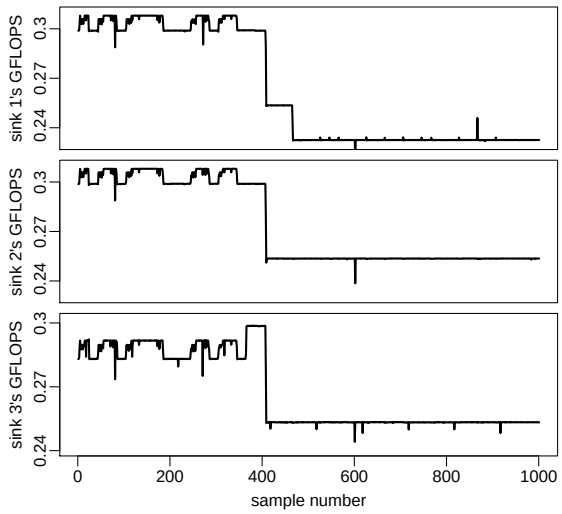


Figure 3.10: Single source to three sink communication.

applications remains above 0.8 for frequencies lower than 130Hz, and starts to fall for frequencies greater than 150Hz. One sink application achieves a peak channel capacity of around 121.6bps; the other one, of 105.8bps; both at a similar communication frequency of around 133.0Hz. The difference between GFLOPS levels of each sink application becomes around 2.67%, which is 4.6% larger than the single sink outcome. As a larger gap between GFLOPS levels of each sink (leading to easier decoding by the third party application) makes communication more robust to noise, we observe the peak channel capacity at a higher frequency in this case. At the same time, C at low frequencies is higher on average when compared to the single sink case.

When we increase the number of sink applications to 3, which activates all 4 cores of the evaluated system, we observe that C drops significantly over all communication frequencies. The reason is that, as all cores of the processor become active at the same time, PM has to enforce a strict power budget across the board. Fig. 3.10 depicts this situation for an example communication window of 1000 GFLOPS samples for each of the three sink applications. Y-axis represents GFLOPS rate; x-axis, sample numbers. The first 400 samples reveal that activity changes of the source is directly visible in the GFLOPS rate of all three sinks (the difference in GFLOPS levels is around 2.86%, larger than both of the previous scenarios). However, after the first 400 samples, PM

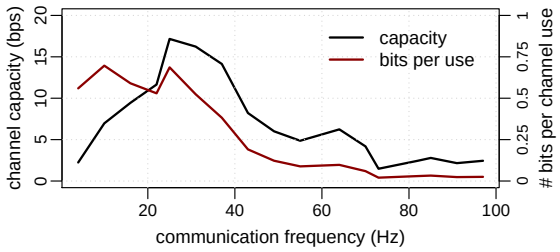
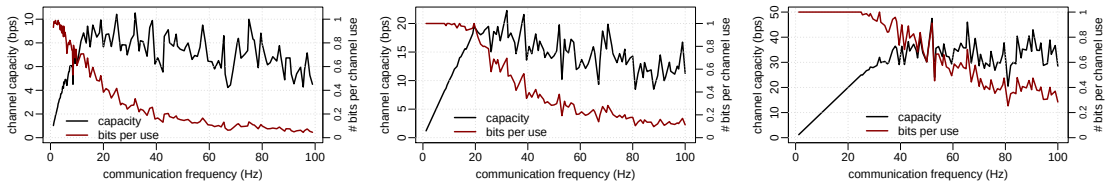


Figure 3.11: Two source to single sink communication.

suddenly enforces a lower power budget for all cores, which masks the impact of activity changes at the source on any of the sink applications' GFLOPS rate for the rest of this example communication window. Therefore, when all cores are utilized by malware applications, and consequently, PM is pushed towards its limits, covert communication through POWERT channels becomes infeasible.

Multiple Sources, One Sink: We conclude the first case study with POWERT communication when multiple instances of the source send the exact same data to a single sink application. Having multiple instances of MPrime getting activated and deactivated simultaneously increases the gap between power demand from the source when sending a one and sending a zero, which can result in a more pronounced difference between the two GFLOPS levels (corresponding to one and zero respectively) of the sink application.

Fig. 3.11 depicts channel characterization curves for two source applications and a single sink each running on a separate core. The peak channel capacity reaches around 17.2bps at a communication rate of around 24.9Hz, which is significantly lower compared to the single source scenario. Although the difference between GFLOPS levels at the sink is about 15.2% on average, channel capacity remains relatively low, since we fail to accurately synchronize the two instances of the source application, MPrime. In other words, since we rely on Linux `kill` to pause and continue the execution of each MPrime instance, and since `kill` has high and non-deterministic latency, without any intervention, the two instances of MPrime simply run out of synch, even in lower frequencies. Therefore, as the communication rate increases, C drops. Introduction of (inevitably complex) synchronization methods can reveal higher peak channel capacity



(a) single source (b) two sources (c) three sources

Figure 3.12: Channel characterization within little cluster: single (a), two (b), three (c) source to sink communication.

under this scenario, which we leave to future work.

3.5.2 Case Study II

In this section, we evaluate POWERT communication on the ARM system (Section 3.4.1). ***Little-to-little POWERT Communication:*** We first characterize POWERT communication when the attack happens within the little cluster only. Fig. 3.12a depicts channel capacity (left y-axis) when we only have a single source application and a single sink, both running on distinct little cores. For lower communication rates (x-axis), C, the number of bits sent per channel use (right y-axis), is close to 1, indicating almost perfect communication. However, as the communication rate increases, C drops with a sharp slope. For this attack scenario, we observe a peak channel capacity of 10.5 bps for a communication rate of around 32.0 Hz.

For a single source application, the difference between GFLOPS levels of the sink (when the source is active vs. inactive) is around 0.13%. To improve this gap and consequently, increase the POWERT communication bit rate, we can increase the number of sources. Fig.s 3.12b and 3.12c demonstrate channel capacity of two and three source applications running on little cores, respectively. We observe that by increasing the number of source applications, POWERT attack bit rate increases as well. For two sources and a single sink, we observe a peak channel capacity of 22.3 bps at a communication frequency of around 32.0 Hz. Besides, for communication rates below 20 Hz, the channel is almost error-free (i. e. $C \approx 1$). For the attack scenario with 3 sources and a single sink, on the other hand, communication remains almost error-free

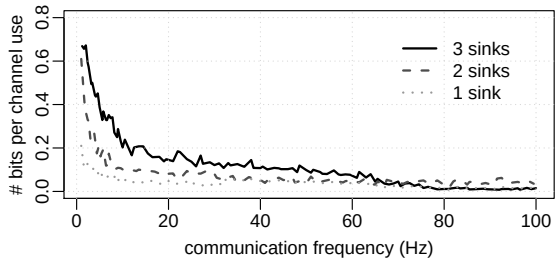
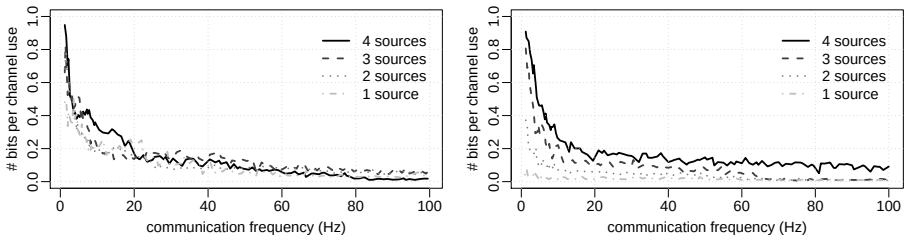


Figure 3.13: Channel characterization within big cluster.

for communication rates below 25 Hz. In this case, the peak channel capacity is around 47.5 bps. The gap between GFLOPS levels of the sink (when all sources are active vs. inactive) for the two and three source scenarios are 0.25% and 0.49% respectively. Although similar to the Intel-based platform, synchronization limits C when frequency increases, for low frequencies C stays high (due to a larger GFLOPS gap), leading to overall higher peak channel capacity compared to the single source case.

We also explore POWERT communication when we have multiple sinks and only a single source application, which leads to higher overall power consumption during the attack (but still, primarily the single source's activity modulates the power headroom of the sinks). We observe that for scenarios employing two and three sinks, the gap between GFLOPS levels of the sinks stays in the same range, 0.12% and 0.11%, respectively. Hence, peak channel capacity remains around the channel capacity for the single sink case (10.5 bps), 7.2 bps and 8.4 bps respectively, which shows that multiple sinks do not improve channel capacity.

Big-to-big POWERT Communication: We next characterize the POWERT channel when both the sink and the source applications only use big cores. Having a single source and a single sink, we see that the number of bits sent per channel use is close to zero, as shown in Fig. 3.13. In other words, covert communication is not feasible. The gap between the GFLOPS levels of the sink (when the source is active vs. inactive) is only around 0.11%. However, as we increase the number of sinks, we observe that the number of bits sent per channel use increases significantly. The gap between GFLOPS levels of the sink when we have two (three) sinks running in parallel increases to 0.40% (0.59%). We observe the peak channel capacity of around 5.8 bps at a communication



(a) big-to-little (b) little-to-big

Figure 3.14: Inter-cluster channel characterization.

rate of near 48.5 Hz, when we have three sinks running on big cores at the same time. Although the communication rate is very low, this analysis proves the existence of POWERT channels on big cores, as well. Since the power hungry application used as source in this case, cpuburn-17, is optimized for Cortex-A7 (little) cores, optimizing it for the big cores can increase channel capacity, which we leave to future work. As it was the case for the little cluster, we do not observe practical benefits by increasing the number of sink applications.

Big-to-little & Little-to-big POWERT Communication: We finally perform POWERT attacks, using sink and source applications on different types of cores, little or big. First, we put source(s) on big cores, and a sink on a little core. As Fig. 3.14a depicts, the number of bits sent per channel use, C, increases when we have multiple source applications running on the big cores at the same time. We observe a peak channel capacity of 5.5 bps at a communication frequency of around 40.5 Hz when four source applications are running on all four big cores. Besides, Fig. 3.14b demonstrates the number of bits per channel use for a scenario having different number of sources running on little cores, and a sink on a big core. Similar to the previous case, we observe that having more number of sources increases the number of bits we can send per channel use. We observe a peak channel capacity of 8.7 bps for four sources at a communication rate of around 61.0 Hz. While the channel capacity for this type of inter-cluster covert communication is not as large as the (intra-cluster) little-to-little communication, this analysis indicates that inter-cluster channels exist, and hence need to be considered when designing effective counter-measures.

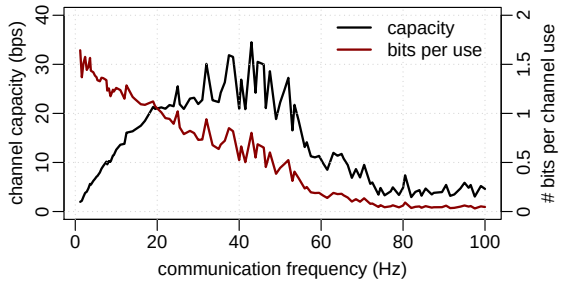


Figure 3.15: 2-bit encoding on little cores.

Multi-level Encoding We next explore how to encode information into four different levels of source's activity, enabling us to send two bits of information per channel use, instead of the simple 1-bit encoding using 0 and 1 only (which we covered so far). We are not able to adjust the activity level of the highly power hungry applications used in this study, however, we can still enforce different activity levels by changing the number of active sources. For example, to send a binary value of "10", we can activate two sources. In other words, we can encode a two-bit value into the number of active sources. This way, we can send more than one bit in each channel use, which can improve channel capacity.

As a proof-of-concept, we apply this multi-level encoding scheme to little cores, where we have observed the best channel profile when using multiple sources for POWERT communication (Fig. 3.12). As depicted in Fig. 3.15, we observe a peak channel capacity (left y-axis) of 34.5 bps at a communication rate (x-axis) of around 43.0 Hz in this case. While at low frequencies we observe around 1.5 bits sent per channel use (right y-axis), the C quickly goes down to near zero (i.e., no information sent) at frequencies above 70 Hz. This is not unexpected as we have observed a similar trend for only one or two sources running in Fig.s 3.12a and 3.12b, respectively. Hence, we cannot reach a channel capacity as high as the three sources case alone, as depicted in Fig. 3.12c.

3.6 CounterMeasures

3.6.1 Power Budget Isolation

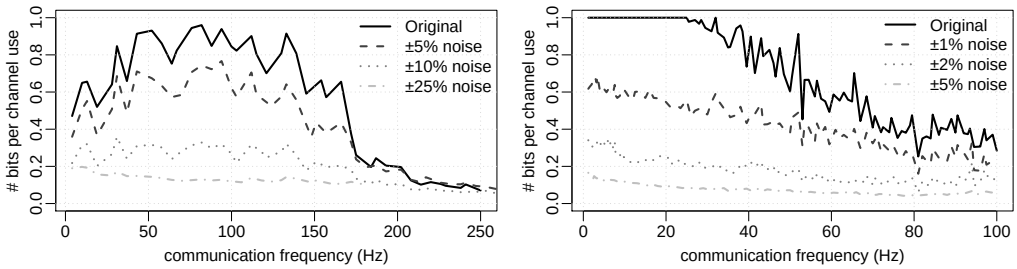
One way to avoid POWERT attacks is to assign a separate, fixed and safe power budget to each entity and thereby to exclude any power budget sharing. In this case, independent local power management is necessary to keep power consumption of each entity under its respective, constant power budget. As an example, we can interpret each L_i from Section 3.3 as an individual power budget per entity. Consequently, even if an entity is the only active entity in the system, it will not be able to operate at a higher performance point which would consume more power than L_i. This can lead to significant performance loss and degrade overall power efficiency.

To quantify the overhead of this countermeasure, i.e., the performance loss caused by fixing (and thereby practically decreasing) per-entity power budget, we run the highly power hungry applicationon all four cores of the first platform (Section 3.5.1), and compare its performance to when only one core (out of four) is active. When all cores are active, each core inevitably can only consume less power at the peak (corresponding to L_i), to meet the overall system-wide power budget. On the other hand, when only one core is active (while other cores are idle), the active core can run at maximum performance by consuming by itself the entire effective budget for all four cores being active. We use the performance difference under both scenarios as a quantitative estimate for the overhead of the naive countermeasure. Overall, we observe a performance degradation of over 30.1%. Hence, the naive countermeasure incurs a high performance penalty which may not always be acceptable.

3.6.2 Operating Frequency Randomization

As reported in Section 3.5.1, the difference between the sinks's GFLOPS levels for the first case study, when communicating at the peak rate, is around 2.7%. The same gap between GFLOPS levels is around 0.5%, for the fastest covert communication on the second case study, as Section 3.5.2 reveals. Hence, on both systems we observe a slim gap that needs to be carefully sensed to be able to accurately decode the leaked information.

Based on this observation, one way that PM can limit the bandwidth for POWERT communication is by imposing random noise on the GFLOPS signal, simply by adding



(a) Intel (b) ARM

Figure 3.16: Impact of adding noise to GFLOPS signal on C.

random noise to the operating frequency of each core. In other words, when PM finds the optimal operating frequency for a core based on power demand, workload behavior, and other parameters, it can add random noise (e.g., in the range of [-2%,+2%]) to it before actually tuning the respective core's frequency. While this countermeasure would inevitably degrade power-efficiency and performance, it can significantly complicate the decoding process, potentially to a point where covert communication becomes impossible.

Fig. 3.16 depicts how adding random uniform noise to GFLOPS affects C, on both platforms in the attack scenario where we observe the highest channel capacity. To limit C to less than 0.1 bits per channel use, we have to add a random uniform noise with the magnitude of [-25%,+25%] of the GFLOPS signal on the Intel-based platform. On the other hand, this magnitude is around [-5%,+5%] on the ARM-based platform, since as reported in Section 3.5.2, gap between GFLOPS levels is smaller on ARM-based platform, compared to the Intel-based platform.

3.6.3 Slowing Down Communication

Section 3.3 provides an analytical model to estimate an upper-bound for POWERT communication, as a function of different system parameters. We can utilize this model to find ways to lower the upper-bound, to a desirable safe level. For instance, one easy way to slow-down POWERT communication is by increasing the decision making period of the power manager, t_{PM}. While this, as well, degrades overall power-efficiency, it can

effectively limit POWERT bandwidth. We can manipulate other parameters, as well, all at the cost of perturbing power management and consequently, degrading overall power-efficiency.

3.7 Related Work

Covert channels: Resource sharing, be it in hardware or software, is inevitable for power or area efficiency, however, almost exclusively brings up security concerns. Since one of the first mentions of covert channel attacks in 1973 [81], a variety of covert channels have been revealed [82, 70, 83, 84, 69, 68, 85, 86, 87]. The vast majority of this work covers cache-based covert channels [68, 82]. Not only higher level caches, but also the main memory and functional units shared by different threads under simultaneous multithreading (SMT) can be subject to information leakage through covert channels [81]. Recent work has also shown how thermal sensors (as a key component of on-chip thermal management) can enable similar covert communication [70]. Thermal effects can also lead to clock skew changes, which attackers can exploit for covert communication [83]. Covert channels which need special privilege, for instance to access hardware monitors like thermal sensors, can be blocked simply by restricting access to those resources. This does not apply to POWERT attacks, since no special privilege is needed to perform these attacks. Other hardware resources such as the memory bus [69], random number generator [84], magnetic field sensors [86], USB charging cable [87], and general purpose graphics processing units [85] are vulnerable, as well. The condition that the sender and receiver need to reside at the same place is necessary for covert communication in earlier studies, while recent studies demonstrate that this requirement can be relaxed if the timing of sender activities can be measured remotely [81]. Similar to many of these covert channels including cache-based covert channels, blocking POWERT attacks inevitably degrades system's performance and energy-efficiency (as explained in Section 3.6). It becomes even more challenging as power budget management is getting more crucial in preserving energy-efficiency of even more power budget limited platforms of the future.

On the modeling side, Hunger et al. proposed a simple mathematical abstraction to capture common characteristics of all microarchitectural channels [77]. While the

model is applicable to many contention-based microarchitectural channels, it does not directly apply to POWERT channels. This is because the model assumes that probing the channel perturbs the data. However, in POWERT channels, multiple receivers can listen to the covert channel, without affecting the transmitted data itself, as shown in Section 3.5.

Power management vulnerabilities: Power management vulnerabilities can result in a variety of security issues. For example, JayashankaraShridevi et al. analyzed two types of attacks enabled by hardware Trojans embedded in the power management unit (PMU) of a mobile system on chip [88]. The first attack leads to higher operating voltages than necessary. The second one delays the activation of power-gated blocks. In both cases, power efficiency degrades. Tang et al. demonstrate another type of vulnerability due to DVFS [89], where an attacker can enforce lower (higher) than safe voltages (frequencies) to induce timing errors. Physical access is not necessary, as software controls voltage regulators and phase-locked loops (PLLs). The authors show how to infer 128-bit AES keys via overclocking the processor. Zhang et al. recently proposed a mitigation technique for such power-management based fault injection attacks, by dynamically "blacklisting" unsafe operating points [90]. The recently revealed DVFS Channel [91] exploits the fact that a core's frequency can be dynamically controlled using DVFS (and not that cores share the same power budget). This type of attack is easier to block by limiting the access to the files containing current frequency information, while POWERT attacks do not need any privilege, making them harder to block. Finally, due to low rate of updates to frequency information files, DVFS Channel demonstrates much lower bit rates, compared to POWERT. PMU Trojan [92] is very similar to DVFS Channel, but attacks happen in hardware. Contrary to POWERT communication, PMU Trojan, as well, does not exploit the fact that cores share the same power budget. Similar to DVFS Channel, this attack also can be blocked by limiting access to operating frequency information of the cores, on the receiving side. Besides, POWERT attacks do not rely on hardware Trojan, and can be performed on regular PM hardware. The available instantaneous power budget itself represents a shared resource, therefore, power management practically entails finding the optimal allocation of the power budget among active tasks of execution. To the best of our knowledge, our study is the first to cover covert channel communication enabled by power headroom modulation, without the

need for any sort of privileged access to shared hardware or software resources.

Chapter 4

GeNVoM: Read Mapping Near Non-Volatile Memory

4.1 Introduction

DNA sequencing is the physical or biochemical process of extracting the order of the four bases (Adenine, Guanine, Cytosine, Thymine) in a DNA strand. As semiconductor technology revolutionized computing, DNA sequencing technology, termed *High-throughput Sequencing* or *Next Generation Sequencing* (NGS), revolutionized genomic research. As a result, modern NGS platforms can sequence hundreds of millions of short DNA fragments in parallel. The sequenced fragments (which represent the NGS output) are referred to as short *read*s and typically contain 100-200 bases [93]. The focus of this study is *read mapping*, a common critical first step spanning a rich and diverse set of emerging bioinformatics applications: mapping each NGS *read* to (the most similar portion of) a reference genome of the same species (which itself is a full-fledged assembly of already processed *read*s).

As a representative example, modern NGS machines from Illumina [93], a prominent NGS platform producer, can sequence more than 600Giga-bases (Gba) per one run, 200× the length of a human genome of approximately 3Gba, which translates into hundreds of millions of output *reads*. Fig. 4.1 depicts the scaling trend in terms of the total number of human genomes sequenced. The values until 2015 reflect historical publication records, with milestones explicitly marked. The values beyond 2015 reflect

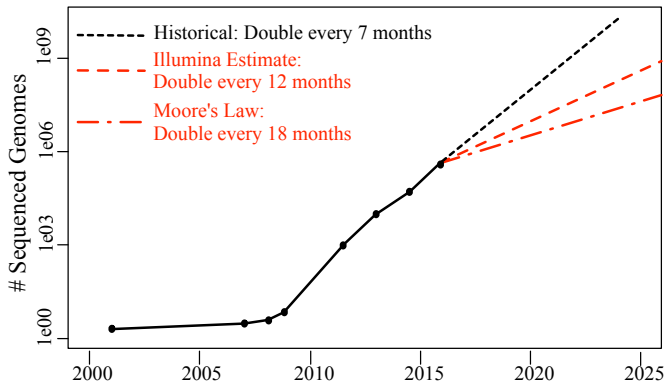


Figure 4.1: Scaling trend for DNA sequencing [94].

three different projections: the first, following the historical growth until 2015; the second, a more conservative prediction from Illumina; the third, Moore's Law. Historically, the total quantity has been doubling approx. every 7 months. Even the more conservative projections from Fig. 4.1 (i.e., 2× increase every 12 or 18 months) result in a very rapid growth, which challenges the throughput performance of *read mapping*.

The wildly increasing scale of the problem per Fig. 4.1 renders well-studied pair-wise similarity detection algorithms inefficient [95]. Worse, *read*s are subject to noise due to imperfections in NGS platforms and genomic variations, which adds to the complexity. Both algorithmic solutions and hardware acceleration via GPUs [96] or FPGAs [97] therefore have to trade mapping accuracy for throughput performance. In other words, *read mapping* by definition is after *similarity matching*. As optimizations are usually confined to compute-intensive stages of mapping, considering scaling projections from Fig. 4.1, most of these solutions are fundamentally limited by data transfer overheads. In this study, we instead take a data-centric position to guide the design (and explore the design space) of scalable, energy-efficient and high-throughput *read mapping*. Specifically, instead of optimizing an algorithm developed for general-purpose computers or GPUs, we rethink the algorithm from the ground up along with the accelerator design.

Read mapping represents a search-heavy memory intensive operation and barely requires complex floating point arithmetic, therefore, can greatly benefit from near-memory processing. Intuition suggests using fast parallel associative search, enabled by

Ternary Content Addressable Memory (TCAM) by construction, in matching short *read* patterns with portions of the large reference genome (stored in TCAM). As we will explain in Section 4.2, however, only *non-volatile* TCAM can accommodate the large memory footprint in an area- and energy-efficient manner [98]. Even then, brute-force non-volatile TCAM search over as large of a search space as *read mapping* demands induces excessive energy consumption, rendering (non-volatile) TCAM-based acceleration infeasible. At the same time, by construction, (non-volatile) TCAM cannot handle similarity matching under NGS or genomic variation induced noise.

This study provides an effective solution, GeNVoM, to tap the potential of *non-volatile* TCAM for scalable, energy-efficient high-throughput *read mapping*. GeNVoM

- introduces a novel similarity matching mechanism for resistive non-volatile TCAM (which can only handle exact matches by construction), to trade mapping accuracy for throughput and energy efficiency in a much more scalable manner than existing solutions;
- features a novel genomic data representation for efficient similarity matching without compromising storage;
- tailors common search space pruning approaches to its novel similarity matching mechanism in order to identify and discard unnecessary non-volatile TCAM accesses (and thereby, to prevent excessive energy consumption);
- accounts for the most prevalent manifestations of noise induced by NGS imperfections and genomic variations during similarity matching, including (base) gaps and insertions/deletions in *reads*;
- employs multi-phase hierarchical similarity matching to enhance mapping accuracy and scalability, where each phase performs progressively more sophisticated mapping, considering only the subset of *read*s the previous phase fails to map.

In the following, we introduce a proof-of-concept GeNVoM implementation. Specifically, Section 4.2 discusses basics; Section 4.3 covers implementation details; Sections 4.4 and 4.5 detail the evaluation; and Section 4.6 provides a compare and contrast to related work.

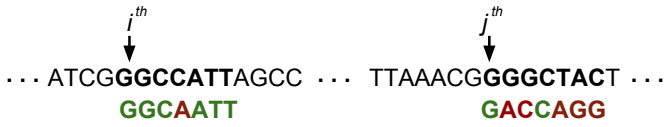


Figure 4.2: Read mapping example.

4.2 GeNVoM: Macroscopic View

Scope: Short (i.e., 100-200 base long) *read*s from modern Illumina NGS platforms [93] constitute more than 90% of all *read*s in the world currently. Accordingly, GeNVoM is designed for short *read* mapping.

Terminology: Without loss of generality, we will refer to each *read* simply as a *query*; and the reference genome, as the *reference*. Each *query* and the *reference* represent strings of characters from the alphabet {A, G, C, T} which stand for the bases {Adenine, Guanine, Cytosine, Thymine}. The inputs to GeNVoM are a dataset of *query*s and the *reference*, where the *reference* is many orders of magnitude longer than each *query*. For example, if the *reference* is the human genome, *reference* length is approximately $3{\times}10^9$ bases. On the other hand, technological capabilities of modern NGS platforms limit the maximum *query* length.

4.2.1 Problem Definition: Read Mapping

Basics: *Read mapping* entails finding the most *similar* portions of a given *reference* to each *query* from a dataset corresponding to the same species, as output by an NGS machine. Fig. 4.2 demonstrates an example, with different portions from the same *reference* on top; two sample *query*s to be mapped, at the bottom. The first (second) *query* results in one (five) base-mismatch(es) when aligned to the i^{th} (j^{th}) base of the *reference*. The *query* length is not representative, but simplifies demonstration.

GeNVoM's input *query*s are subject to noise due to imperfections in NGS platforms and potential genomic variations. Therefore, *read mapping* by definition is after *similarity* rather than an *exact match*. Hence, for each input *query*, GeNVoM tries to locate the *most similar* sub-sequence of the *reference* to the *query*, and returns the range of its indices.

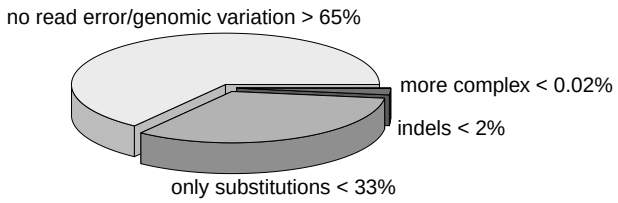


Figure 4.3: Manifestation of genomic variations & read errors.

Mapping Reverse Complement of *read*s: It is not uncommon for NGS platforms to sequence DNA strands in reverse direction. This happens when sequencing starts from the last base of a DNA strand. In this case, the platform outputs the reverse complement of a *read* by interchanging A with T, and C with G. For example, the reverse complement of the sequence ACCGCCTA is TAGGCGGT. NGS platforms typically sequence almost half of the DNA strands in reverse order, hence, GeNVoM is designed to handle these reads.

Sources of Noise in Similarity Matching: In general, the sequenced genome (where the *reads* are coming from) is expected to be slightly different from the *reference* genome, even though they represent the very same species. *Genomic variations* induce such differences, which can lead to base-mismatches between the *query*s and the *reference*, since the *query*s come from the sequenced genome as opposed to the *reference*. NGS platform imperfections, as well, can result in false base-mismatches between the *query*s and the *reference*, due to the so-called *read errors* during sequencing. We will next discuss the most prevalent manifestations of genomic variations and read errors.

Noise Manifestation: Most common genomic variations and read errors manifest themselves in three ways: Random *insertion* of a base, random *deletion* of a base, and random *substitution* of a base with another. Insertions and deletions are often referred to as *indels*. The expected rates of indels and substitutions depend on the type of genomes (hence species) and the NGS technology.

As a representative example, Fig. 4.3 demonstrates the typical share of *read*s having no read errors/genomic variations (>65%), at least one substitution (and no indels) (<33%),

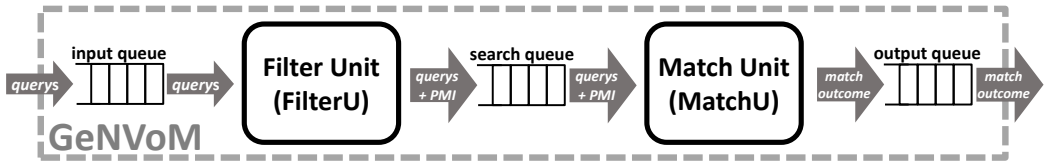


Figure 4.4: Structural organization.

at least one indel (<2%), and more complex manifestations (<0.02%) [99, 100, 101, 102]. Mapping under rare complex manifestations (such as long-indels, gaps, base duplications or inversions) is a daunting task, and to date there is no widely-accepted algorithm to cover all [101, 103]. As Fig. 4.3 shows, substitutions are dominant. Although indel rate is on the lower side, detecting indels is critical for many downstream applications. Hence, GeNVoM is designed to cover both substitutions and indels, but optimized for the common case (no read error/genomic variation and only substitutions), which covers more than 98% of the *read*s per Fig. 4.3. As we will demonstrate in Section 4.3.4, straight-forward expansion of GeNVoM to more complex manifestations such as gaps is also possible.

4.2.2 Why Naive (Non-Volatile) TCAM-based Acceleration Does Not Work

Read mapping essentially is a search-heavy memory intensive pattern matching problem. This suggests TCAM-based acceleration, which by construction can support fast parallel in-memory search. TCAM is a special variant of associative memory (which permits data retrieval by indexing by content rather than by address) that can store and search the "don't care" state X in addition to a logic 0 or 1. Considering the scale of the problem, however, only *non-volatile* TCAM can accommodate the large memory footprint in an area- and energy-efficient manner [98].

We will next look into the energy consumption of *read mapping*, comparing a non-volatile TCAM-based implementation with a highly optimized GPU-based solution deploying one of the fastest known algorithms to date [104]. The non-volatile TCAM mimics the least energy-hungry implementation from Guo et al. [98], corresponding to an array size of 1K×1Kbits = 1Mbits. For this design point, searching for a pattern of

length 1Kbits (which represents the maximum-possible length, i.e., the row length) in the entire array takes approximately 2.5ns and consumes 245nJ.

If we simply encode each base from the alphabet {A, G, C, T} using 2 bits, and if a human genome of approximately 3Giga-bases (= 6Gbits) represents the *reference*, the *reference* can fit into 6K TCAM arrays (of size 1K×1Kbits = 1Mbits). For each *query* of a typical length of 100 bases [93], i.e., 200 bits, the following naive procedure can cover the entire search space: By construction, each 1K×1Kbit TCAM array can search for at most one 1Kbit pattern at a time, which resides in a query register. We can align the most significant bit of the (200bit-long) *query* with the most significant bit position of TCAM's 1Kbit query register, and pad the remaining (1K-200) bits by Xs, for the very first search in the array. We can then repeat the search by shifting the contents of TCAM's query register (i.e., the padded *query*) to the right by one bit at a time, leaving the unused more significant bit positions with Xs, until the least significant bit of the *query* reaches the least significant bit position in the query register. The total number of these bit-wise shifts (and hence, searches) would be in the order of the row length $\approx$ 1K. Putting it all together, mapping a given *query* to the *reference* in this case would take around 1K searches in each of the 6K arrays, with 245nJ consumed per search. The overall energy consumption therefore would become 6K $\times$ 1K $\times$ 245nJ $\approx$ 1500mJ.

The GPU solution from Luo et al. [104] on the other hand, can process 133.3K *query*s per second. Hence, it takes 1/133.3K seconds to map a single *query*. Even under the unrealistic assumption (favoring TCAM) that the entire peak average power (TDP) goes to mapping a single *query* to the *reference*, the energy consumption would become at most 235W $\times$ (1/133.3K)s $\approx$ 1.8mJ.

The GPU and TCAM designs feature similar technology nodes, however, even by favoring TCAM, the TCAM-based naive implementation consumes approx. 3 orders of magnitude more energy than the GPU-based. This difference stems from the gap in the size of the search spaces. While the TCAM-based design considers the entire search space to cover all possible alignments, the GPU-based design first prunes the search space to eliminate infeasible alignments, which in turn leads to orders of magnitude less number of (search) operations. GeNVoM, while deploying non-volatile TCAM arrays, adopts a similar pruning strategy to enable more energy-efficient search.

Even if excessive energy consumption was not the case, (non-volatile) TCAM has

another fundamental limitation which hinders applicability to *read mapping*. As we will detail in Section 4.3.3, even in the presence of "don't cares", TCAM cannot handle similarity matching considering various manifestations of noise due to NGS errors and genomic variations (Section 4.2.1).

To summarize, both, *the excessive energy consumption and lack of support for similarity matching render a direct adaption of non-volatile TCAM-based search infeasible.* The energy overhead of conventional volatile TCAM would be even higher [98], while the restriction on similarity matching directly applies irrespective of volatility.

GeNVoM unlocks the throughput potential of non-volatile TCAM in a scalable and energy-efficient manner through

- a novel **non-volatile** resistive TCAM design capable of **similarity matching** (Sect. 4.3.3);
- a novel **genomic data representation for similarity matching** without compromising storage complexity (Sect. 4.3.2);
- a common filtering mechanism [105] for **search space pruning adapted to non-volatile similarity matching** to prevent excessive energy consumption (Sect. 4.3.1);
- **hierarchical similarity matching** to maximize mapping accuracy without compromising scalability (Section 4.3.4).

Designed for similarity search (in the presence of NGS or genomic variation triggered noise), GeNVoM's non-volatile TCAM arrays can directly handle substitutions, by construction (Sect. 4.3.3). To cover indels and more complex corruptions, on the other hand, GeNVoM adapts *anchoring* within its multi-phase mapping hierarchy (Sect. 4.3.4). The key insight is that complex corruptions that can lead to failed mappings are much less likely to occur in *all* portions of a *read* simultaneously. This makes anchoring very effective – a divide and conquer technique which entails chunking the *read* (at an anchored base position) to shorter substrings and attempting mapping on each chunk simultaneously. Thereby, problematic chunk(s) (and hence the entire *read*) simply follow the alignment dictated by the less-corrupted chunks, which by construction renders the most accurate mapping under noise. Numerous prevalent *read mapping* algorithms [106] therefore use anchoring-based techniques for complex corruptions.

4.2.3 Hardware Organization

Fig. 4.4 provides the structural organization. GeNVoM pipeline comprises two major units: Filter Unit (FilterU) and Match Unit (MatchU). Each *query* from the dataset to be mapped streams into the (first stage of the) GeNVoM pipeline (i.e., FilterU) over the *input queue*. Once the mapping completes, the outcome streams out of the (last stage of the) GeNVoM pipeline (i.e., MatchU) over the *output queue*. Non-volatile TCAM arrays (featuring GeNVoM's novel similarity matching mechanism) within MatchU keep the entire *reference*.

Input and output queues handle the communication to the outside world, by retrieving *query*s on the input end, and upon completion of the mapping, by providing the indices of the most *similar* sub-sequences of the *reference* to each *query*, on the output end.

FilterU filters (indices of) sub-sequences of the *reference* which are more likely to result in a match to the incoming *query*, by examining sub-sequences of the incoming *query* itself. We call these indices *potentially matching indices*, PMI. FilterU feeds the MatchU with a stream of <PMI, *query*> tuples over the *search queue*. MatchU in turn conducts the search by only considering PMI of the *reference*. In this manner, GeNVoM prunes the search space.

The input queue feeds the GeNVoM pipeline with the *query*s to be mapped to the *reference*. The *query* dataset resides in memory. GeNVoM initiates the streaming of the *query*s into the memory-mapped input queue over a Direct Memory Access (DMA) request. The input queue in turn sends the *query*s to FilterU for search space pruning before the search takes place. Finally, for each *query*, once the mapping completes, the output queue collects from MatchU the indices of the sub-sequence of the *reference* featuring the most similar match to the *query*. The output queue is memory-mapped, too. So, GeNVoM writes back these indices to a dedicated memory location, over DMA.

In the following, we will detail the steps for *query* processing in each unit in case of a match. If no sub-sequence of the *reference* matches the input *query*, no mapping takes place, and GeNVoM updates a dedicated flag at the memory address to hold the result. GeNVoM can detect such failed mapping attempts during processing at FilterU or at MatchU.

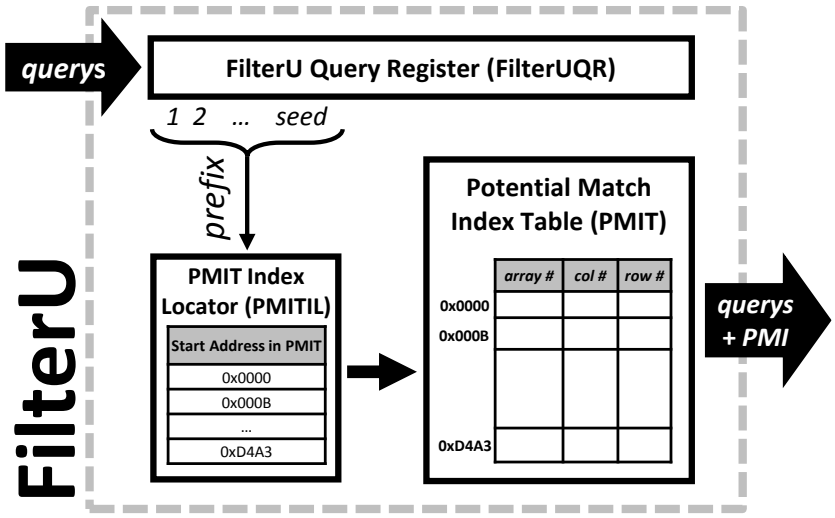


Figure 4.5: Filter Unit (FilterU).

Filter Unit (FilterU)

Fig. 4.5 provides the structural organization of FilterU, which serves the compaction of the search space for each *query* to be mapped to the *reference*, as follows: We will refer to each sub-sequence of length *seed* as a *prefix*, where *seed* represents a design parameter and assumes a much lower value than the *query* length. As each *prefix* is a string of characters from the 4-character alphabet {A, G, C, T}, a *prefix* of length *seed* can take 4^{seed} different forms. Considering the size of the problem, the *reference* is likely to occupy multiple TCAM arrays. FilterU relies on a pre-processing step which entails identifying each *prefix* of length *seed* in the *reference*, and recording the TCAM array, column and row number of the corresponding occurrence. *Potential Match Index Table* PMIT keeps this information.

However, as the same *prefix* may occur multiple times along the *reference* string, PMIT may contain multiple entries for the very same *prefix*. Therefore, FilterU has another table called *PMIT Index Locator* (PMITIL) for bookkeeping. PMITIL serves as a dictionary of 4^{seed} entries, considering all possible 4^{seed} values of the (*seed*-long) *prefix*. Each PMITIL entry refers to a specific *prefix* value, and keeps the start address in PMIT where the TCAM indices for the corresponding occurrence of the *prefix* (along

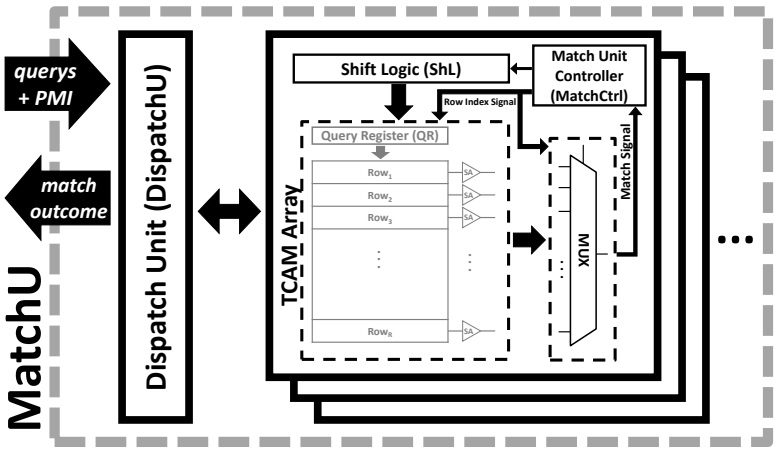
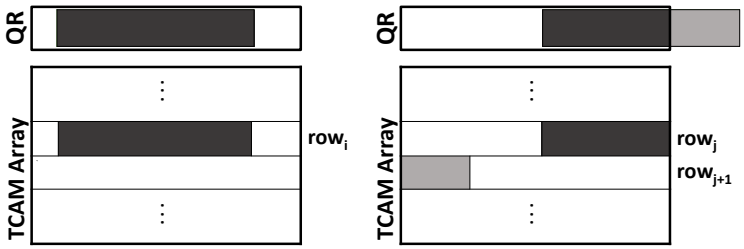


Figure 4.6: Match Unit (MatchU)

the *reference*) reside. As PMIT is organized to keep multiple occurrences (along the *reference*) of the same *prefix* consecutively, it suffices to keep per PMITIL entry just the start address (in the PMIT) for the first occurrence. The end address in this case simply is the start address stored in the next PMITIL entry.

PMIT and PMITIL generation constitutes a pre-processing step which GeNVoM needs to perform only once, offline, for each *reference*, before *read mapping* starts. As *read mapping* entails mapping a large number short *read*s to a given *reference* of the same species, this overhead does not apply to runtime, and is easy to amortize.

Upon receipt of a new *query* from the head of the input queue, FilterU uses the first *seed* bases of the *query* as the *prefix* to consult PMITIL, and subsequently, PMIT. FilterU keeps the *query* being processed in the FilterU *Query Register* (FilterUQR) as filtering is in progress. If there is a match in the PMI tables, FilterU first broadcasts the *query* being processed to all TCAM arrays. Then, it sends the corresponding TCAM array, column and row number (i.e., the Potential Match Indices, PMIs) to MatchU, over the *search queue*. We will refer to these TCAM coordinates as $array_{\#}$, $col_{\#}$, and $row_{\#}$, respectively.



(a) *full match* (b) *fragmented match*

Figure 4.7: Full (a) and fragmented (b) TCAM match.

Match Unit (MatchU)

Fig. 4.6 provides the structural organization of MatchU, which orchestrates search. MatchU features the *Dispatch Unit* (DispatchU) and non-volatile TCAM arrays capable of similarity search under NGS or genomic variation induced noise. DispatchU acts as a scheduler for TCAM search. For each input *query* to be mapped to the *reference*, DispatchU collects the TCAM $array_{\#}$, $col_{\#}$ and $row_{\#}$, as extracted from the PMIT in FilterU, to initiate the targeted search.

The input *query* stays in the *Query Register* (QR) of the TCAM array $array_{\#}$ during TCAM access. *Shift Logic* (ShL) in TCAM array $array_{\#}$ in turn first aligns the *prefix* of length *seed* of the *query* with the *seed*-long (matching) sub-sequence of the *reference* residing (in array $array_{\#}$) in row $row_{\#}$, starting from column $col_{\#}$. To this end, ShL shifts *query* bits in QR and inserts Xs accordingly. *Match Unit Controller* (MatchCtrl) orchestrates this operation. Once alignment completes, MatchCtrl activates the row $row_{\#}$ for search. Once the search completes, MatchCtrl provides DispatchU with the indices of the *reference* which demarcate the most similar sub-sequence to the entire *query*. DispatchU then forwards these indices to the output queue.

Fig. 4.7 depicts two different match scenarios: In Fig. 4.7a, the *query* (shown in dark shade within QR, white space corresponding to Xs for padding) matches a sub-sequence of the *reference* which is entirely stored in a single row of the array. We call this scenario a *full match*. On the other hand, in Fig. 4.7b, the *query* matches a sub-sequence of the *reference* which is stored in two consecutive rows of the array. We call this scenario a

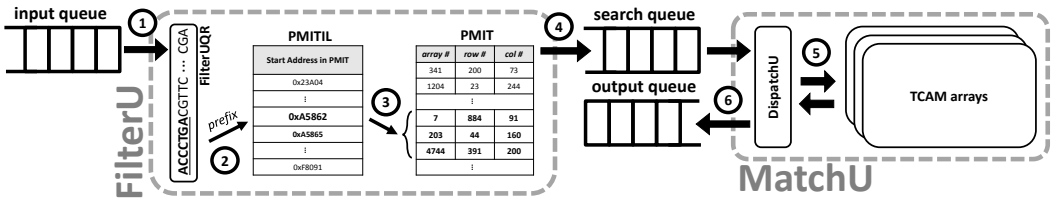


Figure 4.8: Life-cycle of a *query* in GeNVoM.

fragmented match. Fragmentation can happen at both ends of the *query*. For example, in Fig. 4.7b, the first portion of the *query* (shown in darker shade) matches the end of row j, while the rest (shown in lighter shade) matches the beginning of the next row, row j+1. MatchCtrl needs to address such fragmentation as GeNVoM lays out the character string representing the *reference* in two dimensions in each array *consecutively*.

Conventional TCAM can only detect full match. Handling fragmented match requires extra logic. By default, the TCAM array would select the longest sub-sequence l of the *reference* matching the input *query* if a full match is not the case, where l occupies an entire row. The darker-shade region in Fig. 4.7b corresponds to such l. As l may be aligned to either the beginning (Fig. 4.7b) or the end of the *query*, MatchCtrl has to additionally check the next or the previous row, respectively, for a match to the unmatched portion of the *query*. We call the first case a *fragmented tail match*; the second, a *fragmented head match*. In case of a fragmented match, search in the TCAM array takes two steps. As a fragmented match may also happen at TCAM array boundaries, each array's last row duplicates the first row of the next array in sequence.

Putting It All Together

Fig. 4.8 summarizes the 6 steps in mapping a *query* to the *reference*: First, FilterU retrieves a new *query* from the head of the input queue at step ①. In this case *seed*=7 (bases) with the corresponding 7-base *prefix* of the *query* underlined. Then, at step ②, FilterU locates the entry for the 7-base *prefix* of ACCCTGA in PMITIL, and extracts the corresponding PMIT address(es). Next, at step ③, FilterU retrieves TCAM array, column, and row numbers (i.e., $array_{\#}$, $col_{\#}$, and $row_{\#}$; for targeted search in MatchU) for the sub-sequences of the *reference* which match the *prefix* ACCCTGA, from the PMIT

addresses collected at step ②. Finally, FilterU sends the *query* along with $array_{\#}$, $col_{\#}$, and $row_{\#}$ to MatchU over the search queue at step ④. At step ⑤, DispatchU initiates search in TCAM array $array_{\#}$, at $row_{\#}$ and $col_{\#}$, and collects the match outcome. At step ⑥, MatchU sends the match outcome to the output queue.

4.3 GeNVoM: Microscopic View

4.3.1 Search Space Pruning

In order to prune the search space, GeNVoM first locates sub-sequences of the *reference* matching the *seed*-long *prefix* of the *query* in FilterU (Section 4.2.3). *seed* represents a key GeNVoM design parameter which dictates not only the storage complexity, but also the degree of search space pruning, which in turn determines GeNVoM's throughput performance and energy efficiency.

PMITIL grows with 4^{seed}, therefore, the larger the *seed*, the higher becomes the storage complexity. However, a larger *seed* is more likely to result in a lower number of *prefix* matches in the PMI tables, and hence, a lower number of targeted searches in the MatchU. In either case, the *seed* value remains much less than the expected length of the *query*.

PMIT can have at most as many entries as the total number of *seed* long sub-sequences contained within the *reference*. This practically translates into the length of the *reference*, as a *prefix* can start from each base position of the *reference* onward. As PMIT is organized to keep multiple occurrences of the same *prefix* consecutively, each PMITIL entry just keeps the start address in the PMIT for the first occurrence. PMIT, on the other hand, has to keep the <TCAM array number, column number, row number> tuple for each *prefix* match. If the *reference* is the human genome, PMIT would have approximately 3Giga entries. As we will detail in Section 4.4.2, 32 bits suffice to store each <TCAM array number, column number, row number> tuple per PMIT entry; and 32 bits, each <PMIT start address> per PMITIL entry.

PMIT keeps the entries corresponding to the very same *prefix* always at consecutive addresses, and re-orders such entries further to have all entries pointing to the same TCAM array reside at consecutive addresses. GeNVoM processes multiple PMIT matches per *prefix* in this consecutive order. Under such re-ordering, communicating a list

of PMIs and performing search in the array happen in a pipelined fashion. This masks communication latency and consequently, can improve throughput performance significantly.

4.3.2 Data Representation

Each input *query* and the *reference* itself represent character strings over the alphabet {A, G, C, T}. Conventional bioinformatics formats such as FASTA [107] encode each letter from such alphabets of bases by single-letter ASCII codes. However, TCAM arrays conduct the search at bit granularity. Therefore, GeNVoM needs to translate *base character* mismatches to *bit* mismatches. To this end, GeNVoM adopts an encoding which renders the very same number of mismatched bits for a mismatch between any two base characters. This would not be the case, if we encoded each base character in {A, G, C, T} by simply using 2 bits: a base-mismatch would sometimes cause a 2-bit mismatch (e.g., when comparing '01' to '10'); other times, a single-bit mismatch (e.g., when comparing '00' to '10'). GeNVoM's encoding instead uses 3 bits per base character, where any two 3-bit code-words differ by exactly 2 bits, such as {111, 100, 010, 001}. Thereby GeNVoM guarantees that exactly 2 bits would mismatch for any base character mismatch.

4.3.3 Similarity Search

Fig. 4.9 depicts a representative resistive TCAM cell. The two resistors attached to the access transistors Acc_1 and Acc_2, respectively, carry the data bit value D and its complement $\overline{D}$. The high (low) resistance value R_{high} (R_{low}) encodes logic 1 (0). The third resistor attached to Acc_3 is hardwired to logic 1, hence its resistance remains constant at R_{high}.

To search for logic 0, *Search Line* (SL) is set to 0, and its complement $\overline{SL}$ to 1, such that Acc_2 turns on; Acc_1 off. Thereby only the resistor carrying $\overline{D}$, R, gets connected to the *Match Line* (ML). If the cell content was 0, i.e., $D = 0$ and $\overline{D} = 1$, there would be a match, and $R = R_{high}$ applies. Otherwise, if the cell content was 1, i.e., $D = 1$ and $\overline{D} = 0$, there would be a mismatch, and $R = R_{low}$ applies. A symmetric discussion holds for searching for logic 1. On a per TCAM cell basis, R_{high} connected to ML indicates a

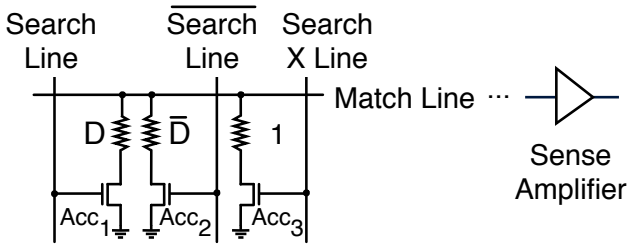


Figure 4.9: Resistive TCAM cell [98].

match, R_{low}, a mismatch. To search for X, both SL and $\overline{SL}$ are set to 0, and *Search X Line* to 1, such that only the hard-wired R_{high} attached to Acc_3 is connected to ML. This is how search for X always renders a match, independent of the value of D. Each cell within each row contributes to the effective resistance connected to ML, R_{eff}, by R_{high} (R_{low}) on a match (mismatch). The *Sense Amplifier* SA (connected to the ML) in each row signals a (mis)match for the entire row depending on the value of R_{eff}. SA would only signal a row-wide match, if all cells match, i.e., if each cell contributes to R_{eff} by R_{high}. Let us call the R_{eff} in this case $R_{row-wide-match}$. SA would signal a row-wide mismatch if at least one cell mismatches, i.e., contributes to R_{eff} by R_{low}. The value of R_{eff} in this case evolves with the number of cell mismatches, and assumes the closest value to $R_{row-wide-match}$ under a single-cell (bit) mismatch. The contents of the TCAM query register directly correspond to the values on the search lines.

In a TCAM array based on the cell from Fig. 4.9, unless all bits within a row match, SA always signals a mismatch for the entire row. However, as explained in Sect. 4.2.1, a matching *query* may indeed have a few bases different from the corresponding subsequence of the *reference*, due to NGS or genomic variation induced noise. To resolve this discrepancy, GeNVoM deploys tunable SAs which associate a wider R_{eff} range with a row-wide match. We can tune these SAs to signal a row-wide match when less than a given number t of bits mismatch, which translates into less than t R_{low}s connected to ML. We will refer to t as the *tolerance*, which represents an adjustable design parameter.

The gap between R_{eff} levels corresponding to different number of mismatching bits decreases as the number of mismatching bits grows, complicating SA design. At the same time, due to PVT (Process, Voltage, Temperature) variations, individual TCAM cell

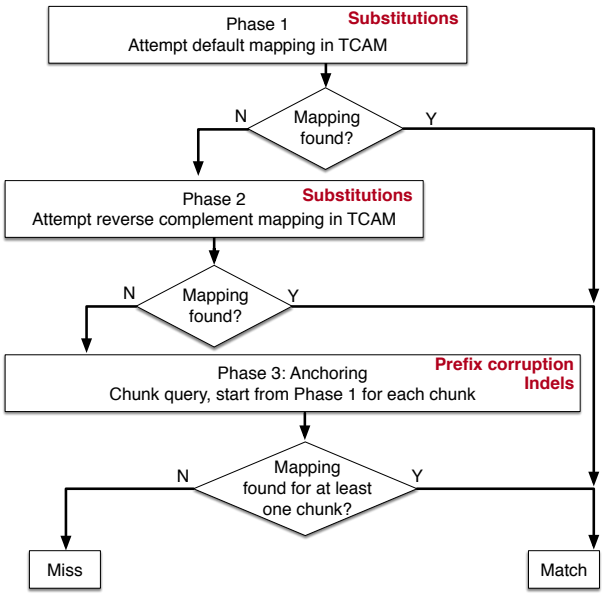


Figure 4.10: GeNVoM's hierarchical multi-phase search flow.

resistance levels may notably deviate from nominal R_{high} or R_{low}, leading to divergence of such R_{eff} levels from their expected values. In Sect. 4.4.4, we will detail how GeNVoM tunes the SAs in a variation-aware way.

Using GeNVoM's 3-bit (per base) encoding (Sect. 4.3.2), any single-base mismatch results in exactly 2-bit mismatches. Under the more intuitive 2-bit encoding, on the other hand, a base-mismatch sometimes causes 2-bit mismatches; other times, a single bit mismatch, further complicating the configuration of *tolerance*.

4.3.4 Hierarchical Multi-Phase Search

Our focus so far was on the very basics of GeNVoM's mapping mechanism. We will next look into the mapping accuracy, specifically, under what circumstances GeNVoM may not be able to map a given *query* to the respective *reference*, which in fact was *similar enough*. In the following, we will refer to such cases as *missed* mappings. NGS imperfections (i.e., read errors) and genomic variations complicate mapping, and thereby can lead to misses.

As explained in Sect. 4.2.1, GeNVoM is designed to operate under all prevalent

manifestations of read errors and genomic variations. To this end, GeNVoM employs multi-phase hierarchical mapping. Fig. 4.10 provides an overview. Each phase acts as a filtering layer for the subsequent phase, which in turn performs more complex mapping. *More complex mapping* entails re-attempting by considering *more complex manifestations* of read errors/genomic variations than the predecessor phase did. In this manner, each phase re-attempts mapping only for the subset of *query*s that the previous phase missed.

If the mapping in a phase fails, MatchU raises the *Missed-Map* signal. GeNVoM in turn feeds *Missed-Map* back to FilterU, to trigger more complex mapping attempts in the subsequent phase(s).

Phase 1 attempts a basic mapping in the TCAM arrays, assuming that a reverse complement (Sect. 4.2.1) is not the case. In Phase 1, FilterU and MatchU closely follow the steps detailed in Sect. 4.2.3. GeNVoM TCAM search, by construction, can effectively identify similarity under base substitutions, which represent the common case per Fig. 4.3. Phase 1 can miss a mapping under three cases:

(i) Reverse complement: the *query* is a *read* sequenced in reverse order. The probability for this case, $P(i)$, is approximately 50%.

(ii) Prefix corruption: the *query*'s *seed*-long *prefix* (used for search space pruning in FilterU) has substitutions or indels. A corrupted *prefix* may lead to ill-addressed search requests, i.e., FilterU sending incorrect PMIs to MatchU. If the probability of having a corruption in a given base location is $P(loc)$, the probability for this case, $P(ii)$, becomes $1-(1-P(loc))^{seed}$. We can estimate $P(loc)$ by adding a typical read error rate of 0.1% [99] to an average genome variation rate of 0.1% [100, 101, 102]. Using this estimate, for a representative *seed* value of 15 (Sect. 4.4), $P(ii)$ barely reaches 3.0%.

(iii) Indels: the *query* contains indels, anywhere. The most common indels are short indels induced by genome variations. Let $P(indel)$ be the probability of a short indel, and len, the length of the *query*. Then, $P(iii) = 1-(1-P(indel))^{len}$ applies. While there is no consensus on $P(indel)$, 0.01% represents a conservative estimate [100, 108], which renders $P(iii) \approx 1.5\%$ for a typical *query* length of 150 [93].

Phase 2 handles missed mappings due to reverse complements. After getting *Missed-Map* from MatchU (at the end of Phase 1), FilterU immediately sends PMIs corresponding

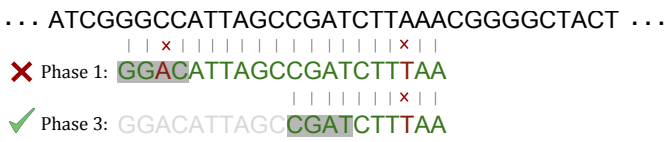


Figure 4.11: An example of anchoring.

to the reverse complement of the *query* to MatchU. To accelerate processing, MatchU employs an extra register inside the Shift Logic, which keeps the reverse complement of the *query* in addition to the original. MatchU copies the reverse complement in this register at the time it gets the original *query* (during Phase 1). Therefore, upon receipt of *Missed-Map*, FilterU does not need to broadcast the reverse complement separately, but only the PMIs for the reverse complement (which FilterU simply extracts by consulting the PMI tables with the *seed*-long *prefix* of the reverse complement.)

Phase 3 handles missed mappings due to *prefix* corruptions and indels, by adapting *anchoring* [106]. This phase processes all *query*s which Phase 2 was not able to map. Phase 3 first anchors the *query* in the middle to chunk the *query* uniformly into two. Then, each chunk separately goes through Phase 1, and if necessary, through Phase 2. Unless Phase 1 (or Phase 2, as need be) manages to map at least one of the two chunks to the *reference*, Phase 3 is considered to miss the mapping.

Fig. 4.11 depicts an example where Phase 3 maps a *read* which Phase 1 fails to map due to prefix corruption. The top row depicts the relevant portion of the *reference*. The second and third rows show the corresponding *read*, with pointers to the matching outcome at Phase 1 and 3, respectively. The alignment of the *read* w.r.t. the *reference* reflects the ideal alignment (which renders the most similar mapping). The shaded portion corresponds to the *prefix* (of length 4 in this case). The *read* and *prefix* lengths are not representative, but ease illustration. Phase 1 fails to identify this alignment due to the single base corruption (C→A) in the third base of the *prefix*. Phase 2 is of not much help either, as a reverse complement is not the case. Phase 3 comes to rescue, by chunking, i.e., *anchoring* the *read* in the middle, and attempting mapping for each half. The mapping of the first half still fails in this case, due to the very same corruption in the *prefix*. The *prefix* of the second half (i.e., CGAT), however, is not corrupted, and GeNVoM TCAM arrays can easily handle the single base mismatch in this half, which

renders the correct alignment as a result – simply following the alignment dictated by the second half for the entire *read*.

Similar to this example, the proof-of-concept GeNVoM design adapts 2-way chunking for anchoring, where multi-way (and not necessarily uniform) chunking can further help reduce the number of missed mappings. Anchoring essentially is a *divide and conquer* method. The key insight is that corruptions to lead to missed mappings are much less likely to occur in all of the shorter chunks simultaneously. In Sect. 4.5.2, we will demonstrate how anchoring can improve mapping accuracy significantly.

Phase 3 can miss a mapping under the very same two conditions as Phase 1 (namely *prefix* corruptions and indels anywhere; cases **(ii)** and **(iii)**), but only if these conditions apply to both of the chunks under *anchoring*. Under uniform two-way chunking, a typical *query* length of 150 bases renders a chunk length of 75, for which $P(ii)$ and $P(iii)$ become 3.0% and 1.0%, respectively. Hence, the probability to miss the mapping of a chunk would be approximately 4.0%. As Phase 3 can miss a mapping only by missing both chunks, the probability of missing a mapping in Phase 3 becomes approximately $(4.0\%)^2 = 0.16\%$.

By construction, anchoring can also help with more complex scenarios than indels or prefix corruptions, including long-indels, gaps, base duplications or inversions, as long as one half of the affected querys is still mappable by GeNVoM. While anchoring reduces the miss probability significantly, Phase 3 may still miss mappings due to rare complex variations. To map such problematic *query*s, GeNVoM can always be paired with sophisticated software algorithms.

4.4 Evaluation Setup

4.4.1 System-level Characterization

Without loss of generality, all components of the proof-of-concept GeNVoM design reside in a single card attached to the PCIe bus. The host loads the *reference*, PMIT, and PMITIL tables to the GeNVoM card before mapping starts.

To explore GeNVoM's design space, we experiment with different numbers (N) of FilterUs and MatchUs under a fixed problem size. Fig. 4.12 provides the overview. In this case, each MatchU stores $1/N$ of the *reference* and the FilterU paired with it only

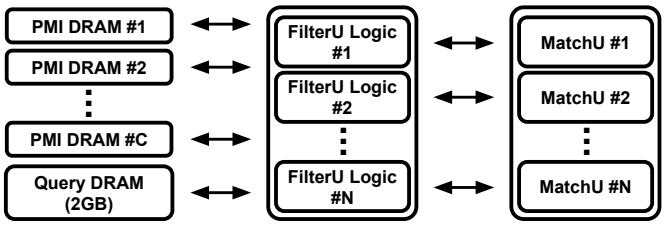


Figure 4.12: Organization of a single GeNVoMcard.

covers the PMIs for that portion of the *reference*.

This translates into banking PMIT to assign the FilterU only one bank covering the *reference* portion in the paired MatchU. As the problem size (which is dictated by the *reference* length) is fixed, such distributed GeNVoM configurations don't result in larger TCAM arrays or PMIT. However, each PMIT bank still needs a separate index locator, i.e., PMITL. As discussed in Sect. 4.2.3, PMITIL capacity depends on the *seed* only, not the size of PMIT. Therefore, N GeNVoM pipelines (with each processing only $1/N$ of the *reference*) still require N full-size PMITILs (of 4^{seed} entries).

In the rest of the evaluation, we characterize each GeNVoM configuration by four parameters: M, C, L, and N. A total on-card DRAM space of MGB is required to store all PMIT and PMITIL. This MGB of memory feeds the GeNVoM pipeline via C channels. N represents the number of FilterU+MatchU pairs, and L, the *seed* value.

GeNVoM relies on a GPU kernel (based on [109]) to re-format the match outcome of mapped *query*s, i.e., to find different SAM format fields such as CIGAR and MAPQ. We run the kernel on an NVIDIA K40 node in parallel, where it processes mapped *query*s in a pipe-lined manner periodically. This significantly reduces throughput overhead of running the GPU kernel, while we quantify its energy-efficiency implications by real measurements (similar to Section 4.4.6).

To quantify throughput and energy-efficiency, we use Ramulator [110]. The default Ramulator can only model accesses to the DRAM chips, so we implemented intra-card interactions during search, control logic and network related operations. We assume LPDDR4-4266 DRAM to store PMIT, PMITIL, and the *query* dataset. We use DRAMPower [111] to estimate the power consumption, which doesn't support LPDDR4. Therefore, we conservatively report DRAM power based on DRAMPower's LPDDR3

model.

4.4.2 PMI Table Generation

As explained in Section 4.3.1, PMIT keeps an entry for each possible *seed*-long *prefix* contained within the *reference*. In other words, PMIT keeps an entry for each possible base position in the *reference* to demarcate the start of a *seed*-long *prefix*. Therefore, PMIT capacity becomes practically independent of the *seed* for feasible *seed* values. Allocating an entry for each possible base position in the *reference*, a tight-enough upper bound for PMIT capacity for the human genome used as the *reference* for evaluation (Section 4.4.5) is approximately 11.4GB, independent of the *seed*.

On the other hand, PMITIL contains 4^{seed} entries. We evaluate GeNVoM considering different *seed* values. Table 4.1 captures PMITIL capacity for practical *seed* values ranging from 10 to 15. PMIT size and $N\times$PMITIL size together, M, determine the DRAM space requirement of the proof-of-concept GeNVoM implementation.

4.4.3 Circuit-level Characterization

The proof-of-concept GeNVoM implementation uses Phase Change Memory (PCM) as the resistive memory technology for TCAM arrays, which features a relatively high R_{high} to R_{low} ratio: 11.5 on average [112]. A higher R_{high} to R_{low} ratio eases sensing (i.e., distinguishing between matches and mismatches as explained in Section 4.3.3), therefore, enables arrays with longer rows. GeNVoM's TCAM arrays are similar to the most energy-efficient design from Guo et al. [98], which corresponds to a 1K×1Kbit configuration.

We synthesize logic circuits by Synopsys Design Compiler vH2013.12 using the FreePDK45 library [113]. To match the technology of our baselines for comparison (Section 4.4.6), we scale the outcome from 45nm to 28nm using ITRS projections [114]. GeNVoM's logic operates at 1GHz. A single search operation takes 1ns to complete, while consuming 0.1nJ of energy. We use ORION2.0 [115] to model the network. H-tree network connecting TCAM arrays operates at 1GHz, while each hop (1 router + link) consumes 3.83mW.

	Size (GB)					
seed	**10**	**11**	**12**	**13**	**14**	**15**
PMITIL	0.004	0.017	0.067	0.268	1.074	4.295

Table 4.1: PMITIL table capacity as a function of *seed*.

4.4.4 Similarity Matching Specification

GeNVoM adopts the Voltage Latch Sense Amplifier (VLSA) design from [116] to implement tunable sensing as explained in Section 4.3.3. We simulate VLSA in HSPICE v2015.06 using the FreePDK45 [113] library. VLSA's threshold voltage sets the boundary between the ranges of the effective resistance, R_{eff}, values the SA perceives as a (row-wide) match or a mismatch. We configure VLSA's threshold voltage to account for potential fluctuation in R_{high} and R_{low} values due to PVT variations.

Setting the sensing threshold

We conduct a Monte Carlo analysis using the (variation-afflicted) R_{high} and R_{low} distributions from IBM [112], extracted from measured data: $\mu(R_{high}) = 243.8K\Omega$, $\sigma(R_{high}) = 50.9K\Omega$, $\mu(R_{low}) = 21.2K\Omega$, and $\sigma(R_{low}) = 2.5K\Omega$. μ and σ represent the mean and the standard deviation. Considering a row size of 1Kbits, we find R_{eff} for 1M sample scenarios each corresponding to a different number of base mismatches. Using the resulting R_{eff} distribution, and capping the maximum number of base mismatches that are permitted to pass as a match (i.e., the *tolerance* as explained in Section 4.3.3), we set SA's sensing threshold in a variation-aware manner. This prevents false negatives, i.e., SA signaling a mismatch for a similar-enough *query*.

Sensing Accuracy

As explained in Section 4.3.3, under PVT variations, sense amplifiers may trigger a (row-wide) match in case of an actual (row-wide) mismatch. For each such case, the number of base mismatches remains higher than the preset *tolerance* value. In the following, we will refer to this difference in the number of base mismatches with respect to the *tolerance* as *overshoot*. These cases are not necessarily errors, and rather translate into a *query* of less similarity than expected being matched to a sub-sequence of the

reference. Therefore, as long as the overshoot (in terms of base mismatches) with respect to the anticipated *tolerance* remains bounded, each such case can easily pass as a *less similar match* (which in fact can be an actual match where the input *query* was significantly corrupted). Monte Carlo analysis from Section 4.4.4 shows that for different representative *tolerance* values used in Section 4.5, overshoot is usually less than 3, with probability of an overshoot of size 3 or larger barely reaching 0.05%. We quantify the impact of SA's inaccuracy on GeNVoM's overall accuracy in Section 4.5.2.

4.4.5 Input Dataset

We use a real human genome, `g1k_v37`, from the 1000 genomes project [117] as the *reference* genome; and 20 million 100-base long real *reads* from NA12878 [118] as a *query* dataset. For mapping accuracy analysis, we further generate 20 million more *query*s using 100-base long randomly picked sub-sequences from this *reference*, which we corrupt considering a read error probability of 0.1% (to mimic modern Illumina platforms), a single substitution [1] probability of 0.09%, and a short indel probability of 0.009%. For a fair comparison (not to favor GeNVoM) we choose the number of *query*s to have the *reference* + *query*s fit into the main memory of the GPU, such that the GPU does not suffer from extra energy-hungry data communication. We also limit the evaluation to a single GeNVoM card to keep the resource utilization comparable to the baselines.

4.4.6 Baseline for Comparison

As comparison baselines, we pick a highly optimized GPU implementation of the popular BWA algorithm, SOAP3-dp [104], and a very efficient hardware accelerator for short *read* alignment, GenAx [119]. A pure software-based implementation of GeNVoM is orders of magnitude slower than SOAP3-dp. We evaluate the throughput performance and power consumption of SOAP3-dp on an NVIDIA Tesla K40 GPU. We measure the power consumption of the GPU using NVIDIA-SMI (System Management Interface) command. We use the same *reference* and *query* dataset (Section 4.4.5) as GeNVoM as inputs. We compare GeNVoM against two different configurations of SOAP3-dp: The

[1] i.e., single-nucleotide polymorphism, SNP, the dominant type of substitutions induced by genomic variations

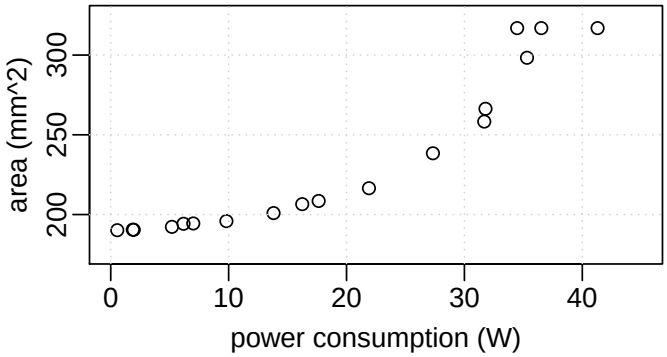


Figure 4.13: Area and power consumption of evaluated GeNVoM configurations.

first one, SOAP$_{SUB}$, only handles substitutions; the second one, SOAP$_{ALL}$, captures all prevalent manifestations of read errors and genomic variations.

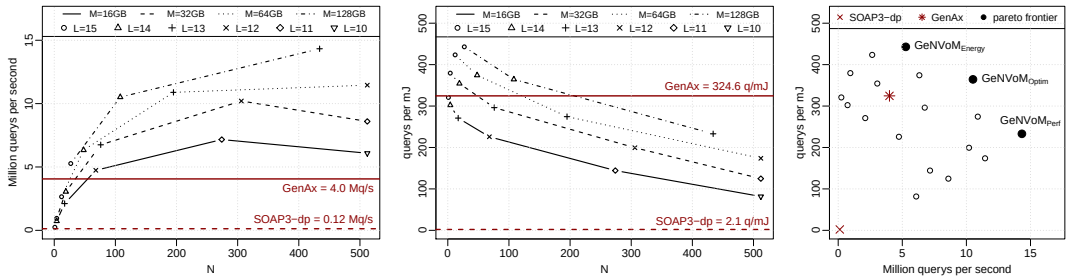
4.4.7 Design Space Exploration

We sweep M, C, L, N (Sect. 4.4.1) to extract the pareto fronts, and to identify the highest throughput ($GeNVoM_{Perf}$); highest energy-efficiency ($GeNVoM_{Energy}$); highest throughput and energy-efficiency ($GeNVoM_{Optim}$) configurations. M ranges from 16GB to 128GB; L, from 10 to 15. For each L, we find the maximum N where PMIT size and $N\times$PMITIL size fits in the given M budget. Besides, we cap N at 512, to limit C, number of LPDDR4-4266 channels feeding PMIs to FilterUs, at 64. For reference, Fig. 4.13 depicts the power vs. area trade-off for each evaluated GeNVoM configuration covering these ranges.

4.5 Evaluation

4.5.1 Throughput Performance and Energy

Larger *seed* values (i.e., L) prune the search space more, resulting in a progressively lower number of search operations in processing each *query*. For example, increasing L from 10 to 15 decreases average number of search operations per *query* from 30.2K to 4.6K. On the other hand, lower L leads to smaller PMITIL tables. As the memory



(a) Throughput Performance (b) Energy (c) Trade-off

Figure 4.14: Throughput performance and energy consumption.

footprint increases with N × PMITIL size, for a given memory budget M, lower L makes higher N possible. And higher N translates into more FilterU + MatchU pairs (each processing a portion of the *reference* in parallel). We will next look into this trade-off.

Fig. 4.14a captures the throughput. Y-axis shows the number of *query*s mapped per second; X-axis, the number of FilterU + MatchU pairs, N. The two horizontal lines correspond to the throughput of the two baselines, SOAP3-dp and GenAx, respectively. Each trendline corresponds to a different M budget; each point on each trendline, to a different L. We don't include L values which result in larger N than 512 (Sect. 4.4.7).

According to Fig. 4.14a, for each M, starting from L=15 (on the left), decreasing L generally makes higher N possible, and consequently increases throughput. However, lower L also prune the search space less, resulting in a progressively higher number of search operations in processing each *query*. For M=32MB (16GB), at L values lower than 12 (11), this effect starts to become dominant, and wipes off the throughput benefits of the higher N (i.e., more FilterU + MatchU pairs operating simultaneously). Besides, we also observe how larger M budgets improve the overall throughput. While most of the GeNVoM configurations render a higher throughput than the baselines (i.e., remain above both baseline lines), the fastest design, $GeNVoM_{Perf}$, is 113.5× (3.6×) faster than SOAP3-dp (GenAx) for $M = 128GB$, $L = 13$, $C = 55$, and $N = 434$. $GeNVoM_{Perf}$ consumes 35.3Watts on average and 298.3mm^2.

Fig. 4.14b captures energy-efficiency. Y-axis shows the number of *query*s mapped per mJoule of energy (q/mJ); X-axis, the number of FilterU + MatchU pairs, N. The

two horizontal lines correspond to the q/mJ of the two baselines. Similar to Fig. 4.14a, each trendline corresponds to a different M budget; each point on each trendline, to a different L.

According to Fig. 4.14b, energy efficiency degrades with decreasing L. The reason is two-fold: Lower L increase the number of search operations per *query* due to less effective search space pruning. At the same time, lower L make higher values of N possible, which increases the power consumption. Still, all (many) GeNVoM configurations are more energy-efficient than SOAP3-dp (GenAx). The most energy efficient design, $GeNVoM_{Energy}$, with 442.9 q/mJ, corresponds to $M = 128GB$, $L = 15$, $C = 4$, and $N = 27$. $GeNVoM_{Energy}$ improves energy-efficiency of SOAP3-dp (GenAx) by 210.9× (1.36×), while consuming 5.2Watts on average and 195.3mm^2.

Putting it all together, Fig. 4.14c depicts the trade-off space of throughput (x-axis) vs. energy-efficiency (y-axis) for the evaluated GeNVoM configurations and the baselines. Points closer to the top-right corner are faster and more energy-efficient. All GeNVoM configurations outperform SOAP3-dp in both throughput and energy efficiency, while a few are *both* faster and more energy-efficient than GenAx. At the pareto-frontier reside $GeNVoM_{Perf}$, $GeNVoM_{Energy}$, and also $GeNVoM_{Optim}$ which represents a sweet spot for both performance and energy-efficiency. $GeNVoM_{Optim}$ corresponds to $M = 128GB$, $L = 14$, $C = 14$, and $N = 108$, consuming 21.9Watts on average and 216.5mm^2. $GeNVoM_{Optim}$ can map 10.5M *query*s per second (84.0× faster than SOAP3-dp; 2.6×, than GenAx) and 364.4 *query*s per mJ (173.5× better than SOAP3-dp; 1.12×, than GenAx). All three pareto-frontier configurations correspond to $M = 128GB$, which indicates that GeNVoM benefits from larger memory space for both throughput and energy-efficiency.

Without loss of generality, for $L = 15$, GeNVoM maps 45.4% of the *query*s in Phase 1; 42.1% in Phase 2 (reverse complement); and 8.4%, in Phase 3. Specifically, in Phase 3, GeNVoM maps 3.0% of the *query*s after anchoring the first half; 2.6% after anchoring the second half; 1.8% after anchoring reverse complement of the first half; and 1.0% after anchoring reverse complement of the second half, respectively. This renders a mapping rate (i.e., the share of successfully mapped *query*s over all) of around 96.0% for GeNVoM, while SOAP3-dp can map 97.4% of the *query*s.

	GeNVoM$_{Energy}$	**GeNVoM$_{Optim}$**	**GeNVoM$_{Perf}$**	**SOAP3-dp**
Misalign. Rate	2.94%	2.97%	2.99%	1.12%
Miss Rate	0.03%	0.03%	0.04%	0.01%
Total	2.97%	3.0%	3.03%	1.13%

Table 4.2: Mapping accuracy of GeNVoM w.r.t. SOAP

4.5.2 Mapping Accuracy

Since we were not able to run experiments using GenAx, we compare the accuracy of GeNVoM to SOAP3-dp only. As SOAP3-dp outperforms the accuracy of BWA [104], which has a very similar accuracy to GenAx, SOAP3-dp is expected to be more accurate than GenAx.

To compare the mapping accuracy of GeNVoM to SOAP3-dp, we use a simulated input dataset with known expected matching indices. We differentiate between two cases: (1) the *query* is aligned to a wrong portion of the *reference*; or (2) the *query* is not aligned to any portion of the *reference*. Table 4.2 shows the corresponding rate of occurrence for (1) as the *Misalignment rate*; for (2), as the *Miss rate*, considering different configurations.

Misaligned *query*s are still mapped to a portion of the *reference*, which may be similar enough. Therefore, *misalignment rate* does not necessarily correspond to an error rate. The numbers in Table 4.2 reflect all problematic cases for GeNVoM due to prefix corruptions, indel mishandling or SA's false positives.

We further observe that both *misalignment rate* and *miss rate* slightly increase with larger L values (as we move right from GeNVoM$_{Energy}$ to GeNVoM$_{Perf}$), due to higher prefix corruption probability. However, the overall accuracy of different GeNVoM configurations considering different L values stay in a similar range. Generally, GeNVoM fails to align only around 0.03% more *query*s than SOAP3-dp, while misaligning around 1.85% more.

Next, we evaluate the effectiveness of GeNVoM's anchoring phase (Phase 3 from Section 4.3.4), in improving the accuracy of mapping over the first two phases. Without loss of generality, we use GeNVoM$_{Energy}$ as a case study, which fails to map 2.88% of the *query*s after the first two phases. However, Phase 3 manages to map 98.6% of the *query*s

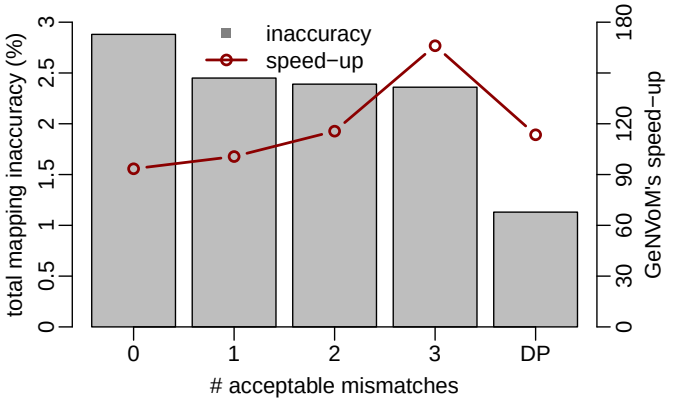


Figure 4.15: *tolerance* vs. accuracy and peformance.

that the first two phases missed (due to indels and prefix corruption), while aligning 91.2% of them to the correct index of the *reference*. This analysis demonstrates how effective multi-phase search is in improving the mapping accuracy when dealing with *query*s failed-to-map by the first two phases.

SOAP3-dp lets users trade accuracy for higher throughput. In this mode, SOAP3-dp turns off its complex, dynamic programming based mapping mechanism, and only considers substitutions. The user can set the accuracy level by tuning the number of acceptable mismatches, which is similar to the *tolerance* level of GeNVoM. A lower number of acceptable mismatches implies a lower *tolerance*. Lower *tolerance* leads to lower mapping accuracy (as chances of missing the mapping of even similar-enough *query*s with a few corruptions increases). At the same time, lower *tolerance* improves SOAP3-dp's throughput (as there is no need for sophisticated processing to cover complex corruptions). On the other hand, *tolerance* does not affect GeNVoM's throughput noticeably, since neither FilterU's nor MatchU's performance depend on it. More specifically, GeNVoM can tune *tolerance* by only adjusting SA's threshold, which negligibly affects TCAM search latency only.

Fig 4.15 depicts how SOAP3-dp's mapping (in)accuracy (left y-axis) changes with *tolerance* in terms of the number of acceptable mismatches (x-axis). The left y-axis corresponds to the *Total* row of Table 4.2. The bar labeled by *DP* captures the default for SOAP3-dp, which relies on complex, dynamic programming based mapping. The

trendline captures GeNVoM$_{Perf}$'s speed-up w.r.t. SOAP3-dp (right y-axis). We observe that, for smaller *tolerance* values, SOAP3-dp becomes less accurate, and at the same time faster, which decreases GeNVoM's speed-up over SOAP3-dp. The lowest inaccuracy of SOAP3-dp, 2.88% for a *tolerance* of 0, is close to the inaccuracy of GeNVoM$_{Perf}$. However, being 21.5% faster than the default SOAP3-dp, this point is still 93.4× slower thanGeNVoM$_{Perf}$. Hence, GeNVoM outperforms even an iso-accuracy baseline by approx. two orders of magnitude.

A note on *acceptability*: In a typical NGS *query* dataset, all *query*s, if concatenated back to back, would be at least 50× longer than the *reference* genome [120]. Therefore, even if we miss the mapping of a few percent of the *query*s (due to different read errors/genome variations), we still have plenty of *query*s to cover such missed regions of the *reference*. The average number of *query*s covering any given base in the *reference* genome is called the *depth*.

Only missing all *query*s covering a specific base of the *reference* would be problematic. What is the probability for GeNVoM to miss all *query*s covering a given base of the *reference*? Let the number of *query*s covering a given base be Q, and let $min(Q)$ denote its minimum. Q follows a Poisson distribution [121]. For a representative *depth* of 50, the probability of having a base covered by less than 10 *query*s is 5.2×10^{-12}, practically negligible. Therefore, we can assume that all bases are covered at least by 10 *query*s, i.e., that $min(Q) = 10$. The worst-case probability of GeNVoM missing a mapping, P_{miss} is around 3.03% (Table 4.2). Hence, the probability of GeNVoM missing all *query*s covering a base of the *reference* becomes ${P_{miss}}^{min(Q)} = 6.5 \times 10^{-16}$, which is practically negligible.

4.6 Related Work

(Short) Read Mapping: With no pre-processing of the *reference*, the computational complexity scales (at least) linearly with the *reference* length. Therefore, popular software implementations such as SOAP [122], Eland (part of the Illumina suite), and MAQ [123] adapt hash-based pre-processing. SOAP2 [124], Bowtie(2) [125, 126], BWA [127, 128], on the other hand, use the (more memory efficient) Burrows-Wheeler Transformation (BWT) of the *reference*. One baseline for comparison, SOAP3-dp [104], represents

an open-source GPU implementation of BWA, which can also handle noise in *read*s, however, unlike GeNVoM, the computational complexity depends on the *tolerance* value. Genax [119], the other baseline for comparison, is an automata-based accelerator. Similar to SOAP3-dp, computational and space complexity of GenAx depend on the *tolerance* value, unlike GeNVoM. High-throughput FPGA implementations [129] also exist, at the expense of orders of magnitude higher power consumption than GeNVoM. The exotic race logic based dynamic programming accelerator [130] can find the similarity between two strings corresponding to the *read* and a sub-sequence of the *reference* in approx. 120ns while consuming 1nJ. This is much slower than GeNVoM, and energy grows with the third power of *read* length which can impair scalability.

Another study introduced an efficient filtering step for hash-based *read mapping* using 3D-stacked memories [131]. This covers filtering (i.e., search space pruning) only, which has a similar functionality to GeNVoM's FilterU. While the high cost of quadratic-time dynamic programming algorithms for string (i.e., *query*) matching motivated this work, GeNVoM relies on much faster and more energy-efficient string matching enabled by resistive TCAM. Hence, GeNVoM employs a simpler, low-latency filtering (incorporated in FilterU), which is tailored to its more efficient string matching (incorporated in MatchU). Although higher-overhead methods like [131] can prune the search space further, embedding such into GeNVoM's FilterU would be overkill, rendering FilterU the system bottleneck and diminishing the benefits from MatchU's fast and energy-efficient similarity matching. We believe that proposals like [131] are more suitable for *long read mapping*, which represents a different problem. Besides GenAx [119], recent work also includes a BWA-based hardware accelerator featuring NVM [132], which is significantly slower than GeNVoM; and very effective hardware acceleration for other significant bioinformatics algorithms [133].

Resistive CAM Accelerators: Guo et al. [98, 134] explore TCAM for accelerating data-intensive applications. Yavits et al. [135] propose an associative processor, which employs resistive CAM based look-up tables to implement diverse functions. Kaplan et al. [136] demonstrate how to efficiently implement Smith-Waterman algorithm (which tries to align strings of similar length) using resistive CAM based look-up tables. Not being able to handle similarity matching under noise, neither of these are directly applicable to *read mapping*. Approximate resistive CAM is also proposed, either using

exotic cells [137] (which complicates match detection, and thereby restricts the row length to at most 8 bits) or limiting the row length to 4 bits only (to find the Hamming distance, their similarity metric, robustly) [138]. While these show great potential, restricted row length hinders applicability to *read mapping* where longer (short) *read*s are emerging with NGS improvements and where TCAM search happens at row-granularity. GeNVoM adds support for approximate matches much less intrusively, by carefully tuning the sense amplifier (SA) reference voltage in a variation-aware manner, without restricting the row size. No CAM array capable of only approximate matching would be sufficient to implement an efficient *read mapping* accelerator by itself, as demonstrated in Section 4.2.2.

Recent representative demonstrations of resistive TCAM include a 1Gbit PCM-based CAM from IBM using IBM 90nm technology [14] with a measured search latency of 1.9ns, and two novel spintronic designs in 45nm [139], where a search takes approx. 0.6ns in 256-wide rows. GeNVoM can adapt any resistive CAM array, including these more recent proposals.

Chapter 5

VARIUS-TC: A Modular Architecture-Level Model of Parametric Variation for Thin-Channel Switches

5.1 Introduction

Both, aggressive miniaturization and rapid introduction of unconventional materials and device architectures (which in turn enables aggressive miniaturization) make the manufacturing process of contemporary digital switches less controllable. At the same time, device-level optimizations to facilitate robust operation under aggressive miniaturization often require modification of the manufacturing process itself, which usually gives rise to new sources of variation in process parameters. As a result, deviation of device parameters from their nominal design specification becomes more likely, rendering performance and power efficiency of manufactured hardware greatly unpredictable. Unlocking the performance and power efficiency benefits of emerging switches hence mandates characterization of variation-incurred unpredictability across the system stack. This necessitates dependable architecture-level models of variation to enable accurate evaluation of system-level implications at early stages of the design.

The speed at which unconventional materials, device architectures and optimizations come to light can easily prohibit the longevity of a tightly calibrated architecture-level model of parametric variation. To overcome this challenge, we introduce VARIUS-TC, a dependable model which decouples architecture-level analysis from circuit- and device-level characterization by careful abstraction. VARIUS-TC comprises three modules which span device, circuit, and architecture layers of the system stack under process variation, respectively:

- The *device module* encapsulates switch-level electrical characteristics such as current-voltage profiles. Closed-form formulae or look-up tables (LUT) can capture electrical characteristics. LUTs can be particularly useful in analyzing emerging devices where no known closed-form formulae exist.
- The *circuit module* encompasses performance and power characteristics at logic gate and memory cell granularity. The output of the device module represents the input of the circuit module.
- The *architecture module* derives the performance and power characteristics at pipeline and (on-chip) memory block granularity. The output of the circuit module represents the input of the architecture module.

By swapping corresponding variants of device and/or circuit modules, VARIUS-TC can keep track of architecture-level implications of process variation as novel switches get discovered. In this study, we focus on a representative class of emerging thin-channel switches, FinFETs, as a case study. In the following, Section 5.2 provides the background; Sections 5.3 and 5.4 explain the parametric variation model; Sections 5.5 and 5.6 cover the evaluation and a representative use case of VARIUS-TC; and Section 5.7 compares and contrasts VARIUS-TC to related work.

5.2 Background

5.2.1 Thin-Channel Switches

The G(ate) of a classic digital switch controls the formation of a conductive path – the channel – between the S(ource) and D(rain) terminals, to turn on the switch. An

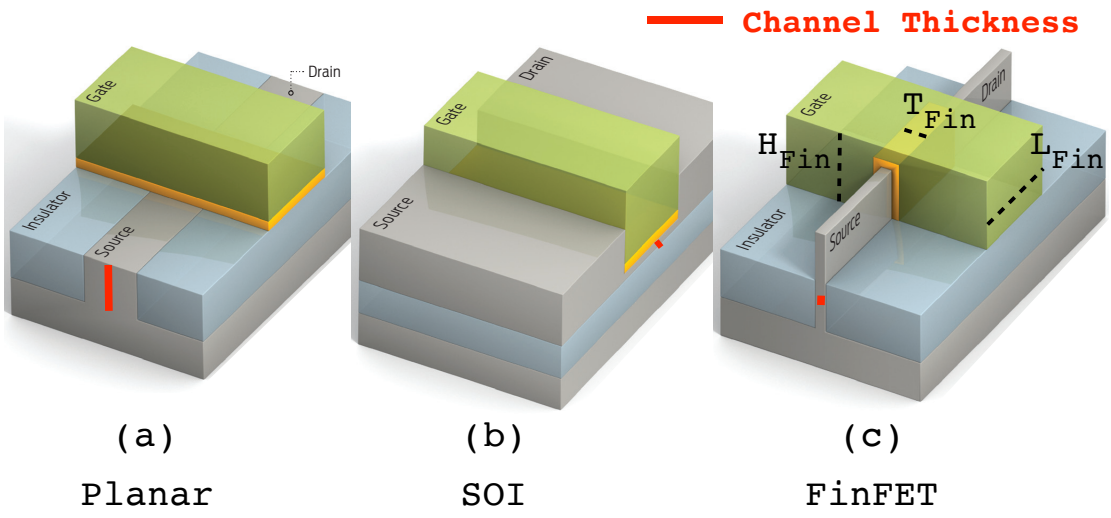


Figure 5.1: Planar vs. thin-channel switches. Adapted from [16].

ideal switch prohibits current flow when turned off: $I_{on}/I_{off} \rightarrow \infty$, where I_{on} (I_{off}) corresponds to the current flow between S and D when the switch is on (off). Under aggressive miniaturization the distance between S and D becomes so small that G loses control over the channel, resulting in a a non-negligible I_{off}. To prevent excessive growth of I_{off}, G needs stronger control over the channel. Two emerging classes of device architectures, ultra-thin body silicon on insulator (UTB-SOI) and fin field effect transistor (FinFET), achieve this by eliminating excess silicon material between S and D, i.e., by making the channel thinner [16]. Fig. 5.1 depicts the device architecture for a classic old-school switch, a *planar* metal-oxide field effect transistor (MOSFET) in (a); UTB-SOI in (b); and FinFET in (c), with the channel thickness explicitly marked. Without loss of generality, the rest of the paper is confined to FinFETs as representative thin channel devices. FinFET switches have their channel rotated by 90° with respect to the planar design, which results in a *fin* sticking out of the substrate plane (Fig. 5.1(c)). The gate surrounds the fin for enhanced channel control, which in turn prevents excessive I_{off}.

5.2.2 Process Variation

Due to the stronger control of the gate over the channel, FinFET features better performance and power characteristics than planar MOSFET [140], however, the 3D structure (Fig. 5.1(c)) complicates the manufacturing process, and introduces new sources of variation. The main challenge stems from the manufacture of thin and uniform fins [16].

Die-to-die (D2D) variation in process parameters mainly stems from *systematic*, i.e., strongly correlated, effects such as lithographic aberrations. D2D variation causes similar changes in the values of switch parameters across the die. Within die variation (WID) in process parameters, on the other hand, can be due both *systematic* and *random* (i.e., uncorrelated) effects such as random dopant fluctuation. Each switch in the die may have a different change in the value of each affected parameter due to WID variation. In the following, we focus on the system-level impact of WID variation in FinFET parameters. A global offset per die (for each affected parameter) can capture D2D variation on top.

All geometric dimensions shown in Fig. 5.1(c) – fin thickness, fin height, and channel length, i.e., T_{Fin}, H_{Fin}, and L_{Fin} – are subject to variation, along with the oxide thickness, T_{ox}, and metal gate work function, Φ_g [141, 142, 143]. Variation in Φ_g comes from the variation in the orientation of the gate metal grains. The threshold voltage, V_{th}, strongly depends on fin dimensions – all subject to variation [141], and governs both performance and power. Variation in V_{th} degrades operating speed by increasing the variance of the critical path distribution. Higher variance causes longer tails in the critical path distribution, where the fastest and the slowest paths reside. In the end, the slowest path determines the operating speed. Moreover, variation in V_{th} induces a higher static power consumption: for the same variation-induced Δ change in V_{th}, switches with $-\Delta$ leak more than switches with $+\Delta$ save. The variation in L_{Fin}, T_{Fin}, and Φ_g has the highest impact on performance and (static) power [144, 143].

5.3 VARIUS-TC: Macroscopic View

Parametric variation can easily degrade the operating speed and increase the static power consumption significantly, hence wipe out performance and power benefits of emerging switches such as FinFET. Accordingly, accurate characterization of variation-incurred

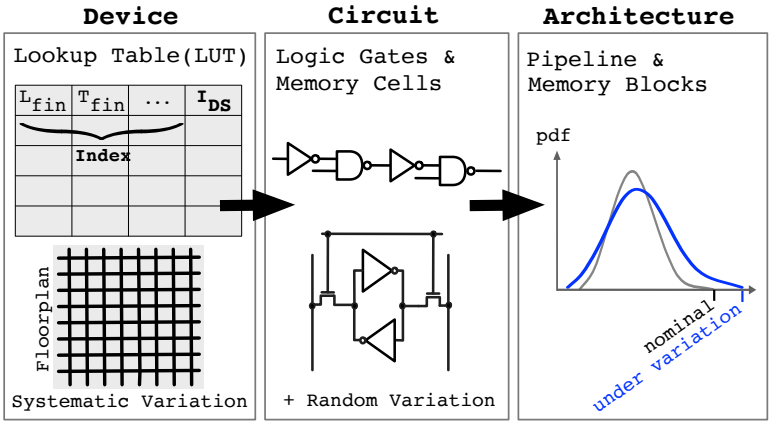


Figure 5.2: Overview of VARIUS-TC.

unpredictability at early stages of the design is critical. Fig. 5.2 provides an overview of VARIUS-TC, a modular, dependable architecture-level variation model. VARIUS-TC features three basic modules which span device-, circuit-, and architecture layers of the system stack. Inputs to VARIUS-TC are the chip floorplan at functional block granularity and physical FinFET parameters for the respective technology node.

5.3.1 Device Module

The *device module* encapsulates closed-form formulae or look-up tables (LUT) to derive I_{on} and I_{off} from (variation-afflicted) physical FinFET parameters. As high-fidelity closed-form formulae can easily become computationally expensive for an architecture-level model, VARIUS-TC provides support for LUTs. The index (i.e., input query) to the LUT is a vector comprising V_{DS} (the drain-source voltage), V_{GS} (the gate-source voltage), T_{ox}, Φ_g, T_{Fin}, H_{Fin}, and L_{Fin}, respectively. The drain-source current I_{DS} represents the LUT output. Depending on the values of V_{GS} and V_{DS}, I_{DS} may correspond to I_{on} or I_{off}.

LUT Generation: Each row of the LUT corresponds to a specific index vector $< V_{DS}, V_{GS}, T_{ox}, \Phi_g, T_{Fin}, H_{Fin}, L_{Fin} >$. We sweep the value of T_{ox}, Φ_g, T_{Fin}, H_{Fin}, and L_{Fin} over a specific range, at a specific resolution. Both the range and the resolution represent VARIUS-TC parameters which evolve as a function of the anticipated (maximum) deviation of physical FinFET parameters from their nominal values under

variation. V_{DS}, V_{GS} values reflect anticipated operating voltages. For each combination, a SPICE simulation generates the corresponding I_{DS} by utilizing a technology model file (such as Predictive Technology Model (PTM) [145, 146]) which tabulates low-level FinFET parameters. Each such index vector along with the corresponding I_{DS} represents a row of the LUT. A finer parametric resolution comes at the cost of increased storage overhead. The LUT also considers different temperature values.

Generation of Systematic Variation Maps: In addition to LUT generation, the device module is in charge of the preparation of systematic variation maps. Following [141, 143] VARIUS-TC models the variation in T_{ox}, Φ_g, T_{Fin}, H_{Fin}, and L_{Fin} by separate Gaussian distributions, each characterized by a different σ/μ. The nominal values from the technology model file represent the (expected value) μ. The (standard deviation) σ represents yet another VARIUS-TC parameter. In order to capture within-die (WID) systematic variation, the device module super-imposes a grid on the chip floorplan and assigns to each grid point a sample from the Gaussian distributions of T_{ox}, Φ_g, T_{Fin}, H_{Fin}, and L_{Fin}, respectively. WID variation of each parameter exhibits spatial correlation, which VARIUS-TC captures by a spherical function: For each parameter subject to variation, the correlation between two switches on die only depends on their Euclidean distance. The correlation assumes its maximum value at distance zero, decreases with increasing distance, and vanishes once the distance exceeds the *correlation range*, ϕ. Similar to [147, 148], VARIUS-TC captures random variation analytically: The circuit module (Section 5.3.2) combines the systematic and random variation in deriving gate- and (memory-)cell-level performance and power characteristics under variation.

5.3.2 Circuit Module

The *circuit module* uses the outcome of the device module – the LUT and systematic variation maps – in deriving gate- and (memory-)cell-level performance and power characteristics under variation. Systematic variation maps incorporate profiles for all, T_{ox}, Φ_g, T_{Fin}, H_{Fin}, and L_{Fin}. The circuit module extracts the vector $< T_{ox}, \Phi_g, T_{Fin}, H_{Fin}, L_{Fin} >$ for each grid point from the systematic variation map and uses this as the query vector to retrieve the corresponding I_{DS} from the LUT. Next, the module plugs I_{DS}, along with the query vector, into the performance model derived from [149]. VARIUS-TC mimics VARIUS(NTV) to factor in random variation analytically in deriving both performance

and power, and in calculating the minimum safe operating voltage, V_{MIN}, per memory cell, under variation.

5.3.3 Architecture Module

Following VARIUS(NTV) methodology, the *architecture module* uses the outcome of the circuit module (1) to extract the critical path distribution within a pipeline stage or a memory block; (2) to determine the minimum safe operating voltage of each memory block, V_{MIN}; (3) to calculate the minimum safe clock period, τ_{MIN}, at a given supply voltage V_{dd}; and (4) to report the probabilities of variation-induced logic and memory (timing or stability) errors as a function of the operating point (i.e., the clock period and supply voltage). A timing error in a logic block emerges if variation-incurred slowdown prevents safe operation at the designated clock period. Similarly, a timing error in a memory block emerges during a read or write, if variation-incurred slowdown prevents completion of the read or write within the designated time window. A stability error is the case, if excessive leakage under variation corrupts the memory content even if the memory is not accessed.

5.4 VARIUS-TC: Microscopic View

5.4.1 Capturing Variation in Logic Blocks

A vector of variation-afflicted physical FinFET parameters, $< T_{ox}, \Phi_g, T_{Fin}, H_{Fin}, L_{Fin} >$, represents the query index to the LUT (Sections 5.3.1, 5.3.2). VARIUS-TC always extracts the LUT entry with the minimum difference to the query index. After retrieving the respective (variation afflicted) I_{on} from the LUT for a given variation profile, *circuit module* determines the variation in gate delay from CI_{on}/V_{dd}, where C represents the equivalent load capacitance, and V_{dd}, the operating voltage. In this manner, VARIUS-TC implicitly considers I_{on}'s dependence on V_{th} and temperature. C is mainly a function of the floorplan and practically does not change with variation.

Using circuit module's gate delay distributions, VARIUS-TC's *architecture module* determines the path delay distribution of the slowest pipeline stage under variation following VARIUS(NTV) methodology [147, 148]. The path delay distribution under

variation, D_{Var}, dictates the maximum path delay $max(D_{Var})$ which in turn determines the minimum possible value of the clock period t_{CLK}. The timing error rate per cycle while operating at a given clock period t_{CLK} becomes $1 - cdf_{D_{Var}}(t_{CLK})$, where cdf represents the cumulative distribution function. $cdf_{D_{Var}}(t_{CLK})$ captures the cumulative probability of path delays (in the respective pipeline stage) to assume a lower value than t_{CLK}. A timing error occurs iff any path delay exceeds the designated clock period t_{CLK}.

A pipeline stage encompasses many paths of different delays. Even if there was no variation, there exist a specific distribution of path delays, D_{Logic}. Variation transforms this distribution to $D_{VarLogic}$:

$$\begin{aligned} D_{Logic} &= D_{Gates} + D_{Wire} \\ D_{Wire} &= k_W \times D_{Logic} \\ D_{Gates} &= (1 - k_W) \times D_{Logic} \end{aligned} \tag{5.1}$$

D_{Gates} from Equation 5.1 corresponds to the delay of a sequence of gates along a path modulo wires; D_{Wire}, to the wire delay, were there no variation. VARIUS-TC neglects variation in wire delay. The multiplier k_W captures the share of wire delay in D_{Logic}.

Equation 5.2 gives the path delay distribution of a (purely logic) pipeline stage under variation. $D_{VarGates}$ reflects changes in D_{Gates} under variation considering both the systematic ($D_{VarSysGates}$) and random ($D_{VarRandGates}$) components:

$$\begin{aligned} D_{VarLogic} &= D_{VarGates} + D_{Wire} \\ &= D_{SysGates} + D_{RandGates} + D_{Wire} \\ D_{SysGates} &= k_{Sys} \times D_{Gates} = k_{Sys} \times (1 - k_W) D_{Logic} \end{aligned} \tag{5.2}$$

Under aggressive miniaturization, the footprint of a pipeline stage becomes so small, that all of its paths fall within the correlation distance ϕ of systematic variation. Therefore, being highly correlated, parameters of all gates along a path in a (purely logic) pipeline stage change practically in the same direction by the same quantity. The coefficient k_{Sys} captures this change due to systematic variation. The random component does not exhibit any correlation, hence a similar coefficient to k_{Sys} does

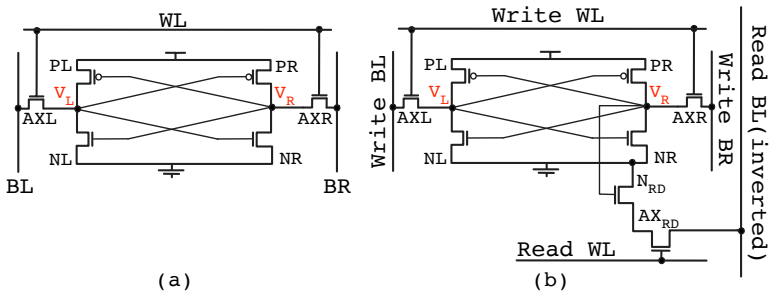


Figure 5.3: 6-T(ransistor) (a) and 8-T cell (b). V_R and V_L are the voltages at nodes R and L, respectively.

not apply. VARIUS-TC derives the random component by composing independent and identically distributed gate delay distributions (under random variation) instead.

5.4.2 Capturing Variation in On-chip Memory Blocks

When compared to logic, on-chip memory blocks are more susceptible to variation due to both the higher density (i.e., higher number of transistors per area) and smaller size transistors. VARIUS-TC models the variation in a conventional 6T(ransistor) cell (Figure 5.3(a)) following VARIUS(NTV) methodology [147, 148]. The cell consists of two inverters, formed by *PR-NR* and *PL-NL* in a positive feedback loop, and two access transistors, *AXR* and *AXL*. V_R stores the cell's value and V_L its complement. To read from or write to the cell, word-line *WL* is driven high to connect the cell to the bit-lines *BL* and *BR*. To read, the bit-lines are pre-charged to logic high. To write, *BR* is pre-conditioned to the value to be written, and *BL*, to its complement. In the following, we assume that V_R=0 without loss of generality. Since the cell is symmetric, the discussion applies directly to V_R=1.

If, during a read, the time needed to produce a voltage difference between the two bit-lines exceeds the period that *WL* stays high, a timing error occurs. Variation-induced increases in V_{th} of the discharge transistors, AXR and/or NR can trigger timing errors during read, because the transistors become slower. A timing error can also occur during a write, if the write is unable to change the logic state of the target cell by the end of the designated write duration. Variation-induced shifts in the switching threshold of the

PR/NR inverter and/or PL becoming stronger than AXL under variation can trigger such errors. Variation can also distort the logic value stored in the cell due to excessive leakage of the transistors forming the inverters, even if the cell is not being accessed: Variation-induced shifts in V_{th} of the transistors forming the inverters, which increase the leakage, trigger such stability errors.

VARIUS-TC supports the 8-T(ransistor) cell of Figure 5.3(b) [150], as well, which is optimized for low power operation. This cell is easier to design reliably because it decouples the transistors used for reading from those used for writing such that they can be optimized separately. Specifically, the two additional transistors N_{RD} and AX_{RD} are only responsible for reads, while the rest coordinates the writes exactly as in the 6T cell.

For a cell storing 0 ($V_R = 0,\ V_L = 1$), a (non-redundant) write completes once V_L becomes logic 0. V_L represents the input to the inverter formed by transistors *PR-NR*, the output of which determines V_R. Therefore, as V_L becomes logic 0, this inverter forces V_R to logic 1. Accordingly, to model write timing errors, VARIUS-TC extracts $D_{VarWriteCell}$, the time that node L takes to reach the switching threshold (V_{SWITCH}) of the *PR-NR* inverter – V_{SWITCH} in this case corresponds to the maximum voltage to be interpreted as logic 0 by the inverter. In doing this, the architecture module closely follows the methodology from [148]. The device and circuit modules, on the other hand, deviate from [148] due to the utilization of the LUT for the extraction of respective currents. After obtaining the probability distribution for $D_{VarWriteCell}$, VARIUS-TC computes the distribution of the maximum of $D_{VarWriteCell}$ over all the cells in a line, $D_{VarWriteLine}$. The probability

$$P[D_{VarWriteLine} > t_{WRITE}]$$

gives the probability of a write timing error, where t_{WRITE} is the designated write duration.

Relaxing timing constraints (e.g., increasing the designated clock period, t_{CLK}) can help mitigate timing errors, however, no such remedy applies to stability errors. For a cell storing 0 ($V_R = 0,\ V_L = 1$), at low V_{dd}, the voltage V_L reduces by construction. When the cell is not accessed, if V_L reduces enough – due to leakage through NL and AXL, to reach the V_{SWITCH} of the *PR-NR* inverter, the cell content can easily get distorted. A stability error occurs when the leakage current through the NL and AXL

transistors in Figure 5.3(b) reduces V_L below the V_{SWITCH} of the *PR-NR* inverter while the cell is not being accessed. At that point, the cell's state gets lost.

VARIUS-TC's architecture module captures stability errors following [148]; the key difference from [148] is the adoption of the LUT by the device and circuit modules in calculating the respective currents. VARIUS-TC calculates the stability error probability per cell by

$$P_{Cell,Err} = P[V_L(Vdd) - V_{SWITCH}(Vdd) < 0].$$

If no redundant cells are available,

$$P_{Line,Err} = 1 - (1 - P_{Cell,Err})^{line\ size}$$

gives the corresponding error probability of a line, where *line size* denotes the number of cells per line; and $1 - (1 - P_{Cell,Err})^{line\ size}$, the probability that at least one cell fails. The error probability per memory block becomes

$$P_{Mem,Err} = 1 - (1 - P_{Line,Err})^{number\ of\ lines}$$

in this case, with $number\ of\ lines$ denoting the number of lines in the respective memory block. VARIUS-TC can also calculate the minimum allowable supply voltage to avoid such errors: $Vdd_{MIN,Cell}$ by solving

$$V_L(Vdd_{MIN,Cell}) = V_{SWITCH}(Vdd_{MIN,Cell})$$

for voltage under variation, where $V_L(Vdd_{MIN,Cell})$, and $V_{SWITCH}(Vdd_{MIN,Cell})$, respectively, denote the values of V_L and V_{SWITCH} at $V_{dd} = Vdd_{MIN,Cell}$. VARIUS-TC calculates $Vdd_{MIN,Line}$ by $max(Vdd_{MIN,Cell})$.

5.4.3 Capturing Variation in Static Power

VARIUS-TC derives the static power distribution by integrating the static power distribution per switch under variation, over all switches within the encapsulating logic or memory block. $V_{dd} \times I_{off}$ gives the per switch static power under variation, with V_{dd} being the operating voltage; and I_{off}, the leakage current under variation as generated/fetched from the LUT by the device/circuit modules.

Table 5.1: FinFET technology nodes [145].

Parameter	Unit	PTM 10nm	PTM 16nm	PTM 20nm
L_{Fin}	nm	14	20	24
T_{Fin}	nm	8	12	15
H_{Fin}	nm	21	26	28
T_{ox}	nm	12	13.5	14
ϕ_g	eV	4.42	4.40	4.37
V_{NOM}	V	0.75	0.85	0.9

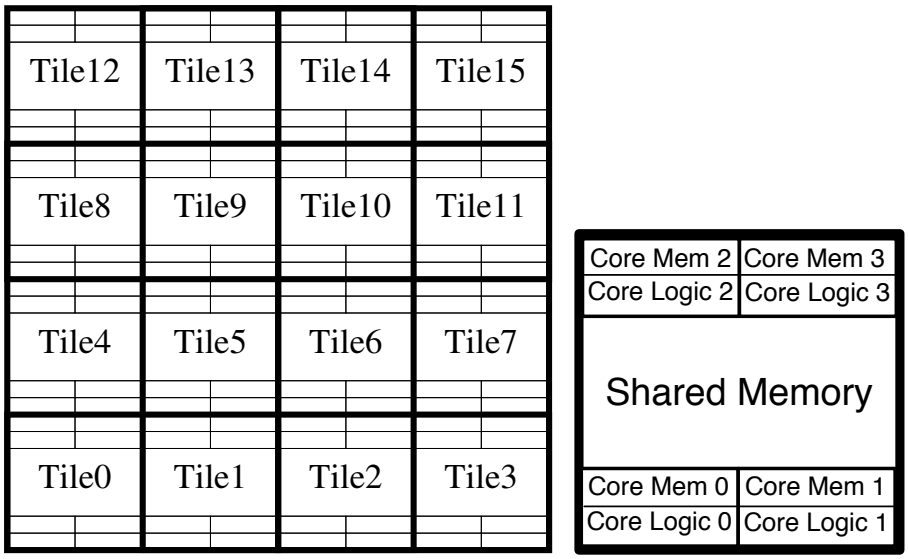
Table 5.2: Different levels of variation used in evaluation.

Parameter	Low var.	Medium var.	High var.
L_{Fin}	3.5%	7%	10.5%
T_{Fin}	3.5%	7%	10.5%
ϕ_g	0.16%	0.32%	0.48%

5.5 Evaluation Setup

We evaluate VARIUS-TC for different FinFET technology nodes from PTM, as listed in Table 5.1. As the baseline for comparison, we deploy 16nm PTM high-performance (HP) for planar MOSFET, where the nominal V_{th}, V_{thNOM} is 0.48V, and the nominal effective channel length, 16nm. We set σ/μ for cumulative variation to 5% which results in $\approx$ 3.5% of systematic and random variation under equal share. We consider three different levels of variation – *low*, *medium*, and *high*, with the corresponding σ/μ of FinFET parameters given in Table 5.2. To generate systematic variation maps, we set the correlation distance ϕ to 0.1 [147, 148].

We repeat each experiment for 100 dies, and report the characteristics for the median die. The chip is organized in 4×4 clusters (Figure 5.4). Each cluster has 4 in-order Intel Xeon Phi [151] like cores (each of 1GHz and with 32KB L1 private instruction and data cache) and 2MB shared (4-bank) L2 cache. We deploy 8T memory cells. The intra-cluster interconnection network is a bus; the inter-cluster, a 2D torus. We deploy the microarchitectural simulator Sniper-6.0 [59] integrated with the architectural power model McPAT [58] (scaled to 16nm) for running PARSEC-3.0 [56] applications using simsmall input with 64 threads. We experiment with BS:*blackscholes*, BT:*bodytrack*, CN:*canneal*, FR:*ferret*, FA:*fluidanimate*, SC:*streamcluster*, SW:*swaptions*, and XX:*x264*.



(a) chip (b) tile

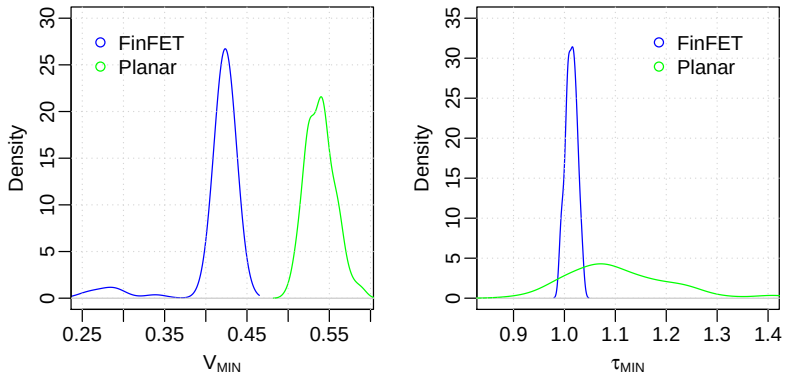
Figure 5.4: Floorplan of the evaluated manycore architecture.

5.6 Evaluation

5.6.1 FinFET vs. Planar MOSFET under Variation

We first characterize the WID variation in minimum safe operating voltage, V_{MIN}, and minimum safe clock period at the nominal voltage, τ_{MIN} considering 16nm FinFET and 16nm planar MOSFET based designs. V_{MIN} represents the minimum safe operating voltage for each memory block, which practically prevents the onset of stability errors under variation. τ_{MIN}, on the other hand, is the minimum clock period at which each core can safely operate (at the nominal supply voltage) under variation. Operation at τ_{MIN} practically prevents the onset of timing errors.

Fig. 5.5(a) depicts the kernel density estimate for V_{MIN}; Fig. 5.5(b), for τ_{MIN}. Kernel density estimates correspond to continuous histograms, with the area under the curve = 1. The nominal supply voltage, V_{NOM} is 0.85V for FinFET, and 0.7V for planar MOSFET, respectively. We deploy the *high variation* profile from Table 5.2 for FinFET, and the *low variation* equivalent (where variation in L_{Fin} corresponds to the variation in effective channel length) for planar MOSFET, to favor the planar MOSFET based



(a) V_{MIN} (b) τ_{MIN}

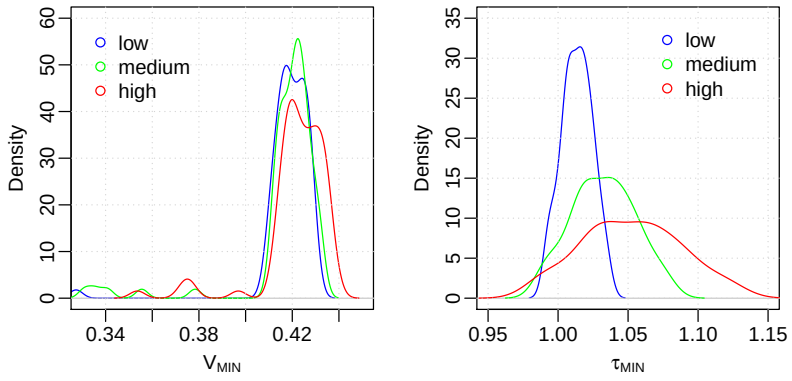
Figure 5.5: WID variation of FinFET vs. planar MOSFET.

design.

Fig. 5.5(a) captures WID variation for V_{MIN} across all memory blocks on chip; Fig. 5.5(b), for τ_{MIN} across all cores. The x-axis of Fig. 5.5(a) depicts the WID V_{MIN} spread in Volts. We observe that the FinFET based design outperforms its planar counterpart: The planar MOSFET based design renders a maximum V_{MIN} of 0.59V; the FinFET based, of only 0.46V. At the same time, the V_{MIN} spread across die remains notably larger for the planar case. This difference in the spread of the distributions is even more pronounced for τ_{MIN}. In Fig. 5.5(b), the x-axis is normalized to the nominal critical path delay, τ_{NOM}, when operating at the nominal operating voltage, V_{NOM} (were there no variation). The planar MOSFET based design renders a maximum normalized τ_{MIN} of 1.412; the FinFET based, of only 1.035. Our results confirm previous studies which report enhanced resilience of FinFET based designs to variation [140].

5.6.2 Impact of Variation Level

We next analyze how WID variation in V_{MIN} and τ_{MIN} changes as a function of the variation in each critical physical FinFET parameter (Section 5.2.2). As in Section 5.6.1, we use the 16nm PTM FinFET. Fig. 5.6 captures the trend for the three different variation profiles – *low*, *medium*, and *high* – from Table 5.2, following the same format as Fig. 5.5. We observe that the spread of both V_{MIN} and τ_{MIN} distributions increase



(a) V_{MIN} (b) τ_{MIN}

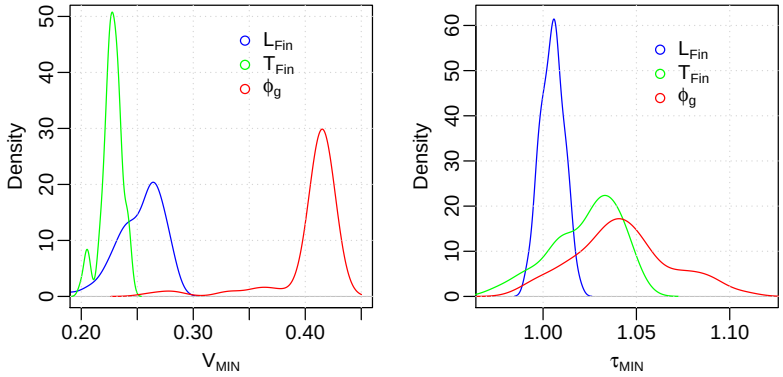
Figure 5.6: Impact of different levels of WID variation.

as the level of variation increases, to give rise to longer tails. When compared to V_{MIN}, τ_{MIN} exhibits higher sensitivity to the level of variation.

5.6.3 Sensitivity Analysis

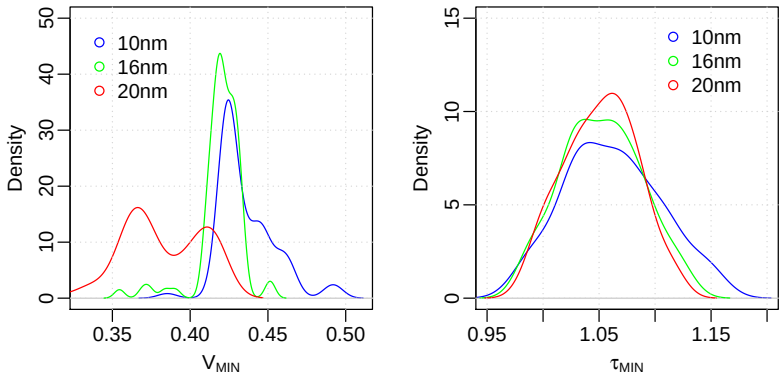
Sensitivity to Physical FinFET Parameters: The variation in L_{Fin}, T_{Fin}, and Φ_g has the highest impact on power and performance [144, 143]. Accordingly, we confine our sensitivity analysis to these three parameters, and consider each in isolation in Fig. 5.7. We experiment with the 16nm PTM FinFET under *high* variation (Table 5.2). In line with previous work [143], we observe that for both, V_{MIN} and τ_{MIN} variation, Φ_g represents the critical parameter. Φ_g predominantly delivers the worst WID variation profiles, and its impact is higher on the distribution of V_{MIN}. The maximum value of V_{MIN} becomes 0.29V under *high* variation for L_{Fin} (in isolation); 0.25V for T_{Fin} (in isolation); and 0.42V for Φ_g (in isolation), respectively. The maximum value of τ_{MIN} becomes 1.019 for L_{Fin}; 1.051 for T_{Fin}; and 1.108 for Φ_g, in isolation.

Sensitivity to Technology Scaling: Fig. 5.8 captures the sensitivity to technology scaling, considering PTM FinFET at 20nm, 16nm, and 10nm under *high* variation (Table 5.2). We observe that the impact of WID variation increases with technology scaling, for both, V_{MIN} and τ_{MIN}, to render longer tails. Our analysis so far covers PTM nodes optimized for H(igh) P(erformance). PTM L(ow) P(ower) nodes render



(a) V_{MIN} (b) τ_{MIN}

Figure 5.7: Sensitivity to physical FinFET parameters.



(a) V_{MIN} (b) τ_{MIN}

Figure 5.8: Sensitivity of WID variation in V_{MIN} (a) and τ_{MIN} (b) to technology scaling, considering PTM model files optimized for high performance (HP).

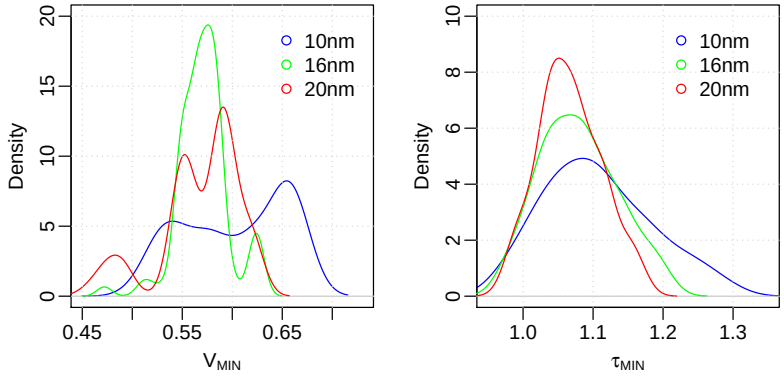


Figure 5.9: Sensitivity of WID variation in V_{MIN} (a) and τ_{MIN} (b) to technology scaling, considering PTM model files optimized for low power (LP).

similar trends, as shown in Fig. 5.9, when we repeat these experiments.

5.6.4 Impact on Static Power

Fig. 5.10 characterizes how chip-wide static power changes when compared to the no-variation case, under three different levels of variation (Table 5.2) for PTM HP FinFET at 16nm. We observe that the static power increases notably, by 2.04×, 4.51×, and 9.52×, under *low*, *medium*, and *high* levels of variation.

5.6.5 An Example Use Case

Under contemporary technology scaling, one promising way to cram more cores into the available power budget is reducing the operating voltage V_{dd} aggressively. If V_{dd} remains slightly above the threshold voltage V_{th}, power consumption can decrease by more than an order of magnitude [152, 153]. This unconventional regime of operation, Near-Threshold Voltage (NTV) Computing, enables more cores to operate simultaneously. Power savings increase with the proximity of the near-threshold V_{dd} to V_{th}. Unfortunately, as V_{dd} reaches V_{th}, not only degrades the (minimum) clock period τ_{MIN}, hence, the (maximum) operating frequency f ($\propto 1/\tau_{MIN}$), but also, resilience to variation weakens.

For embarrassingly parallel applications, we can avoid performance degradation by

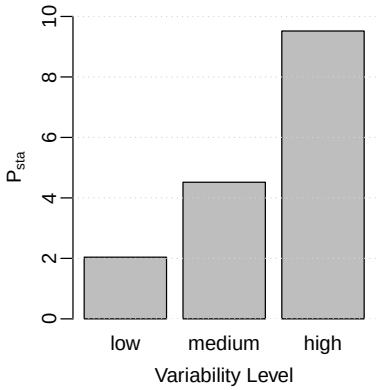
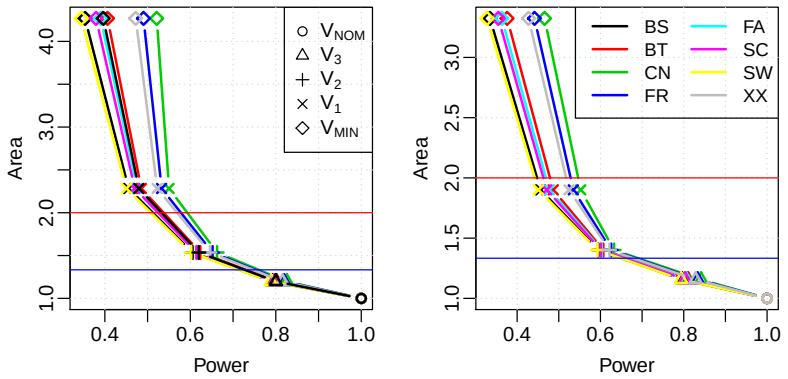


Figure 5.10: Variation in (normalized) static power, P_{sta}.



(a) under *high* variation (b) under *low* variation

Figure 5.11: Area vs. power trade-off as a function of the operating voltage for PTM HP FinFET at 16nm.

distributing (the same) computation[1] to more cores. While each core operates at the degraded f at NTV, the execution time can still reduce due to the reduced amount of work per core. At the same time, power savings due to NTV operation exceed the power cost of more cores participating in computation. This solution, however, is only viable if the available chip area can accommodate the higher number of active cores required to mask the f degradation at NTV.

Fig. 5.11 captures the area overhead (y-axis) along with the corresponding power consumption (x-axis) as a function of the operating voltage, for the *high* (a) and *low* (b) variation profiles from Table 5.2, in compensating the f degradation at NTV by distributing computation to more cores, for the benchmark applications from Section 5.5. The axes are normalized, respectively, to the area and power of the non-variation-afflicted baseline operating at V_{NOM}. For both profiles, the slowest of the variation-afflicted cores determines the degraded operating frequency $f$at any given operating voltage. Each shape captures a different voltage value, and each trendline, a different benchmark application.

High variation renders a higher slowdown than *low* variation, hence demands more active cores (and incurs a higher area overhead) to compensate for the degraded f. Accordingly, the span of the y-axis is larger in Fig. 5.11(a). For each profile, we consider three voltage points, uniformly sampled from $[V_{MIN}, V_{NOM}]$ range. Due to the difference in the variation profiles, (chip-wide) V_{MIN} values also differ between Fig. 5.11(a) and (b), and so do the values of the voltage samples, V_1, V_2, and V_3 (in ascending order) [2]. We deploy VARIUS-TC (i) to extract V_{MIN}; (ii) to calculate the safe f and power consumption at each voltage level.

Featuring the maximum power savings at minimum area overhead, the bottom-left corner in Fig. 5.11 demarcates the desirable operating region. The two horizontal lines show boundaries for area overheads of 2× and 1.33×, respectively. For example, if the area overhead has to remain below 1.33×, we cannot operate at V_{MIN} – only V_3 becomes feasible for both of the variation profiles. On the other hand, if an area overhead of up to 2× is acceptable, we can lower the operating voltage to V_2 under *high* variation (Fig. 5.11(a)), and to V_1 under *low* variation (Fig. 5.11(b)). Even under *low* variation,

[1] i.e., by keeping the problem size constant

[2] $V_{NOM} = 0.85V$, $V_3 = 0.75V$, $V_2 = 0.65V$, $V_1 = 0.55V$, and $V_{MIN} = 0.44V$ in Fig. 5.11(a).

operating at V_{MIN} incurs an area overhead of more than 3×. Similarly, we can extract the maximum possible power savings at a given area budget from Fig. 5.11.

5.7 Related Work

Most of the proposed variation models for FinFET do not reach the architecture level [143, 141, 142]. Existing architecture-level variation models such as VARIUS(NTV) [147, 148], on the other hand, are tailored for planar MOSFET only. That said, VARIUS-TC represents an extension of VARIUS(NTV) to FinFET. A recent FinFET-based model, FinCANON [144] reaches the architecture level, but focuses on the network on chip and memory, rather than the processor logic. At the same time, FinCANON does not feature any probabilistic model to analyze major variation-triggered error modes, as opposed to VARIUS-TC. Another model that reaches the architecture-level is McPAT-PVT [154]. McPAT-PVT, as well, fails short of providing statistical reliability analysis under variation, as opposed to VARIUS-TC. On the other hand, both FinCANON and McPAT-PVT feature modular macro-models derived from TCAD-based device-level simulations, similar to VARIUS-TC's LUT.

Chapter 6

On Approximate Speculative Lock Elision

6.1 Introduction

A simple type of synchronization, mutual exclusion, is crucial for the correct execution of parallel programs. Mutual exclusion restricts accesses (which involve updates) to shared data objects to one parallel task at a time. Lock-based synchronization serves this purpose. Before updating shared data objects, each parallel task has to first *lock* the critical section, where the respective data objects reside, to prevent simultaneous update attempts from other tasks. A typical parallel program may incorporate multiple critical sections protected by locks, which can be interleaved with each other in different ways. Independent of the frequency of occurrence or interleaving, each lock imposes a total or partial order on the execution of parallel tasks. Ergo, each lock represents a point of serialization, and thereby can easily hurt the scalability of parallel programs.

To enhance parallel scalability in the face of inevitable mutual exclusion, recent studies [155, 156, 157, 158] proposed to approximate mutual exclusion by eliminating a subset of locks spatio-temporally. The idea is to exploit the inherent noise tolerance of the emerging R(ecognition), M(ining), and S(ynthesis) applications [159] in mitigating approximation incurred violations of basic parallel execution semantics. RMS algorithms are iterative and often probabilistic to process massive yet noisy data. The solution space usually features a range of valid outputs [160]. Therefore, RMS applications can tolerate

inaccuracies emanating from the data flow, as opposed to the control [161, 162, 163, 164]. In other words, RMS applications can mask approximation incurred semantic violations only if the divergence from fully-synchronized execution manifests as inaccuracies in the data flow. Even if approximation does not result in catastrophic program termination, for approximate mutual exclusion to be viable, the accuracy loss must remain bounded.

State-of-the-art optimizations for mutual exclusion focus on elimination of redundant, i.e., unnecessary, synchronization events – thereby, serialization points – without compromising computation accuracy [165, 166, 167, 168, 169]. Approximate mutual exclusion can complement these techniques by eliminating more synchronization points (in addition to the redundant) as long as the approximation incurred loss in computation accuracy remains at acceptable levels.

Approximation can apply to classic mutual exclusion implemented by locks, and as a more scalable alternative, to speculative (lock-based) synchronization such as *Transactional Memory (TM)* or *Speculative Lock Elision (SLE)* [165, 170, 171, 167, 166]. In this study, we assess the viability of approximate mutual exclusion by approximating SLE, which was adopted by hardware transactional memory (HTM) implementations from industry [172]. In the following, we will refer to this HTM based baseline for comparison as *SLE* in short. Under SLE, parallel tasks do not attempt to lock critical sections before updates to shared data. Instead, they directly enter the critical section by speculating that no other task would attempt a simultaneous update. While correct speculation can eliminate serialization due to lock-based synchronization, misspeculation can result in conflicting accesses to shared data, which in turn can corrupt data values. SLE has to carefully track potential misspeculation to be able orchestrate recovery upon detection of misspeculation. To this end, all updates by speculative accesses should be buffered, and only reflected to the architectural state (i.e., committed) once speculation is deemed correct.

Approximate Speculative Lock Elision (ASLE), the focus of this study, translates into spatio-temporal omission of conflict detection and/or recovery upon misspeculation – only if potential loss in computation accuracy, as induced by potential data corruption due to conflicting accesses, remains acceptable. There is no need to buffer speculative data in this case as there is no need to recover. As a result, ASLE can cut-off the overhead incurred by conflict detection, speculative storage, and recovery (upon misspeculation).

In the following, by exploring the trade-off space of computation accuracy versus approximation-enabled speed-up, we assess the feasibility of Approximate spatio-temporal Speculative Lock Elision, ASLE. In accordance with recent work [155, 157, 158, 173], we observe that, frequently enough, approximation induced data races manifest as corruption in data flow, hence, as degradation in computation accuracy. Our main contribution lies in the feasibility analysis of approximation. Specifically:

- We investigate the efficacy of exploiting semantic and temporal characteristics of critical sections (i.e., transactions in the context of TM) in controlling the degree of approximation, i.e., in preventing excessive loss in computation accuracy.
- We devise a light-weight Approximate Speculative Lock Elision (ASLE) implementation, which exploits existing hardware support for HTM.
- We quantitatively compare and contrast ASLE with SLE, which does not compromise computation accuracy as opposed to ASLE.
- We provide practical guidelines for ASLE, based on our findings.

In the rest of this study, Section 6.2 covers the motivation and background; Section 6.3, practical knobs and policies to control and to bound the accuracy loss under ASLE along with practical limitations; Sections 6.4 and 6.5, the evaluation of the viability of ASLE; and Section 6.6, related work.

6.2 Background & Motivation

6.2.1 Synchronization as a Barrier to Parallel Scalability

For a representative set of RMS applications, Figure 6.1 captures how the time overhead of synchronization evolves as the number of threads (as a measure of the degree of parallelism) increases[1]. The y-axis depicts, for each thread count, the percentage of the total execution time spent in synchronization events. Per Amdahl's Law, time spent in synchronization represents a loose upper bound for approximation-enabled speed-up.

[1] We deploy the largest available input data set for each benchmark. Section 6.4 provides the experimental setup.

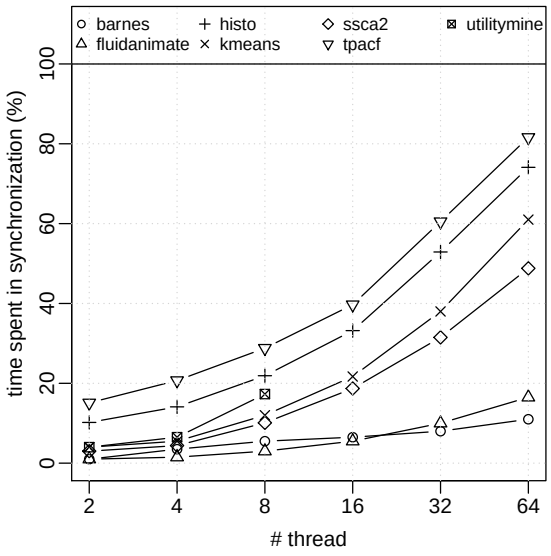


Figure 6.1: % time overhead of synchronization.

We observe that the synchronization overhead increases with higher degrees of parallelism. This is because, under a fixed problem size, per thread work reduces as the thread count increases. Accordingly, all threads, including the slowest, finish earlier, and the overall execution time reduces. At the same time, with increasing thread count, the number of sharers for a given chunk of data tends to grow, giving rise to more frequent synchronization. For example, as the thread count increases from 4 to 64, the time spent in synchronization increases from 4.4% to 48.8%; from 0.2% to 16.5%; and from 10.2% to 74.1% for *ssca2*; *fluidanimate*; and *histo*; respectively. This observation holds under an increasing problem size (aka weak scaling), as well. In this case, the work per thread remains constant where the thread count increases, which oftentimes results in more sharers, therefore, more frequent synchronization [174].

6.2.2 Approximate Mutual Exclusion: Challenges

Under approximate mutual exclusion, multiple independent accesses to modify shared data can proceed simultaneously. Consequently, approximation may not only corrupt data values residing in critical sections, but also prevent program termination, by

e.g., giving rise to deadlocks. In mitigating approximation induced violations of basic execution semantics, the inherent noise tolerance of RMS applications can only help if the violations do not escape to control-flow, and do not result in excessive degradation in computation accuracy by corrupting data-flow [161, 162, 163, 164].

Under approximate mutual exclusion, even in the absence of catastrophic program termination, the magnitude of data corruption becomes hard to predict. Worse, approximation induced data value corruption can propagate to application outputs in numerous ways [175, 176]. The magnitude of corruption at the outputs depends on the sensitivity of program outputs to the corrupted data values (residing in critical sections subject to approximate mutual exclusion). During execution, this sensitivity may change temporally, as well. Most iterative RMS algorithms progressively refine their outputs each iteration, until meeting predefined convergence criteria [177]. Accordingly, iterations further in the execution may tolerate approximate mutual exclusion less than iterations at the beginning. Therefore, approximation may also affect how fast the algorithm converges to a solution.

6.2.3 Case Study: kmeans

A heavily used RMS application for data mining, *kmeans*, relies on iterative refinement (in its most standard form) in partitioning its input data points into k clusters. Every iteration enforces the assignment of each input point to the closest possible cluster. The algorithm terminates once assignments stabilize. *kmeans* from the STAMP benchmark suite [178] incorporates three critical sections:

The first protects the assignment of new points to clusters:

```
1 *new_centers_len[index]
        = *new_centers_len[index] + 1;
2 <Insert new point to list>
```

`new_centers_len[index]` keeps the number of points assigned to cluster `index`. Line 2 inserts the new point to the list containing all points assigned to cluster `index`. Under approximation, multiple threads may try to assign a point to cluster `index` simultaneously. This may corrupt the update to `new_centers_len[index]` in line 1, or the corresponding insertion of the point to the cluster's list in line 2. A corruption in line 1 can easily

render a lower (than actual) number of points per cluster. A corruption in line 2, on the other hand, may skew cluster centers. *kmeans* can mask both types of corruption due to the iterative refinement of cluster centers until convergence.

The next critical section controls the distribution of input data points among threads:

```
start = global_i;
global_i = start + CHUNK;
```

In this critical section, each thread tries to get `CHUNK` number of points for processing. The points reside in an array with `global_i` pointing to the index of the last point already assigned to a(nother) thread. Accordingly, the next thread in the row increments `global_i` by `CHUNK`. This critical section mainly affects control flow. Under approximation, multiple threads may end up processing the very same points. Such redundant processing may easily increase the overall execution time particularly if the thread count is no greater than the total number of chunks (i.e., number of sets of points of size `CHUNK`). At the same time, the accuracy may degrade as adding the same point to a cluster multiple times may skew cluster centers.

The final critical section protects convergence control:

```
global_delta = global_delta + delta;
```

`global_delta` captures the total number of new cluster assignments overall; `delta`, in a given iteration. The algorithm terminates if `global_delta` falls below a predefined threshold. This critical section mainly affects control flow. Under approximation, multiple threads may try to update `global_delta` simultaneously. As a result, `global_delta` can possibly assume a lower value than under fully-synchronized execution, and trigger premature termination. Premature termination is likely to boost performance due to faster, yet false, convergence. The corresponding accuracy loss, however, may become excessive. Adjusting the threshold or imposing a fixed number of iterations may work better than approximate mutual exclusion in this case, as we will demonstrate in Section 6.5.

6.3 Approximate Speculative Lock Elision

We will next analyze the viability of approximate synchronization using Speculative Lock Elision (SLE), which was adopted by hardware transactional memory implementations

from industry [172], as a baseline for comparison. In the following, we will refer to this HTM based baseline for comparison as *SLE* in short. The key difference of Approximate Speculative Lock Elision (ASLE) from SLE is spatio-temporal omission of conflict detection and/or recovery upon misspeculation – however, only if potential loss in computation accuracy, as induced by potential data corruption due to conflicting accesses, remains acceptable.

A key design question for ASLE hence becomes *how to decide where in program and when to turn off conflict detection and/or recovery upon misspeculation* – which is the equivalent of *how to select the locks to elide* in approximating classic mutual exclusion. The intuitive answer to this question is *when the execution reaches an inaccuracy-tolerant critical section.* Recall that the tolerance of the target application domain, RMS, to inaccuracy in computation comes from (i) mostly probabilistic algorithms often using iterative refinement; and (ii) inputs containing a very large number of inaccurate, often redundant data elements. Due to (i), it is barely possible to differentiate inaccuracy-tolerant critical sections from others without analyzing program semantics. Unfortunately, (ii) complicates the identification of inaccuracy-tolerant critical sections further, as the inaccuracy-tolerance tends to change as a function of inputs. Therefore, profiling-based identification of inaccuracy-tolerant critical sections becomes inevitable. These constraints do complicate programming. The proof-of-concept ASLE implementation covered in this study relies on profiling-based identification of inaccuracy-tolerant critical sections, similar to [179, 176, 156].

After identifying inaccuracy-tolerant critical sections, the next design question becomes *how to turn off conflict detection and/or recovery upon misspeculation* within this subset of critical sections – which is the equivalent of *how to elide the locks* in approximating classic mutual exclusion. A critical section may lend itself well to approximation, however, the corresponding accuracy loss may fluctuate at runtime due to variations in the degree of parallelism, size and characteristics of input data, or environmental conditions. Therefore, enforcing approximation at compile time may not always be safe. Instead, by communicating *potentially* inaccuracy-tolerant critical sections subject to approximation to the hardware, we can devise adaptive policies to control the degree of approximation at runtime. To this end, we can adapt programming language or

instruction set extensions as suggested by [180, 181, 182, 157]. To demarcate potentially inaccuracy-tolerant critical sections subject to approximation, similar to AMD's ASF [183] or Intel's TSX [172] for SLE, the proof-of-concept ASLE implementation relies on compiler-managed instruction set extensions `APPROX_ACQUIRE` and `APPROX_RELEASE`.

We will next discuss the remaining open questions of *how to control the degree of approximation* and *how to prevent excessive accuracy loss under approximation.*

6.3.1 Knobs to Control the Degree of Approximation

ASLE can degrade computation accuracy only if multiple parallel tasks attempt to enter a critical section simultaneously, i.e., if accesses to shared data conflict. The magnitude of accuracy loss is likely to grow with an increasing number of conflicting accesses. In other words, approximation in higher contention critical sections is likely to render higher loss in computation accuracy as observed by previous work [158]. Based on this insight, we will next investigate how contention profiles can serve as a knob to adjust the accuracy loss due to approximation. In the following, we will refer to *the number of conflicting accesses* as a proxy for *contention.* On the other hand, the *share of conflicting accesses over all accesses* constitutes the *conflict rate.*

To quantify how strongly correlated contention and accuracy loss are, we experiment[2] with a very aggressive type of approximate (speculative) mutual exclusion, where we never check for conflicts – and hence, (can) never trigger recovery. In the rest of this study, we will refer to this as *Brute-force Lock Elision* (BLE). BLE is more likely to result in conflicts than ASLE. Under classic mutual exclusion, BLE corresponds to permanent spatio-temporal elision of the lock for the duration of the entire program. For this analysis, we only consider inaccuracy-tolerant critical sections (i.e., critical sections lending themselves well to approximation), as identified by profiling. For the representative RMS benchmarks deployed in this study we identify three distinctive cases:

i. Persistently low contention critical sections where the conflict rate assumes a very low value throughout execution, which does not change with contention.

[2]We deploy the largest available input data sets and 64 threads. Section 6.4 details the experimental setup.

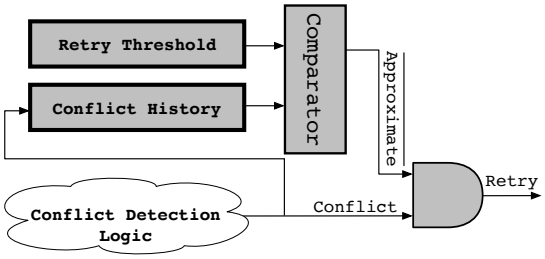


Figure 6.2: A light-weight approximation extension (depicted in gray) to speculative lock elision (SLE).

ii. Relatively low contention critical sections where accuracy loss is strongly correlated (i.e., changes almost linearly) with conflict rate. Generally, conflict rate increases with contention, and a higher conflict rate renders higher accuracy loss.

ii. High contention critical sections where accuracy loss is strongly correlated (i.e., changes almost linearly) with conflict rate. Generally, conflict rate increases with contention, and a higher conflict rate renders higher accuracy loss.

We conclude that, modulo **i.**, *contention* or *conflict rate* can serve as a practical knob to control the accuracy loss under approximation by ASLE. For **i.** BLE is a better option than ASLE, as we will demonstrate in Section 6.5.

6.3.2 A Proof-of-Concept ASLE Implementation

Speculative Lock Elision (SLE) has to carefully track potential conflicts to be able orchestrate recovery upon conflict detection. The baseline SLE implementation already features `Conflict Detection Logic` which we can exploit to keep track of *contention* in controlling the degree of approximation. Figure 6.2 details a light-weight approximation extension (shaded in gray) to SLE in hardware. Under SLE (which does not feature any of the shaded approximation extensions) `Conflict Detection Logic` asserts the `Conflict` signal upon detection of conflicting accesses. Assertion of `Conflict` triggers the `Retry` signal in order to have conflicting accesses to shared data re-attempted.

Approximate SLE, ASLE, on the other hand, can opportunistically omit these `Retry`

attempts, as long as degradation in computation accuracy due to conflicts remains at acceptable levels. The proof-of-concept ASLE implementation uses the very same `Conflict Detection Logic` as SLE to detect conflicts. However, under approximation, assertion of `Conflict` triggers the `Retry` signal only if the conflicting accesses can lead to excessive loss in computation accuracy. To control the degree of accuracy loss, i.e., approximation, ASLE relies on two buffers: `Retry Threshold` and `Conflict History`. `Retry Threshold` and `Conflict History` represent special purpose programmable registers per core. `Retry Threshold` controls the frequency of approximation, while `Conflict History` keeps track of contention.

`Conflict History` is an N-bit shift register, and stores the N most recent values of the `Conflict` signal. Under ASLE, each thread attempts to to update the architectural state once done with the execution of a critical section subject to approximation. At this stage, if there was no conflict, i.e., the thread finds the `Conflict` signal not set, no retry is necessary (i.e., `Retry` signal is not set). Otherwise, if there was a conflict, i.e., the thread finds the `Conflict` signal set, `Retry` is only triggered if we disable approximation by resetting the `Approximate` signal from Figure 6.2. We next look into how the proof-of-concept ASLE implementation manages the `Approximate` signal.

6.3.3 Runtime Policies to Control Accuracy Loss

We set `Approximate` from Figure 6.2 as a function *func* of how the number of conflicts (encountered during the N most recent commit attempts, on a per core basis) compares to a given threshold, which is stored in `Retry Threshold`. The number of conflicts encountered during the N most recent commit attempts corresponds to the number of 1's in `Conflict History`. We next devise several runtime policies (each featuring a different *func*) which span the trade-off space of accuracy loss vs. speed-up under approximation.

Under the first policy, we trigger `Approximate` if the number of conflicts (encountered during the N most recent commit attempts, on a per core basis) is **Higher-Than** `Retry Threshold`. At the end of execution of each critical section subject to approximation,

- if `Conflict` = 0: ASLE falls back to SLE (**Case-1**).
- if `Conflict` = 1 and `Conflict History` keeps a conflict count lower than `Retry`

`Threshold`: ASLE falls back to SLE (**Case-2**).

- if `Conflict` = 1 and `Conflict History` keeps a conflict count higher than `Retry Threshold`: by setting `Approximate` (which in turn resets `Retry`), we let conflicting tasks proceed without attempting retry (**Case-3**).

Higher-Than thereby enables approximation only if contention, i.e., the number of conflicting accesses, exceeds a preset threshold. Therefore, **Higher-Than** *is biased to approximate higher contention critical sections, which render a higher number of conflicts. Approximating predominantly higher contention critical sections may deliver higher speed-up (as higher contention implies more retries which ASLE can effectively cut-off), however, the accompanied loss in accuracy may become hard to bound.* Under **Higher-Than**, a higher value of `Retry Threshold` leads to less frequent approximation, and therefore, (more likely) to less accuracy loss.

We define the dual policy of **Higher-Than** as **Lower-Than**, by inverting the sense of the threshold logic for **Case-2** and **Case-3**. Under **Lower-Than**,

- if `Conflict` = 0: ASLE falls back to SLE.
- if `Conflict` = 1 and `Conflict History` keeps a conflict count higher than `Retry Threshold`: ASLE falls back to SLE.
- if `Conflict` = 1 and `Conflict History` keeps a conflict count lower than `Retry Threshold`: by setting `Approximate` (which in turn resets `Retry`), we let conflicting tasks proceed without attempting retry.

Contrary to **Higher-Than**, **Lower-Than** *is biased to approximate lower contention critical sections, which render a lower number of conflicts. Approximating predominantly lower contention critical sections may deliver modest speed-up, however, the accompanied loss in accuracy is likely to be modest, as well.* Under **Lower-Than**, a higher value of `Retry Threshold` leads to more frequent approximation, and therefore, (more likely) to more accuracy loss.

Putting It All Together: Under **Higher-Than**, ASLE reacts to high contention, and triggers approximation mainly to boost performance. Under **Lower-Than**, ASLE reacts to low contention, and triggers approximation mainly to keep the accuracy loss bounded,

which is accompanied by a more modest speed-up than **Higher-Than**. Under both policies, `Retry Threshold` controls the frequency of approximation, therefore, the loss in accuracy. Fine-tuning the value of `Retry Threshold`, possibly dynamically at runtime, can help prevent excessive loss in computation accuracy.

An application can feature many critical sections, each exhibiting different contention characteristics. Figure 6.2 shows basic hardware support to keep track of a single critical section. However, if critical sections of an application are sufficiently apart from each other in time, this hardware can also accommodate accurate support for multiple critical sections. Otherwise, we can introduce an array of `Conflict History` along with an array of `Retry Threshold`, each array element to keep track of a separate critical section. RMS applications we experiment with do not feature more than a few (static) inaccuracy-tolerant critical sections, as Section 6.5 reveals.

6.3.4 Practical Limitations

Demand for application specific characterization: Not every critical section lends itself well to approximation as demonstrated in Section 6.2.3. Oftentimes, selection of locks subject to elision demands significant semantic knowledge about the application. Profiling may help, but cannot cover all possible use cases. For ASLE policies , we relied on profiling, first, to identify critical sections which (when relaxed) do not lead to catastrophic termination, in the form of non-termination or excessive degradation in output accuracy. We excluded these critical sections from approximation. For the rest, we identified, through a more detailed profiling step, higher contention critical sections and adjusted the ASLE thresholds accordingly, based on the average contention rates.

Confining approximation-induced inaccuracies in data-flow: Exploiting contention profiles, ASLE policies can only *qualitatively* control the accuracy loss. While ASLE policies (particularly, the **Lower-Than** variants) can probabilistically prevent excessive accuracy loss, they fail short of bounding its magnitude. To be able to bound the magnitude of accuracy loss, we need to *quantitatively* characterize how approximation-induced inaccuracies in the data flow propagate to the application output to degrade the output accuracy. An important step in addressing this fundamental limitation is adapting previous work such as [164, 184] to enforce approximation incurred inaccuracies confined in data flow where RMS applications can tolerate corruption, as opposed to

Benchmark	Description	Input files	Accuracy metric
barnes (splash2x)	n-body Simulation	simsmall of splash2x simmedium of splash2x simlarge of splash2x	avg. relative deviation of coordinates
fluidanimate (RMS-TM 3)	n-body Simulation	simsmall of parsec 2.0 simmedium of parsec 2.0 simlarge of parsec 2.0	avg. relative deviation of coordinates

histo (Parboil 2.5)	Saturating Histogram	`-i img.bin`
		`-i uniform.in`
		`-i guassion.in`
			avg. relative deviation of bin values

kmeans (STAMP)	Data Mining	`-m15 -n15 -t0.001 -i random-n2K`
		`-m15 -n15 -t0.0001 -i random-n16K`
		`-m15 -n15 -t0.00001 -i random-n65K`
			ratio of wrong clusterings

ssca2 (STAMP)	Graph Analysis	`-s13 -i1.0 -u1.0 -l3 -p3`
		`-s16 -i1.0 -u1.0 -l3 -p3`
		`-s18 -i1.0 -u1.0 -l3 -p3`
			ratio of differences in the number of edges

tpacf (Parboil 2.5)	Two Point Angular Correlation Function	`-n 100 -p 487`
		`-n 100 -p 4096`
		`-n 100 -p 10391`
			avg. relative deviation of bin values

utilitymine (RMS-TM 3)	Association Rule Mining	`data...nitems_10.patlen_6 0.01` `data...nitems_1.patlen_6 0.01`	avg. relative deviation of utilities

Table 6.1: RMS benchmarks deployed.

control flow. Decoupling data flow from control is already a challenging task beyond the scope of this study.

Need for safety-nets: Even if we effectively managed to confine approximation-induced inaccuracies to data flow of RMS applications, there is no deterministic guarantee that data flow errors do not result in catastrophic program termination. Accordingly, for ASLE to be viable, we still need to devise safety-nets (e.g., based on checkpoint-recovery) to facilitate recovery without compromising design complexity. A more viable solution is enabling approximation only under very strong statistical guarantees.

6.4 Evaluation Setup

Benchmarks: As captured by Table 6.1, we deploy a representative set of RMS applications from PARSEC [185], Parboil [186], STAMP [178], SPLASH2x [187], and RMS-TM [188] suites. *barnes* and *fluidanimate* represent n-body simulations. *histo* computes a large, 2-D saturating histogram. *kmeans* implements an iterative clustering

Benchmark	File(line)	Length	Frequency	Cont.	Protection
barnes	code.C(721)	Long	Medium	Low	global min and max
fluidanimate	pthreads.cpp(635)	Short	High	Low	cell density
	pthreads.cpp(641)	Short	High	Low	cell density
	pthreads.cpp(744)	Short	High	Low	cell acceleration
	pthreads.cpp(761)	Short	High	Low	cell acceleration
histo	main.c(101)	Short	High	Low	bins update
kmeans	normal.c(168)	Long	High	High	center update
ssca2	c...Graph.cpp(475)	Long	High	High	add edge to list
tpacf	model_com...cpu.c(52)	Short	High	High	bins update
utilitymine	utility.cpp(281)	Short	Medium	Low	update utility

Table 6.2: Critical sections subject to approximation.

# of cores	65
Core	4-issue wide OoO
Private L1D	32KB 8-way, 64B LRU, 1-cycle hit
Private L1I	32KB 8-way, 64B LRU, 1-cycle hit
Private L2	512KB 8-way, 64B LRU, 11-cycle hit
Shared L3	130MB 16-way, 64B LRU, 30-cycle hit
Main memory	90ns round-trip
Technology	22nm
Voltage/ frequency	1.0V/1.053GHz

Table 6.3: Architectural parameters.

algorithm. *ssca2* consists of four graph kernels; we use the first kernel which constructs an efficient graph data structure. *tpacf* generates a histogram capturing the spatial distribution of astronomical observations. *utilitymine*, an Associate Rule Mining (ARM) application, extracts high-utility itemsets from a database.

Approximation Targets: We confine approximation to critical sections (CS) protected by locks or transactions. The applications feature critical sections of different length, (dynamic execution) frequency, and contention characteristics, as depicted in Table 6.2. We focus on critical sections exercised frequently enough to qualify as potential performance bottlenecks. Further, if approximation of a critical section is likely to result in catastrophic program termination, we exclude it from consideration.

Metrics to Quantify Accuracy Loss: To quantify the accuracy loss due to approximation, we adapt accuracy metrics from Misailovic et al. [189] along with Akturk et al. [190]. The last column of Table 6.1 depicts the accuracy metrics.

Simulation Infrastructure: We deploy Sniper [59] for microarchitectural simulation, as configured in Table 6.3. We implement all policies from Section 6.3 in Sniper. The baseline SLE implementation represents a TSX-like design [172], which we evaluate under both eager and lazy conflict detection. Not to compromise simulation speed, Sniper tends to model the interactions among threads at a coarse-granularity, which may lead to implicit serialization. A similar restriction applies to many microarchitectural simulators. To mitigate this effect, we increase the simulator's thread synchronization

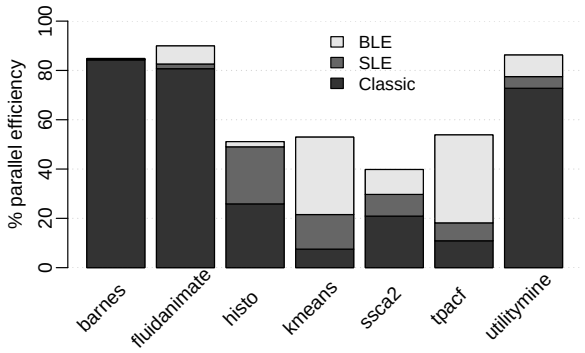


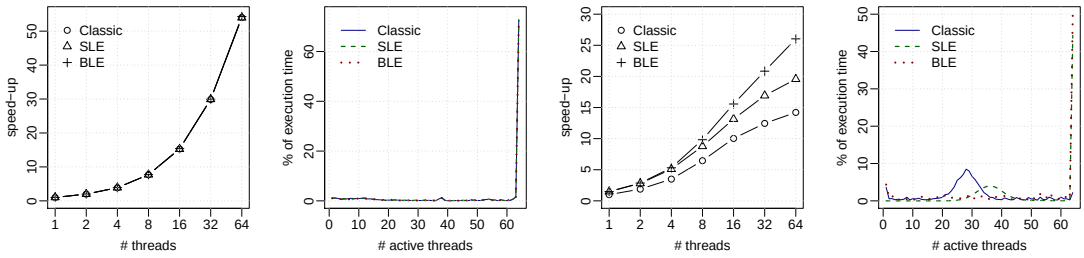
Figure 6.3: Parallel efficiency under lock elision.

frequency to its maximum value. To establish statistical significance of our results, we repeat each experiment 100 times, and report the mean. To capture the sensitivity of the execution outcome to inputs, we use a range of input data-sets for each application, as tabulated in the third column of Table 6.1. Our simulations involve up to 64 threads. For Parboil applications we implemented the pthread versions, and made sure that the pthread implementation outperformed the omp-based baseline not to favor any ASLE policy.

6.5 Evaluation

6.5.1 Impact on Execution Time

We first compare and contrast the execution time under classic locking (*Classic*) with speculative lock elision (*SLE*) (implemented in hardware), and with persistent, spatio-temporal brute-force lock elision (*BLE*), for each application. We deploy the largest input data set for each benchmark. For each lock implementation, we report *parallel efficiency* as *the ratio of the speed-up (over the single-threaded execution) to the maximum possible speed-up for a given thread count.* For example, if an 64 threaded run of a benchmark delivers 50× speed-up over single-threaded execution, parallel efficiency becomes $50/64 \approx 78\%$. In Figure 6.3, the y-axis provides the % parallel efficiency, where each bar corresponds to a benchmark, and each stack depicts the % improvement in parallel efficiency under a particular lock implementation.

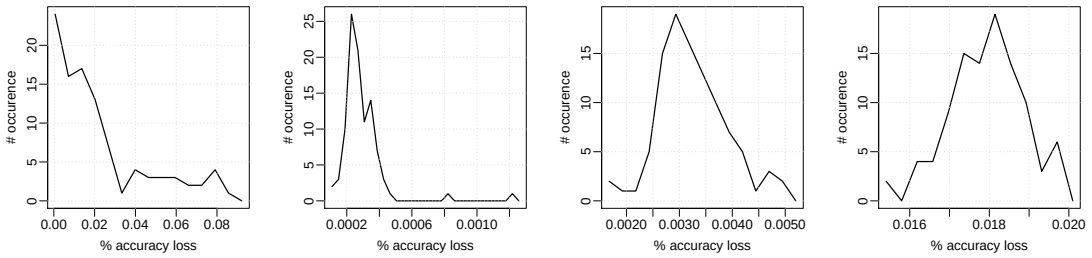


(a) speed-up (b) concurrency profile (c) speed-up (d) concurrency profile

Figure 6.4: Speed-up and concurrency profile for *barnes* (a,b) and *ssca2* (c,d) .

We observe that lock elision (be it *BLE* or *SLE*) does not improve the efficiency of barnes. For *fluidanimate*, *BLE* improves the efficiency by 8.8% over *SLE*, which itself delivers 2.3% more efficiency over *Classic*. These efficiency numbers evolve to 49.0% and 51.1% for *histo*; to 29.7% and 39.81% for *ssca2*; and to 18.2% and 53.8% for *tpacf*. Finally, *utilitymine* experiences 77.5% efficiency under *SLE*, and an additional 11.3%, under *BLE* for 8-threads (maximum number of threads for this application). For *kmeans*, we restrict the comparison to a single iteration of the application (as explained in Section 6.5.4). In this case, *SLE* delivers 21.5%; *BLE*, 52.9% efficiency.

Figures 6.4a and 6.4b provide the corresponding speed-up and concurrency profiles for *barnes*. The speed-up is normalized to the execution time of the single-threaded run under *Classic*, and reported as a function of the thread count on the x-axis. From Figure 6.4a, we observe that lock elision does not deliver any speed-up for this application. The concurrency profile from Figure 6.4b – which captures the % of execution time (as depicted on the y-axis) the application features a specific number of concurrently active threads (as depicted on the x-axis) – verifies this trend. Figures 6.4c and 6.4d provide the same analysis for *ssca2*, which, as opposed to *barnes*, demonstrates enhanced concurrency under lock elision.

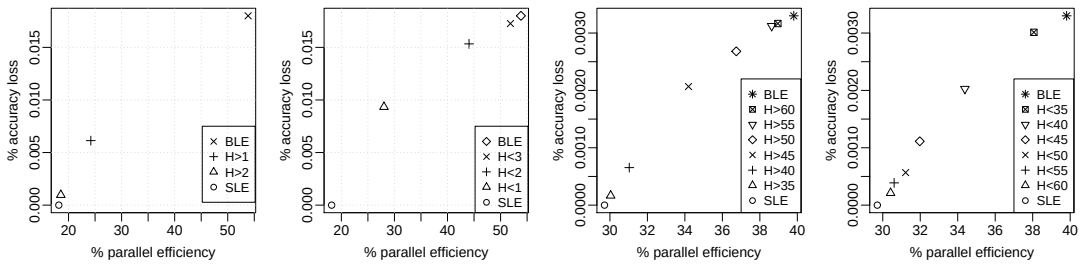


(a) *fluidanimate* (b) *histo* (c) *ssca2* (d) *tpacf*

Figure 6.5: Accuracy loss under brute-force lock elision, *BLE*, to accompany the efficiency profiles from Figure 6.3.

6.5.2 Impact on Accuracy Loss

Figure 6.5 captures how *BLE* changes the computation accuracy to accompany the (potential) efficiency improvement from Figure 6.3. *Classic* and *SLE* do not compromise the computation accuracy. The analysis reflects 64-threaded runs for the largest input set. The figures report the distribution of % accuracy loss over 100 runs. No accuracy loss applies to *barnes* , since the execution trajectory under *BLE* closely follows the execution trajectory under *Classic* or *SLE*. *fluidanimate* (Figure 6.5a) shows a modest accuracy loss – less than 0.092% – due to the very low contention of its locks. The accuracy loss remains negligible for *histo*, as well (Figure 6.5b). This is because *histo* features a very large number of bins (in the order of hundreds), which decreases the probability of the same bin being accessed by multiple threads, even under high contention. We will provide a detailed accuracy analysis for *kmeans* in Section 6.5.4. *ssca2* exhibits an accuracy loss of 0.0031%, on average. *tpacf* also has a histogram as its output, but suffers from a significantly higher accuracy loss than *histo* due to the lower number bins (in the order of 20) (Figure 6.5d). The % accuracy loss distribution under *utilitymine* assumes a similar pattern to *tpacf* over a range of $[0.009 - 0.011]\%$ – although the utilities in the output of *utilitymine* degrade slightly, the final high-utility itemsets very closely follow the exact outcome, i.e., the outcome under *Classic* or *SLE*. Overall, the modest % accuracy loss observed across all applications renders the approximation-enabled parallel



(a) **Higher-Than** (b) **Lower-Than** (c) **Higher-Than** (d) **Lower-Than**

Figure 6.6: Trade-off space for *tpacf* (a,b) and *ssca2* (c,d) .

efficiency enhancement under *BLE* viable (Figure 6.3).

6.5.3 Controlling Accuracy Loss

We next characterize how the trade-off space for % accuracy loss vs. % (parallel) efficiency (Section 6.5.1) evolves under policies from Section 6.3.3. We report % accuracy loss on the y-axis and % efficiency on the x-axis. All simulations reflect 64-threaded runs, using the largest input size. In the following, we only consider applications with non-negligible contention. Figure 6.6 shows the trade-off space under **Higher-Than** and **Lower-Than** policies, considering different values of `Retry Threshold` for *tpacf*. In this case, `Conflict History` keeps a very small number (of ones) throughout the execution. We adjust the range for `Retry Threshold` accordingly. For instance, **Higher-Than** 1 policy enforces approximation only if the one count of `Conflict History` is greater than 1, to render an ≈2.9× more accurate result than under *BLE*, where efficiency falls by ≈30% beyond *BLE* ($H > 1$ from Figure 6.6a). **Lower-Than** 1 policy, on the other hand, renders an ≈1.9× more accurate result than under *BLE*, where efficiency falls by more than 20% beyond *BLE* ($H < 1$ in Figure 6.6b).

Following our observations from Section 6.3.1, we experiment with larger values of `Retry Threshold` for the higher contention application *ssca2*. Figure 6.6 depicts the trade-off space. In this case, **Higher-Than** 45 ($H > 45$) policy improves the accuracy by 37.3% over *BLE*, accompanied by a more than 5% reduction in efficiency (Figure 6.6c). **Lower-Than**, $H < 45$ policy, on the other hand, improves the accuracy by 66.3% over

BLE, accompanied by around 8% reduction in efficiency (Figure 6.6d).

Overall, we observe that this basic set of policies can span a rich trade-off space, which we can exploit to deliver the optimal accuracy loss vs. parallel scalability under varying application-specific constraints. The programmer or the runtime can choose between these two policies depending on the relative importance of speed-up vs. accuracy.

6.5.4 Case Study: kmeans

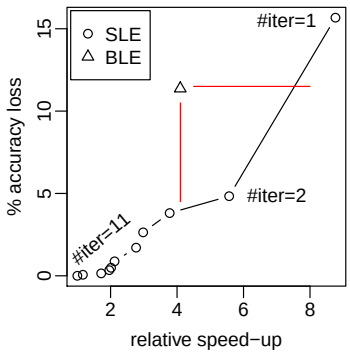


Figure 6.7: Trade-off space for *kmeans*.

Since *kmeans* relies on iterative refinement, by enforcing a lower number of iterations until convergence than the fully-synchronized baseline execution, we can trade accuracy for speed-up, without approximating synchronization. To quantify this effect, we turned off the convergence check of the algorithm, and manually enforced a fixed number of iterations (*#iter*) for each run, under both SLE and BLE, for 64 threads. Figure 6.7 summarizes our findings: Each point corresponds to a fixed iteration count, *#iter* (which increases as we move from top-right to bottom-left). The x-coordinate of each point captures the speed-up under *#iter*; the y-coordinate, the corresponding % loss in accuracy. The speed-up is reported with respect to the baseline *SLE* without any approximation under the default iteration count until termination (i.e., by enforcing the default convergence check). The speed-up reduces with increasing values of *#iter*, the number of iterations executed until termination. The triangle demarcates the outcome under *BLE*; the % accuracy loss on the y-axis, the speed-up on the x-axis. The red rectangle demarcates the region of "higher speed-up at lower accuracy loss" than possible

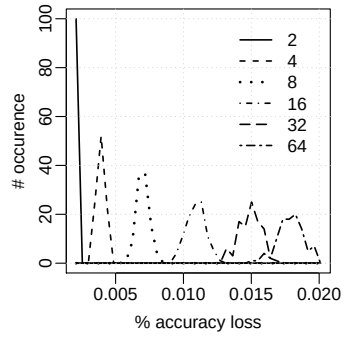


Figure 6.8: Sensitivity of % accuracy loss to thread count.

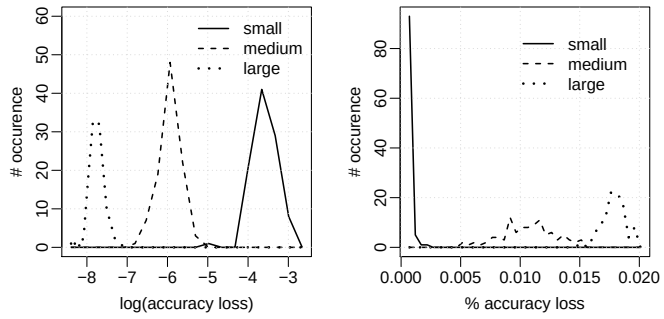
under *BLE*. We observe that *#iter*= 2 indeed renders a point in this more favorable region than *BLE*. Accordingly, enforcing a fixed number of iterations may result in a better accuracy versus speed-up trade-off than possible by enforcing *BLE*.

6.5.5 Sensitivity Analysis

Figure 6.8 depicts how the distribution of % loss in accuracy evolves with an increasing thread count for the largest input of *tpacf* under *BLE*. A higher number of threads leads to a higher number of conflicts, which progressively increases the % accuracy loss. A similar trend applies for all of the RMS benchmarks we experimented with.

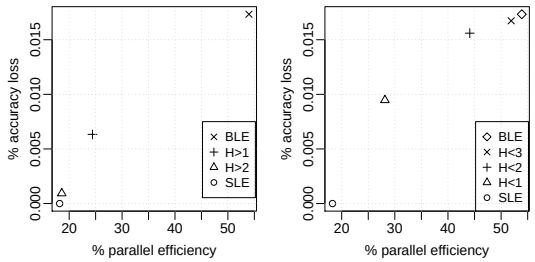
However, the picture changes across different benchmarks when we deploy different input sizes. We observe two distinct patterns, as captured in Figure 6.9 for two representative applications, *tpacf* (Figure 6.9b) and *ssca2* (Figure 6.9a), respectively. A larger input size renders a higher % accuracy loss for *ssca2*, while the opposite trend applies for *tpacf*.

So far, without loss of generality, for (A)SLE we assumed *Lazy* conflict detection; i.e., conflict detection being fired at the end of critical sections. *Eager* [191] conflict detection, on the other hand, would trigger the `Conflict` signal immediately upon identification of conflicts inside the critical section. The performance of applications under these two policies may vary. For instance, *histo* and *tpacf* show very similar parallel efficiency under *Eager* to *Lazy* (with less than 0.5% difference), since they both feature a very short critical section. For applications with longer critical sections, such as *ssca2*, the



(a) *ssca2* (b) *tpacf*

Figure 6.9: Sensitivity of % accuracy loss to input size.



(a) **Higher-Than** (b) **Lower-Than**

Figure 6.10: Trade-off space for *tpacf* under *Eager* conflict detection .

difference becomes more visible: for example, parallel efficiency of *ssca2* improves from 29.7% under *Lazy* to 33.2% under *Eager* conflict detection.

Figure 6.10 depicts how the trade-off space of accuracy loss versus parallel efficiency looks like under *Eager* conflict detection for *tpacf*. Under *Eager*, ASLE still delivers a rich trade-off space of accuracy vs. speed-up. A similar trade-off space applies for other applications, as well. Comparing *Eager* and *Lazy* conflict detection, only the range of parallel efficiency benefits varies.

6.6 Related Work

Effective techniques to mitigate the overhead of synchronization [168, 169, 192], including speculation [165, 166, 167, 170, 171], eliminate unnecessary synchronization events without compromising accuracy. By eliminating even more synchronization points, approximate synchronization can complement these techniques [182, 156, 157, 158]. Under speculative synchronization, parallel tasks (threads or transactions) proceed speculatively past synchronization points. If speculation does not result in conflicting accesses to shared data or violation of parallel execution semantics, this class of techniques can eliminate serialization due to synchronization, at the cost of extra storage for speculative state, control logic to detect violations of parallel execution semantics, and to orchestrate roll-back to a safe state and retry in case of misspeculation. Approximation can mitigate (if not eliminate) the overhead of storage, conflict detection, or recovery incurred by speculation. While previous studies on approximate synchronization focus mostly on programming language implications or basic quantitative characterization [182, 156, 157], we explore a transparent hardware extension to speculative lock elision in order to orchestrate approximation.

6.7 Discussion & Future Work

Under persistent high contention, due to a monotonic increase in conflict count (and independent of the value of `Retry Threshold`) once conflict count exceeds `Retry Threshold`, **Higher-Than** can get stuck at **Case-3** (Section 6.3.3), i.e., fall back to persistent, brute-force lock elision in space and time (BLE). Symmetrically, **Lower-Than** can get stuck at SLE. Brute-force lock elision is not always safe, and can easily lead to excessive accuracy loss. On the other hand, SLE may render a too conservative execution by blocking approximation opportunities. We can avoid this by tracking the rate of change (i.e., the gradient) of conflict count, and by re-interpreting `Retry Threshold` to correspond to a threshold for the gradient, rather than for an immediate value of conflict count. This insight would give rise to two new policies, **Gradient-Higher-Than** and **Gradient-Lower-Than**. In this case, `Conflict History` would log the history of most recent N values of the conflict count in a shift buffer. **Gradient** policies can then derive the rate of change from the difference between the most and least recent values of the conflict

count. Under **Gradient-Higher-Than** (**Gradient-Lower-Than**), we would elide the lock if the gradient exceeds (remains lower than) the value of `Retry Threshold`. In this manner, ASLE can better respond to fine grain changes in contention profiles. At the same time, **Gradient** policies would barely demand any profiling to determine the range for `Retry Threshold`, as all we need to determine would be monotonicity.

The evaluated benchmarks in this study (Section 6.5) do not show any pathology to necessitate **Gradient** policies and perform well under the lower complexity basic policies (**Higher-Than** and **Lower-Than**). We hence leave further exploration to future work.

In this study, we evaluated the basic idea using a hardware implementation because hardware transactional memory (HTM) is usually more efficient than software transactional memory, and extension of already existing hardware support for HTM in commercial systems to ASLE would be straight-forward, via addition of two registers and a comparator per core. That said, ASLE can also be implemented using Intel's TSX extensions in software. We believe that a hardware solution is less intrusive and faster, since for example, there is no memory access involved to update the conflict history which would be the case otherwise.

ASLE policies enable the user to adjust the level of approximation by tuning the policy thresholds. In this manner, ASLE can prevent excessive accuracy loss which is not the case for BLE. ASLE does not always guarantee better accuracy at the same performance level as BLE, but rather provides the user with the option of choosing a feasible point from the accuracy-performance trade-off space.

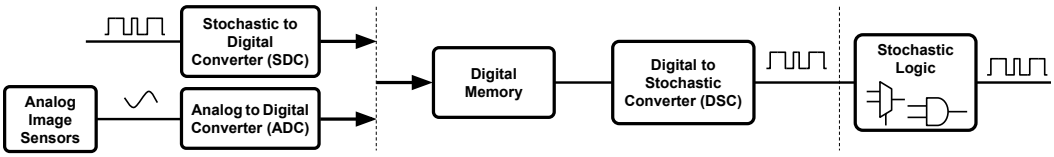
Chapter 7

On Memory System Design for Stochastic Computing

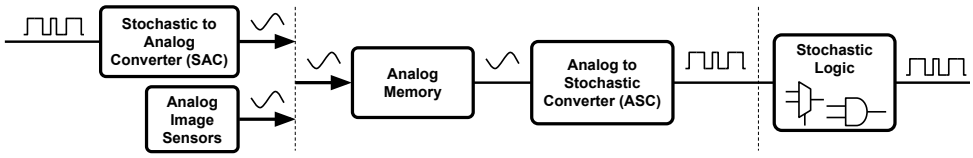
7.1 Motivation

Stochastic Computing (SC) has received renewed attention in recent [193, 19, 194, 195]. This is due to the growing uncertainty in design parameters, and therefore, in design functionality, as induced by imbalances in modern technology scaling. Representing and processing data as quantized probabilities, SC becomes a natural fit. Data operands in SC take the form of bitstreams which encode probabilities: independent of the length (and interleaving of 0s and 1s), the ratio of the number of 1s to the length of the bitstream determines the operand value. Computation accuracy increases with the length of the bitstream at the cost of higher-latency stochastic operations [194]. Still, computing with probabilities can reduce arithmetic complexity significantly, such that the hardware resource cost and the power consumption become orders of magnitude less than their conventional (i.e., non-stochastic) counterparts [196, 197]. At the same time, computing with probabilities results in better tolerance to inaccuracy in input data operands [194].

The common focus of SC proposals from 1960s onwards has been stochastic logic (arithmetic), neglecting memory, which represents a crucial system component. Memory mainly serves as a repository for data collected from external resources (e.g., sensors) or data generated by previous steps of computation, to be used at later stages of computation. Algorithmic characteristics dictate both, the memory capacity requirement



(a) Baseline Near-Sensor Stochastic Image Processor featuring Digital Memory



(b) StochMem featuring Analog Memory

Figure 7.1: Baseline Near-Sensor Stochastic Image Processor vs. StochMem.

and the memory access pattern (particularly for data re-use). Most SC proposals deploy conventional digital memories (designed and optimized for non-stochastic computing) to address such algorithmic needs. Unfortunately, this practice increases hardware design complexity due to the discrepancy in conventional digital (i.e., non-stochastic) and stochastic data representations. Digital to/from stochastic data conversion can reach 80% or more of the overall energy consumption and hardware cost, which can easily diminish any benefit from stochastic computing [19, 20]. *In this study, we rethink the memory system design for stochastic computing.*

Practically *seamless* conversion options between analog and stochastic data representations [198, 199] makes analog memory stand out as a particularly promising point in the memory design space for SC. The downside is potential loss in data accuracy, where a divergence between the written/stored and the read values (at the same memory address) often becomes inevitable, however, which stochastic logic can mask due to its implicit tolerance to inaccuracy in input data operands.

This study quantitatively characterizes the potential of analog memory for seamless SC, using a representative near-sensor stochastic image processing system as a case study[1]. We will refer to the resulting (practically) seamless stochastic system as StochMem. Cameras have already become ubiquitous sensors. There is a demand for near-sensor image processing both to reduce costly communication with the cloud and

to enhance security and privacy. [1] Real-time image processing algorithms often track differences between a stream of frames. It is not uncommon that the processing of the instantaneous frame requires comparison to a history of previously processed frames, which has to be stored in and retrieved from some form of memory. In the following, we will cover five representative image processing applications which span diverse compute and memory access characteristics.

7.2 Toward Seamless SC

We will first compare and contrast StochMem featuring analog memory with the corresponding stochastic near-sensor image processor featuring conventional digital memory as a representative baseline.

7.2.1 Baseline: Stochastic Logic + Conventional Memory

Fig. 7.1a provides an overview for the baseline stochastic near-sensor image processor featuring conventional digital memory (designed and optimized for non-stochastic computing). The input data operands may represent the result bitstreams of previous steps of (stochastic) computation, or may directly come from analog image sensors. To be able to store such input data in conventional digital memory, a *Stochastic to Digital Converter*, SDC (for stochastic input bitstreams) or an *Analog to Digital Converter*, ADC (for analog inputs coming from sensors) become necessary. Moreover, further (stochastic) processing of the stored data necessitates a *Digital to Stochastic Converter*, DSC, upon data retrieval from digital memory. In the following we briefly describe key system components.

Stochastic Logic incorporates a circuit of basic Boolean gates to carry out the application-specific stochastic computation (Section 7.3.2). The inputs and outputs are both stochastic bitstreams.

[1] 1. Non-stochastic, analog near-sensor image processing accelerators such as [200] exist. The focus of this study is not design and exploration of image processing accelerators. The scope rather is memory system design for stochastic computing where we use a representative stochastic system to characterize the impact of memory.

Stochastic to Digital Converter (SDC) can generate the conventional binary representation for any stochastic bitstream. A digital counter usually serves the purpose, by keeping track of the number of 1s in the input bitstream to be converted. An SDC carries out data conversion if the inputs to the stochastic system represent result bitstreams from previous steps of (stochastic) computation.

Analog to Digital Converter (ADC) becomes necessary if the inputs to the stochastic system directly come from analog image sensors. Conventional ADCs can serve the purpose. For most applications of SC (including the case study in this study) an 8 to 10-bit ADC is sufficient [196].

Digital to Stochastic Converter (DSC) transforms conventional binary data retrieved from digital memory (for further stochastic processing) to stochastic bitstreams. Commonly, DSC achieves this by comparing an unbiased random number (obtained from a random number generator) to the binary value to be converted. A one is attached to the output (stochastic) bitstream if the random number is less than the binary value (to be converted); zero, otherwise. The random number generator can rely on physical random sources or pseudo-random constructs such as Linear Feedback Shift Registers (LFSRs).

7.2.2 StochMem: Stochastic Logic + Analog Memory

The data converters (SDC or ADC and DSC) incorporated into the baseline stochastic system from Fig. 7.1a each has a significant energy and area footprint [19], which can easily nullify potential benefits from SC. In order to reduce this overhead, StochMem replaces the conventional digital memory with its analog counterpart. Fig. 7.1b provides the overview for the resulting SC system. In the following we briefly describe key StochMem components:

Stochastic Logic is the same as under the baseline system.

Stochastic to Analog Converter (SAC) replaces the SDC of the conventional system. SAC can generate the analog representation for any stochastic bitstream. A conventional analog integrator can serve the purpose, by measuring the fraction of time a stochastic input bitstream stays at logic 1. Such an integrator usually has a smaller energy and area footprint than the SDC of the baseline system (Section 7.3.3). A SAC carries out data conversion if the inputs to StochMem represent result bitstreams from previous

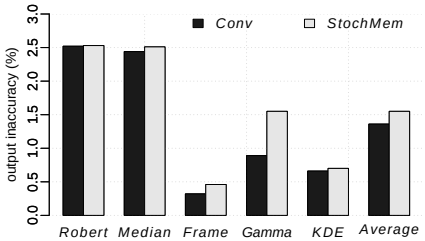
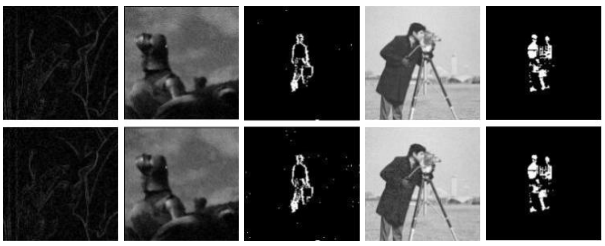


Figure 7.2: Output inaccuracy of the baseline vs. StochMem.



(a) *Robert* (b) *Median* (c) *Frame* (d) *Gamma* (e) *KDE*

Figure 7.3: Output images: Baseline (StochMem) on top (bottom).

steps of (stochastic) computation.

Analog to Stochastic Converter (ASC) transforms data from analog memory (for further stochastic processing) to stochastic bitstreams, similar to the DSC of the conventional system. As representative examples, [198, 199] both cover energy-efficient ways for generating stochastic bitstreams from analog inputs.

7.3 Evaluation Setup

7.3.1 System Design

We evaluate three stochastic near-sensor image-processing designs: two different implementations of the baseline from Fig. 7.1a ($Conv_{LFSR}$ and $Conv_{MTJ}$) and StochMem. The two baseline designs differ in the implementation of data converters as follows:
$Conv_{LFSR}$: The baseline SC system featuring a 10-bit LFSR and a comparator as the DSC unit.
$Conv_{MTJ}$: The baseline featuring a DAC followed by an MTJ-based ASC as a more energy-efficient DSC. The rest of the system is identical to $Conv_{LFSR}$.

All systems first store the input in the memory. Then, they convert it to stochastic, and feed it to the stochastic logic.

(a) *Robert* (b) *Median* (c) *Frame* (d) *Gamma* (e) *KDE*

Figure 7.4: Input (expected output) per application on top (bottom).

7.3.2 Stochastic Applications

To evaluate *Stochastic Logic* from Fig. 7.1, we use stochastic circuits of five representative image processing applications: *Robert* (Robert's cross edge detection), *Median* (median filter noise reduction), *Frame* (frame difference-based image segmentation) from [196]; *Gamma* (gamma correction) from [20]; and *KDE* (kernel density estimation-based image segmentation) from [201].

As input, we use 128×128 gray-scale images for *Robert, Median, Frame*, and *Gamma*; and 33 recent frames of a video, for *KDE*. Fig. 7.4 shows the input (expected output) images used for each application on the top (bottom) row. Expected output captures the maximum-possible accuracy. To calculate the accuracy of the end results, we calculate the average pixel-by-pixel difference between the output image of each stochastic circuit and the corresponding maximum-possible-accuracy output.

7.3.3 Hardware Parameters

Table 7.1 summarizes the area and energy consumption of different units of the evaluated stochastic systems. We synthesize logic units (including the stochastic circuit implementations of the five benchmark applications from Section 7.3.2), LFSR, digital comparator, and counter units using Synopsys Design Compiler vH2013.12 with a 45nm gate library. The Floating-Gate (FG) analog memory implementation follows [205]. To model inaccuracy of FG memory, we add Gaussian noise (with standard deviation from measured data in [205]) to the stored data.

For a fair evaluation, we assume that the input to both the baseline designs and

Table 7.1: Area and energy breakdown.

Stochastic Logic		
Circuit	Area (um^2)	Energy (pJ)(@1GHz)
Robert	339	0.440
Median	5382	4.090
Frame	457	0.413
Gamma	76	0.042
KDE	8691	7.094
Baseline System Parameters		
Unit	Area (um^2)	Energy (pJ)(@1GHz)
ADC 10-bit [202, 203]	50,000	20
SRAM cell	0.35	10
DSC: 10-bit LFSR	194	0.355
DSC: 10-bit Comparator	96	0.041
DSC: DAC 8-bit [204]	16,000	64
SDC: 10-bit Counter	254	0.179
StochMem System Parameters		
Unit	Area (um^2)	Energy (pJ)(@1GHz)
Analog memory cell [205]	58.7	10 (RD) / 100 (WR)
ASC [199]	15	0.030
SAC (integrator)	110	0.010

StochMem directly comes from analog image sensors. All designs output a stochastic bitstream. Therefore, the evaluated systems do not feature an SDC or SAC on the feedback path from memory (Fig. 7.1). However, we include these units in Table 7.1 for the sake of completeness. SAC area (energy) cost is 2.3× (17.9×) less than SDC. Accordingly, if the evaluated systems deployed these units (as explained in Section 7.2), StochMem would have shown even larger gains when compared to the baseline.

7.4 Evaluation

Since all three alternative designs operate at the same frequency, they have similar throughput. So, we start the evaluation with a quantitative characterization of the accuracy loss in the outputs due to the potential read-write discrepancy of the analog memory incorporated in StochMem. We continue with energy consumption and conclude with area cost.

7.4.1 Output Accuracy of StochMem

A known downside of analog memory technologies is the potential discrepancy between values read and written/stored. We model the impact of this discrepancy after the accuracy measurements of a representative analog memory implementation [205]. All evaluated benchmark applications produce images as output. Therefore, we capture the accuracy loss in the output by the average per-pixel deviation (and SSIM [190]) from the "expected" output for each application as shown in the bottom row of Fig. 7.4.

Fig. 7.2 demonstrates the % output inaccuracy (in terms of average per-pixel deviation) of StochMem and the baseline designs for all applications under a stochastic bitstream length of 1024. The y-axis is normalized to the expected accuracy values corresponding to the images in the bottom row of Fig. 7.4. The two baseline designs evaluated, $Conv_{LFSR}$ and $Conv_{MTJ}$ (Section 7.3.1), feature the very same output inaccuracy, as given by the *Conv* bar in Fig. 7.2. We observe that, overall, the degradation (with respect to *Conv*) in the output accuracy of StochMem remains negligible. Only for *Gamma*, the inaccuracy becomes around 0.7% worse than *Conv*. For all other applications, the inaccuracy worsens by less than 0.15%. On average, the % output inaccuracy of StochMem is 1.55%; of *Conv*, 1.36%, with respect to the expected outputs.

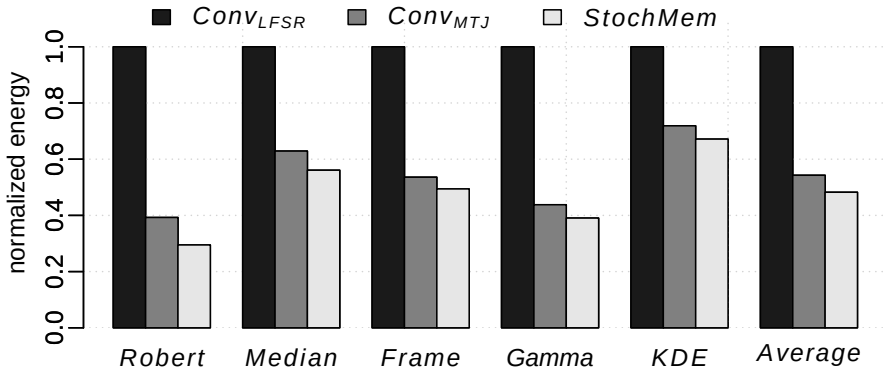


Figure 7.5: Energy consumption normalized to $Conv_{LFSR}$.

Besides, on average SSIM gets 3.2% worse for different applications. For the worst-case application, *Gamma*, SSIM gets 7.3% worse than *Conv*.

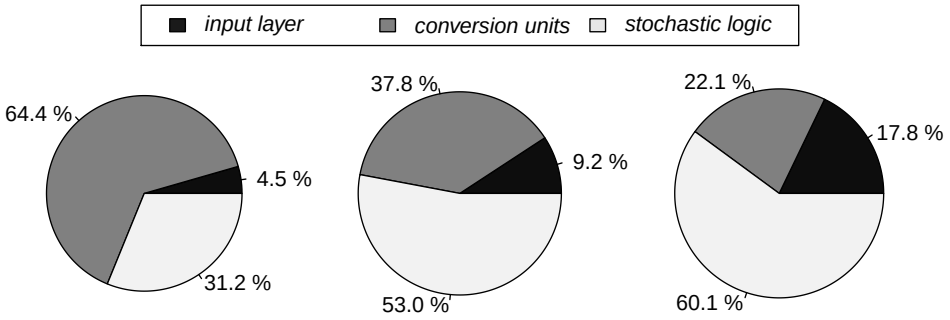
Fig. 7.3 tabulates the output images for all benchmark applications under StochMem and *Conv*. In accordance with the comparison results from Fig. 7.2, the difference in output accuracy is barely perceivable.

We repeat these experiments for 3 different bitstream lengths: 128, 256, and 512 bits. The average output inaccuracy of StochMem with respect to *Conv* increases from 4.08% to 4.21%, from 2.63% to 2.77%, and from 1.87% to 2.03%, as the bitstream length increases from 128 to 512, respectively. The relatively small degradation in the output inaccuracy is in line with the experimental outcomes summarized in Fig.s 7.2 and 7.3.

7.4.2 Reduction in Energy Consumption

We next compare and contrast the energy consumption of the evaluated stochastic designs. In the following, we report the experimental results for a bitstream length of 1024 without loss of generality. As Fig. 7.5 depicts, due to its more energy-efficient DSC implementation, $Conv_{MTJ}$ can decrease the energy consumption with respect to $Conv_{LFSR}$ significantly, by 45.7% on average. Introducing analog memory – i.e., StochMem – can reduce the energy consumption further, by 11.1% on average over $Conv_{MTJ}$.

To demonstrate the sources of these energy gains, we quantify the share of energy spent in different units. We expect an energy-efficient stochastic system to spend most



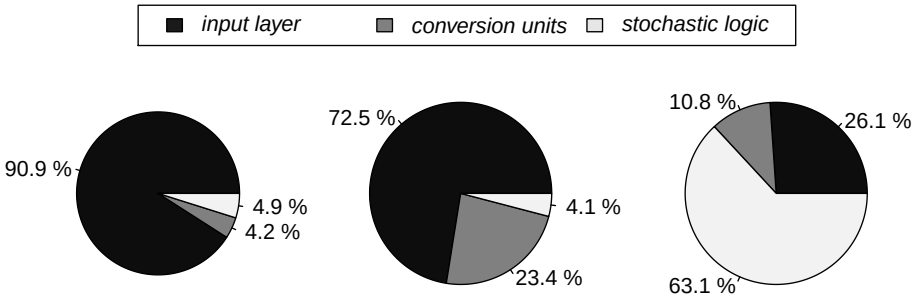
(a) $Conv_{LFSR}$ (b) $Conv_{MTJ}$ (c) StochMem

Figure 7.6: Share of energy consumed by different units.

Table 7.2: Area in μm^2.

Apps	Logic	$\boldsymbol{Conv_{LFSR}}$				$\boldsymbol{Conv_{MTJ}}$					***StochMem***		
		Memory	ADC	DSC	Total	Memory	ADC	DAC	ASC	Total	Memory	ASC	Total
Robert	339	21	50000	1450	51810	21	50000	16000	75	66435	183	75	597
Median	5382	38		2900	58320	38			150	71570	336	150	5868
Frame	457	17		772	51246	17			60	66534	153	60	670
Gamma	76	35		1156	51267	35			120	66231	306	120	502
KDE	8691	122		6166	64979	122			630	75443	1071	630	10392

of its energy budget on computation, rather than on data conversion and input operand retrieval. Pie charts from Fig. 7.6 differentiate between the shares of energy spent in the *input layer* (which covers the input operand retrieval and hence constitutes the ADC, if applicable, and memory units); in the *conversion units* (which constitute the ASC or DSC); and in the *stochastic logic* (which captures the actual computation). Figures 7.6a, 7.6b, and 7.6c, show the shares for $Conv_{LFSR}$, $Conv_{MTJ}$, and StochMem separately (Section 7.3.1). As the charts reveal, share of *stochastic logic* (*conversion units*) increases (decreases) from 31.2% (64.4%) to 53.0% (37.8%) and to 60.1% (22.1%), as we move from $Conv_{LFSR}$ to $Conv_{MTJ}$ and to StochMem respectively. StochMem represents the most energy efficient design, featuring the lowest (highest) energy share for data conversion (computation), when compared to $Conv_{LFSR}$ and $Conv_{MTJ}$.



(a) $Conv_{LFSR}$ (b) $Conv_{MTJ}$ (c) StochMem

Figure 7.7: Pie-charts demonstrating share of hardware cost (in terms of area) across different units.

7.4.3 Reduction in Area

In this section, we evaluate the area cost of each alternative. Since tailoring ADC and DAC units to each application was out of the scope of this study, for the baselines (i.e., $Conv_{LFSR}$ and $Conv_{MTJ}$) we deploy an ADC and a DAC unit of minimal area (which represents the hypothetical best-case in terms of area cost), even if these units fail short of providing the required precision. Accordingly, if we were to incorporate realistic ADC or DAC units (which would likely incur a much higher area overhead), StochMem (which does not employ any ADC or DAC) would have shown even larger area savings in comparison to the baseline.

Table 7.2 summarizes the area cost for the evaluated stochastic designs (columns) for the stochastic benchmark applications (rows). While $Conv_{MTJ}$ consumes notably less energy than $Conv_{LFSR}$ (Section 7.4.2), it requires an extra DAC which increases the area overhead (with respect to $Conv_{LFSR}$) by 20.0% on average. On the other hand, StochMem can cut the area cost significantly, by about 93.7% (with respect to $Conv_{LFSR}$) on average, by eliminating the need for costly conversion units. Only StochMem can deliver area and energy benefits at the same time.

Fig. 7.7 depicts a detailed break-down of area consumption among different units. Similar to Fig. 7.6, pie charts from Fig. 7.7 differentiate between the shares of area in the *input layer*, *conversion units*, and *stochastic logic*, respectively. Only 4.9% of the

area in $Conv_{LFSR}$ goes to the *stochastic logic*, while the *input layer* consumes 90.9%. *Stochastic logic* in $Conv_{MTJ}$ has even a smaller share of area (4.1%) when compared to $Conv_{LFSR}$. On the other hand, in StochMem, 63.1% of the area goes to *stochastic logic*; only 10.8%, to *conversion units*.

Data conversion in conventional SC systems necessitates high-overhead units such as LFSRs+comparators, ADCs, or DACs. StochMem-like SC systems, on the other hand, can eliminate or replace these units with lighter-weight counterparts leading to substantial energy and area savings.

Chapter 8

Conclusion

Moore's Law projected that the most cost-effective number of transistors per integrated circuit (IC) doubles every year, which has stayed almost true for many decades, until today. Robert H. Dennard formulated another trend, ICs getting faster and more energy-efficient exponentially year-by-year as well, known as *Dennard Scaling*. This trend was crucial, since power budget of ICs is severely bounded due to cooling limits, and cannot scale well. Dennard Scaling does not hold any more, leading to severe growth in total power consumption of newer ICs, bringing systems to a point where large number of transistors of ICs become unusable due to overheating, resulting in *the dark silicon* problem. In this era, with a limited power budget, the only way to increase performance is by improving energy-efficiency.

In this thesis, we explored a diverse set of approaches to improve energy-efficiency of computing systems, and consequently, performance. In Chapter 2, we first demonstrated how minimization of power conversion loss in on-chip voltage regulators via gating induces thermal concerns. Then, we presented a set of practical thermally-aware gating policies to mitigate thermal concerns of the on-chip voltage regulators, while maintaining an acceptable voltage noise profile. In Chapter 3, we introduced a novel covert channel induced by aggressive power management techniques used in modern processors to maximize energy-efficiency of computing. Besides, we characterized this channel on two representative processors and observed communication rates as high as 120 bits per second. In Chapter 4, we demonstrated an accelerator for read mapping based on a Processing Near-Memory technology, TCAM, which is suitable for search-intensive

workloads. We evaluated the proposed design, and showed that it outperforms an alternative ASIC-based accelerator by 3.6×. In Chapter 5, we introduced a modular architecture-level model of parametric variation for Thin-Channel switches, which replace conventional planar switches for higher energy-efficiency. Besides, we showed that Thin Channel switches are more prone to variation, rendering parametric variation as a crucial task in designing processors based on these devices. In Chapter 6, we explored how spatio-temporal omission of contention-heavy synchronization points in parallel applications can reduce energy waste in synchronization, and consequently, improve performance while keeping output accuracy loss acceptable. Finally, in Chapter 7, we showed how analog memories are more suitable for stochastic computing systems, compared to the conventional digital memories. Using 5 case study applications, we demonstrated that reducing the energy wasted in data conversion can lead to around 52.8% reduction in total energy consumption on average, while inducing negligible accuracy loss.

References

[1] Gordon E Moore. Cramming more components onto integrated circuits. electronics 38 (8): 114–117, 1965.

[2] Robert H Dennard, Fritz H Gaensslen, V Leo Rideout, Ernest Bassous, and Andre R LeBlanc. Design of ion-implanted mosfet's with very small physical dimensions. *IEEE Journal of Solid-State Circuits*, 9(5):256–268, 1974.

[3] Mark Horowitz. 1.1 computing's energy problem (and what we can do about it). In *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2014 IEEE International*, pages 10–14. IEEE, 2014.

[4] David Brooks. Whats the future of technology scaling? (`https://www.sigarch.org/whats-the-future-of-technology-scaling/`), October 2018.

[5] Hadi Esmaeilzadeh, Emily Blem, Renee St. Amant, Karthikeyan Sankaralingam, and Doug Burger. Dark silicon and the end of multicore scaling. In *Proceedings of the 38th Annual International Symposium on Computer Architecture*, ISCA '11, pages 365–376, New York, NY, USA, 2011. ACM.

[6] Ganesh Venkatesh, Jack Sampson, Nathan Goulding, Saturnino Garcia, Vladyslav Bryksin, Jose Lugo-Martinez, Steven Swanson, and Michael Bedford Taylor. Conservation cores: reducing the energy of mature computations. In *ACM SIGARCH Computer Architecture News*, volume 38, pages 205–218. ACM, 2010.

[7] Ulya R. Karpuzcu, Brian Greskamp, and Josep Torrellas. The BubbleWrap Many-core: Popping Cores for Sequential Acceleration. In *International Symposium on Microarchitecture*, December 2009.

[8] Andrew Danowitz, Kyle Kelley, James Mao, John P. Stevenson, and Mark Horowitz. Cpu db: Recording microprocessor history. *Commun. ACM*, 55(4):55–63, April 2012.

[9] Selçuk Köse. Thermal Implications of On Chip Voltage Regulation: Upcoming Challenges and Possible Solutions. In *Proceedings of the IEEE/ACM Design Automation Conference*, June 2014.

[10] S Karen Khatamifard, Longfei Wang, Weize Yu, Selçuk Köse, and Ulya R Karpuzcu. Thermogater: Thermally-aware on-chip voltage regulation. In *Computer Architecture (ISCA), 2017 ACM/IEEE 44th Annual International Symposium on*, pages 120–132. IEEE, 2017.

[11] S Karen Khatamifard, Longfei Wang, Amitabh Das, Selçuk Köse, and Ulya R Karpuzcu. Powert channels: A novel class of covert communication exploiting power management vulnerabilities. In *International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2019.

[12] S Karen Khatamifard, Longfei Wang, Selçuk Köse, and Ulya R Karpuzcu. A new class of covert channels exploiting power management vulnerabilities. *IEEE Computer Architecture Letters*, 17(2):201–204, 2018.

[13] Hung-Wei Tseng, Yang Liu, Mark Gahagan, Jing Li, Yanqin Jing, and Steven J Swanson. *Gullfoss: Accelerating and simplifying data movement among heterogeneous computing and storage resources*. Department of Computer Science and Engineering, University of California, San Diego, 2015.

[14] J. Li, R. Montoye, M. Ishii, K. Stawiasz, T. Nishida, K. Maloney, G. Ditlow, S. Lewis, T. Maffitt, R. Jordan, L. Chang, and P. Song. 1Mb 0.41 μm2 2T-2R cell nonvolatile TCAM with two-bit encoding and clocked self-referenced sensing. In *Symposium on VLSI Circuits*, June 2013.

[15] S Karen Khatamifard, Zamshed Chowdhury, Nakul Pande, Meisam Razaviyayn, Chris Kim, and Ulya R Karpuzcu. A non-volatile near-memory read mapping accelerator. *arXiv preprint arXiv:1709.02381*, 2017.

[16] K. Ahmed and K. Schuegraf. Transistor Wars. *IEEE Spectrum*, 48(11), 2011.

[17] Karen S. Khatamifard, Michael Resch, Nam Sung Kim, and Ulya R. Karpuzcu. VARIUS-TC: A Modular Architecture-Level Model of Parametric Variation for Thin-Channel Switches. In *International Conference on Computer Design (ICCD)*, October 2016.

[18] S. K. Khatamifard, I. Akturk, and U. R. Karpuzcu. On Approximate Speculative Lock Elision. *IEEE Transactions on Multiscale Computing Systems, Special Issue on Emerging Technologies and Architectures for Manycore Computing*, November 2017.

[19] A. Alaghi, Cheng Li, and J.P. Hayes. Stochastic circuits for real-time image-processing applications. In *Design Automation Conference (DAC), 2013 50th ACM / EDAC / IEEE*, pages 1–6, May 2013.

[20] Weikang Qian, Xin Li, M.D. Riedel, K. Bazargan, and D.J. Lilja. An architecture for fault-tolerant computation with stochastic logic. *Computers, IEEE Transactions on*, 60(1):93–105, Jan 2011.

[21] S. K. Khatamifard, H. Najafi, A. Ghoreyshi, U. R. Karpuzcu, and D. J. Lilja. StochMem: Towards Seamless Stochastic Computing Systems with Analog Memories. *Computer Architecture Letters (CAL)*, January 2018. In press.

[22] Mark Horowitz. Computing's Energy Problem (and what we can do about it). *Keynote at IEEE International Solid-State Circuits Conference*, 2014.

[23] Nasser Kurd, Muntaquim Chowdhury, Edward Burton, Thomas P Thomas, Christopher Mozak, Brent Boswell, Praveen Mosalikanti, Mark Neidengard, Anant Deval, Ashish Khanna, et al. Haswell: A Family of IA 22nm Processors. In *Proceedings of the IEEE International Solid-State Circuits Conference*, February 2014.

[24] Edward A Burton, Gerhard Schrom, Fabrice Paillet, Jonathan Douglas, William J Lambert, Kaladhar Radhakrishnan, and Michael J Hill. FIVR – Fully Integrated Voltage Regulators on 4th Generation Intel Core SoCs. In *Applied Power Electronics Conference and Exposition (APEC), 2014 Twenty-Ninth Annual IEEE*, pages 432–439. IEEE, 2014.

[25] Zeynep Toprak-Deniz, Michael Sperling, John Bulzacchelli, Gregory Still, Ryan Kruse, Seongwon Kim, David Boerstler, Tilman Gloekler, Raphael Robertazzi, Kevin Stawiasz, et al. Distributed System of Digitally Controlled Microregulators Enabling Per-Core DVFS for the POWER8^{TM} Microprocessor. In *Proceedings of the IEEE International Solid-State Circuits Conference*, February 2014.

[26] Seong Joong Kim, Romesh Kumar Nandwana, Qadeer Khan, Robert Pilawa-Podgurski, and Pavan Kumar Hanumolu. A 1.8V 30-to-70MHz 87% Peak-Efficiency 0.32mm^2 4-Phase Time-Based Buck Converter Consuming 3μA/MHz Quiescent Current in 65nm CMOS. In *Proceedings of the IEEE International Solid-State Circuits Conference*, February 2015.

[27] Sung-Yun Park, Jihyun Cho, Kyuseok Lee, and Euisik Yoon. PWM Buck Converter with >80% PCE in 45μA-to-4mA Loads Using Analog-Digital Hybrid Control for Implantable Biomedical Systems. In *Proceedings of the IEEE International Solid-State Circuits Conference*, February 2015.

[28] Yi-Ping Su, Chiun-He Lin, Shen-Yu Peng, Ru-Yu Huang, Te-Fu Yang, Shin-Hao Chen, Ting-Jung Lo, Ke-Homg Chen, Chin-Long Wey, Ying-Hsi Lin, et al. 90% Peak Efficiency Single-Inductor-Multiple-Output DC-DC Buck Converter with Output Independent Gate Drive Control. In *Proceedings of the IEEE International Solid-State Circuits Conference*, February 2015.

[29] Min Kyu Song, Lei Chen, Joseph Sankman, Stephen Terry, and Dongsheng Ma. A 20V 8.4W 20MHz Four-Phase GaN DC-DC Converter with Fully On-Chip Dual-SR Bootstrapped GaN FET Driver Achieving 4ns Constant Propagation Delay and 1ns Switching Rise Time. In *Proceedings of the IEEE International Solid-State Circuits Conference*, February 2015.

[30] Christopher Schaef, Kapil Kesarwani, and Jason T Stauth. A Variable-Conversion-Ratio 3-Phase Resonant Switched Capacitor Converter with 85% Efficiency at 0.91Wmm^2 using 1.1nH PCB-Trace Inductors. In *Proceedings of the IEEE International Solid-State Circuits Conference*, February 2015.

[31] Toke Meyer Andersen, Florian Krismer, Johann Walter Kolar, Thomas Toifl, Christian Menolfi, Lukas Kull, Thomas Morf, Marcel Kossel, Matthias Brändii, and

Pier Andrea Francese. A feedforward controlled on-chip switched-capacitor voltage regulator delivering 10w in 32nm soi cmos. In *Solid-State Circuits Conference-(ISSCC), 2015 IEEE International*, pages 1–3. IEEE, 2015.

[32] Yan Lu, Junmin Jiang, Wing-Hung Ki, C Patrick Yue, Sai-Weng Sin, U Seng-Pan, and Rui Paulo Martins. A 123-phase DC-DC Converter-Ring with Fast-DVS for Microprocessors. In *Proceedings of the IEEE International Solid-State Circuits Conference*, February 2015.

[33] Junmin Jiang, Yan Lu, Cheng Huang, Wing-Hung Ki, and Philip KT Mok. A 2-3-Phase Fully Integrated Switched-Capacitor DC-DC Converter in Bulk CMOS for Energy-Efficient Digital Circuits with 14% Efficiency Improvement. In *Proceedings of the IEEE International Solid-State Circuits Conference*, February 2015.

[34] Abhishek A Sinkar, Hao Wang, and Nam Sung Kim. Workload-Aware Voltage Regulator Optimization for Power Efficient Multi-Core Processors. In *Proceedings of the Conference on Design, Automation and Test in Europe*, March 2012.

[35] Francisco Javier Mesa-Martinez, Ehsan K. Ardestani, and Jose Renau. Characterizing Processor Thermal Behavior. In *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems*, March 2010.

[36] Joonho Kong, Sung Woo Chung, and Kevin Skadron. Recent Thermal Management Techniques for Microprocessors. *ACM Computing Survey*, 44(3), June 2012.

[37] Wei Huang, Shougata Ghosh, Sivakumar 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 (VLSI) Systems*, 14(5), May 2006.

[38] Wei Huang, Mircea R Stan, Sudhanva Gurumurthi, Robert J Ribando, and Kevin Skadron. Interaction of Scaling Trends in Processor Architecture and Cooling. In *IEEE Semiconductor Thermal Measurement and Management Symposium*, February 2010.

[39] Wael El-Essawy. IPMItoolRaw Command Interface to OpenPOWER POWER8 On Chip Controller: Sensor Reading Commands (`https://github.com/open-power/docs/blob/master/occ/OCC_ipmitool_sensors.pdf`), Version 0.4 (2016).

[40] Pingqiang Zhou, Won Ho Choi, Bongjin Kim, Chris H Kim, and Sachin S Sapatnekar. Optimization of On-Chip Switched-Capacitor DC-DC Converters for High-Performance Applications. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, November 2012.

[41] Selçuk Köse and Eby G Friedman. Distributed On-Chip Power Delivery. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 2(4), December 2012.

[42] Inna Vaisband and Eby G Friedman. Heterogeneous Methodology for Energy Efficient Distribution of On-Chip Power Supplies. *IEEE Transactions on Power Electronics*, 28(9), September 2013.

[43] Toke M Andersen, Florian Krismer, Johann W Kolar, Thomas Toifl, Christian Menolfi, Lukas Kull, Thomas Morf, Marcel Kossel, Matthias Brändli, Peter Buchmann, et al. A 4.6 w/mm 2 power density 86% efficiency on-chip switched capacitor dc-dc converter in 32 nm soi cmos. In *Applied Power Electronics Conference and Exposition (APEC), 2013 Twenty-Eighth Annual IEEE*, pages 692–699. IEEE, 2013.

[44] Eric J Fluhr, Joshua Friedrich, Daniel Dreps, Victor Zyuban, Gregory Still, Christopher Gonzalez, Allen Hall, David Hogenmiller, Frank Malgioglio, Ryan Nett, et al. POWER8: A 12-Core Server-Class Processor in 22nm SOI with 7.6Tb/s Off-Chip Bandwidth. In *Proceedings of the IEEE International Solid-State Circuits Conference*, February 2014.

[45] Wonyoung Kim, Meeta S Gupta, Gu-Yeon Wei, and David Brooks. System Level Analysis of Fast, Per-Core DVFS Using On-Chip Switching Regulators. In *Proceedings of the IEEE International Symposium on High Performance Computer Architecture*, February 2008.

[46] Hans Meyvaert, Gerard Villar Piqué, Ravi Karadi, Henk Jan Bergveld, and Michiel SJ Steyaert. A Light-Load-Efficient 111 Switched-Capacitor DC-DC Converter with 94.7% Efficiency While Delivering 100mW at 3.3V. In *Proceedings of the IEEE International Solid-State Circuits Conference*, February 2015.

[47] Per Hammarlund, Alberto J Martinez, Atiq A Bajwa, David L Hill, Erik Hallnor, Hong Jiang, Martin Dixon, Michael Derr, Mikal Hunsaker, Rajesh Kumar, et al. Haswell: The Fourth-Generation Intel Core Processor. *IEEE Micro*, 34(2), March 2014.

[48] Waclaw Godycki, Christopher Torng, Ivan Bukreyev, Alyssa Apsel, and Christopher Batten. Enabling Realistic Fine-Grain Voltage Scaling with Reconfigurable Power Distribution Networks. In *Proceedings of the Annual IEEE/ACM International Symposium on Microarchitecture*, December 2014.

[49] Woojoo Lee, Yanzhi Wang, and Massoud Pedram. Optimizing a Reconfigurable Power Distribution Network in Multicore Platform. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 34(7), July 2015.

[50] Woojoo Lee, Yanzhi Wang, and Massoud Pedram. VRCon: Dynamic Reconfiguration of Voltage Regulators in a Multicore Platform. In *Proceedings of the Conference on Design, Automation and Test in Europe*, March 2014.

[51] Orhun Aras Uzun and Selçuk Köse. Converter-Gating: A Power Efficient and Secure On-Chip Power Delivery System. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 4(2), June 2014.

[52] Joonho Kong, Sung Woo Chung, and Kevin Skadron. Recent Thermal Management Techniques for Microprocessors. *ACM Computing Surveys (CSUR)*, 44(3), 2012.

[53] Poki Chen, Chun-Chi Chen, Chin-Chung Tsai, and Wen-Fu Lu. A Time-to-Digital-Converter-Based CMOS Smart Temperature Sensor. *IEEE Journal of Solid-State Circuits*, 40(8), August 2005.

[54] Anuja Sehgal, Peilin Song, and Keith A Jenkins. On-chip Real-Time Power Supply Noise Detector. In *Proceedings of the 32nd European Solid-State Circuits Conference*, September 2006.

[55] Vijay Janapa Reddi, Meeta Gupta, Glenn Holloway, Michael D Smith, Gu-Yeon Wei, and David Brooks. Predicting Voltage Droops Using Recurring Program and Microarchitectural Event Activity. *IEEE Micro*, 30(1), January 2010.

[56] 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 22nd Annual International Symposium on Computer Architecture*, ISCA '95, 1995.

[57] Sam Likun Xi, Hans Jacobson, Pradip Bose, Gu-Yeon Wei, and David Brooks. Quantifying Sources of Error in McPAT and Potential Impacts on Architectural Studies. In *Proceedings of the IEEE International Symposium on High Performance Computer Architecture*, February 2015.

[58] Sheng Li, Jung Ho Ahn, Richard D. Strong, Jay B. Brockman, Dean M. Tullsen, and Norman P. Jouppi. McPAT: An Integrated Power, Area, and Timing Modeling Framework for Multicore and Manycore Architectures. In *Proceedings of the Annual IEEE/ACM International Symposium on Microarchitecture*, December 2009.

[59] Trevor E. Carlson, Wim Heirman, and Lieven Eeckhout. Sniper: Exploring the Level of Abstraction for Scalable and Accurate Parallel Multi-core Simulation. In *International Conference for High Performance Computing, Networking, Storage and Analysis*, 2011.

[60] 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*, 1(1), March 2004.

[61] Runjie Zhang, Kaushik Mazumdar, Brett H Meyer, Ke Wang, Kevin Skadron, and Mircea R Stan. Transient Voltage Noise in Charge-recycled Power Delivery Networks for Many-layer 3D-IC. In *Proceedings of the IEEE/ACM International Symposium on Low Power Electronics and Design*, July 2015.

[62] 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*, June 2014.

[63] Ke Wang, Brett H Meyer, Runjie Zhang, Micrea Stan, and Kevin Skadron. Walking Pads: Managing C4 Placement for Transient Voltage Noise Minimization. In *Proceedings of the IEEE/ACM Design Automation Conference*, June 2014.

[64] Kevin Skadron, Mircea R Stan, Wei Huang, Sivakumar Velusamy, Karthik Sankaranarayanan, and David Tarjan. Temperature-aware Microarchitecture. In *ACM SIGARCH Computer Architecture News*, volume 31, 2003.

[65] David J Lilja. *Measuring Computer Performance: a Practitioner's Guide*. Cambridge University Press, 2005.

[66] Ehsan K Ardestani, Francisco J Mesa-Martinez, Gabriel Southern, Elnaz Ebrahimi, and Jose Renau. Sampling in Thermal Simulation of Processors: Measurement, Characterization, and Evaluation. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 32(8), 2013.

[67] Howard David, Eugene Gorbatov, Ulf R Hanebutte, Rahul Khanna, and Christian Le. Rapl: memory power estimation and capping. In *Low-Power Electronics and Design (ISLPED), 2010 ACM/IEEE International Symposium on*, pages 189–194. IEEE, 2010.

[68] Colin Percival. Cache missing for fun and profit, 2005.

[69] Zhenyu Wu, Zhang Xu, and Haining Wang. Whispers in the hyper-space: High-speed covert channel attacks in the cloud. In *USENIX Security symposium*, pages 159–173, 2012.

[70] Ramya Jayaram Masti, Devendra Rai, Aanjhan Ranganathan, Christian Müller, Lothar Thiele, and Srdjan Capkun. Thermal covert channels on multi-core platforms. In *USENIX Security Symposium*, pages 865–880, 2015.

[71] Hubert Ritzdorf. Analyzing Covert Channels on Mobile Devices. *M.S. Thesis, ETH*, 2012.

[72] Pradip Bose, Alper Buyuktosunoglu, John A Darringer, Meeta S Gupta, Michael B Healy, Hans Jacobson, Indira Nair, Jude A Rivers, Jeonghee Shin, Augusto Vega, et al. Power management of multi-core chips: Challenges and pitfalls. In *Proceedings of the Conference on Design, Automation and Test in Europe*, pages 977–982. EDA Consortium, 2012.

[73] Zeynep Toprak-Deniz, Michael Sperling, John Bulzacchelli, Gregory Still, Rudolf Kruse, Seongwon Kim, David Boerstler, Tilman Gloekler, Raphael Robertazzi, Kevin Stawiasz, et al. 5.2 Distributed system of digitally controlled microregulators enabling per-core DVFS for the POWER8 TM microprocessor. In *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2014 IEEE International*. IEEE, 2014.

[74] Peter Greenhalgh. Big. little processing with arm cortex-a15 & cortex-a7. *ARM White paper*, 17, 2011.

[75] *MPrime*. `https://aur.archlinux.org/packages/mprime/`.

[76] cpuburn-a7 for arm cortex-a7. `https://github.com/ssvb/cpuburn-arm/blob/master/cpuburn-a7.S`. Accessed: 2010-09-30.

[77] Casen Hunger, Mikhail Kazdagli, Ankit Rawat, Alex Dimakis, Sriram Vishwanath, and Mohit Tiwari. Understanding contention-based channels and using them for defense. In *High Performance Computer Architecture (HPCA), 2015 IEEE 21st International Symposium on*, pages 639–650. IEEE, 2015.

[78] Thomas M Cover and Joy A Thomas. *Elements of information theory*. John Wiley & Sons, 2012.

[79] POWER8 On Chip Controlle: Measuring and Managing Power Consumption: `https://hpm.ornl.gov/Archives/HPM15/documents/HPM2015_Rosedahl.pdf`.

[80] Jack Doweck, Wen-Fu Kao, Allen Kuan-yu Lu, Julius Mandelblat, Anirudha Rahatekar, Lihu Rappoport, Efraim Rotem, Ahmad Yasin, and Adi Yoaz. Inside 6th-generation intel core: new microarchitecture code-named skylake. *IEEE Micro*, (2):52–62, 2017.

[81] J. Szefer. Survey of microarchitectural side and covert channels, attacks, and defenses. *IACR Cryptology ePrint Archive*, pages 1–28, 2016.

[82] Zhenghong Wang and Ruby B Lee. Covert and side channels due to processor architecture. In *Computer Security Applications Conference, 2006. ACSAC'06. 22nd Annual*, pages 473–482. IEEE, 2006.

[83] Steven J Murdoch. Hot or not: Revealing hidden services by their clock skew. In *Proceedings of the 13th ACM conference on Computer and communications security*, pages 27–36. ACM, 2006.

[84] Dmitry Evtyushkin and Dmitry Ponomarev. Covert channels through random number generator: Mechanisms, capacity estimation and mitigations. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 843–857. ACM, 2016.

[85] H. Naghibijouybari, K. N. Khasawneh, and N. Abu-Ghazaleh. Constructing and characterizing covert channels on gpgpus. In *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 354–366, 2017.

[86] N. Matyunin, J. Szefer, S. Biedermann, and S. Katzenbeisser. Covert channels using mobile device's magnetic field sensors. In *Proceedings of the 21st Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 525–532, 2016.

[87] R. Spolaor, L. Abudahi, V. Moonsamy, M. Conti, and R. Poovendran. No free charge theorem: A covert channel via usb charging cable on mobile devices. In *Applied Cryptography and Network Security - ACNS 2017*, pages 83–102, 2017.

[88] Rajesh JayashankaraShridevi, Chidhambaranathan Rajamanikkam, Koushik Chakraborty, and Sanghamitra Roy. Catching the flu: emerging threats from a third party power management unit. In *Design Automation Conference (DAC), 2016 53nd ACM/EDAC/IEEE*, pages 1–6. IEEE, 2016.

[89] Adrian Tang, Simha Sethumadhavan, and Salvatore Stolfo. Clkscrew: exposing the perils of security-oblivious energy management. In *USENIX Security Symposium*, 2017.

[90] S. Zhang, A. Tang, Z. Jiang, S. Sethumadhavan, and M. Seok. Blacklist core: machine-learning based dynamic operating-performance-point blacklisting for mitigating power-management security attacks. In *Proceedings of the 23rd IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*, pages 1–6, 2018.

[91] Murugappan Alagappan, Jeyavijayan Rajendran, Miloš Doroslovački, and Guru Venkataramani. Dfs covert channels on multi-core platforms. In *Very Large Scale Integration (VLSI-SoC), 2017 IFIP/IEEE International Conference on*, pages 1–6. IEEE, 2017.

[92] Md Nazmul Islam and Sandip Kundu. Pmu-trojan: on exploiting power management side channel for information leakage. In *Proceedings of the 23rd Asia and South Pacific Design Automation Conference*, pages 709–714. IEEE Press, 2018.

[93] Illumina sequencing by synthesis (SBS) technology: `https://www.illumina.com/technology/next-generation-sequencing/sequencing-technology.html`.

[94] Zachary D Stephens, Skylar Y Lee, Faraz Faghri, Roy H Campbell, Chengxiang Zhai, Miles J Efron, Ravishankar Iyer, Michael C Schatz, Saurabh Sinha, and Gene E Robinson. Big Data: Astronomical or Genomical? *PLOS Biology*, 13(7), July 2015.

[95] S Aluru and N Jammula. A review of hardware acceleration for computational genomics. *IEEE Design & Test*, 31(1), 2014.

[96] Petr Klus, Simon Lam, Dag Lyberg, Ming Sin Cheung, Graham Pullan, Ian McFarlane, Giles SH Yeo, and Brian YH Lam. BarraCUDA - a fast short read sequence aligner using graphics processing units. *BMC Research Notes*, 5(1), January 2012.

[97] Y Chen, B Schmidt, and D L Maskell. A hybrid short read mapping accelerator. *BMC Bioinformatics*, 14(1), 2013.

[98] Qing Guo, Xiaochen Guo, Yuxin Bai, and Engin Ipek. A resistive TCAM accelerator for data-intensive computing. *IEEE International Symposium on Microarchitecture (MICRO)*, 2011.

[99] Melanie Schirmer, Rosalinda D'Amore, Umer Z Ijaz, Neil Hall, and Christopher Quince. Illumina error profiles: resolving fine-scale variation in metagenomic sequencing data. *BMC Bioinformatics*, 17(1), 2016.

[100] J. M. Mullaney, R. E. Mills, W. S. Pittard, and S. E. Devine. Small insertions and deletions (indels) in human genomes. *Human Molecular Genetics*, 19(R2), 2010.

[101] Sung-Min Ahn, Tae-Hyung Kim, Sunghoon Lee, Deokhoon Kim, Ho Ghang, Dae-Soo Kim, Byoung-Chul Kim, Sang-Yoon Kim, Woo-Yeon Kim, Chulhong Kim, Daeui Park, Yong Seok Lee, Sangsoo Kim, Rohit Reja, Sungwoong Jho, Chang Geun Kim, Ji-Young Cha, Kyung-Hee Kim, Bonghee Lee, Jong Bhak, and Seong-Jin Kim. The first korean genome sequence and analysis: full genome sequencing for a socio-ethnic group. *Genome research*, 19(9):1622–1629, 2009.

[102] Jeremiah Wala, Pratiti Bandopadhayay, Noah Greenwald, Ryan O'Rourke, Ted Sharpe, Chip Stewart, Steven E Schumacher, Yilong Li, Joachim Weischenfeldt, Xiaotong Yao, Chad Nusbaum, Peter Campbell, Matthew Meyerson, Cheng-Zhong Zhang, Marcin Imielinski, and Rameen Beroukhim. Genome-wide detection of structural variants and indels by local assembly. *bioRxiv*, page 105080, 2017.

[103] Ryan P. Abo, Matthew Ducar, Elizabeth P. Garcia, Aaron R. Thorner, Vanesa Rojas-Rudilla, Ling Lin, Lynette M. Sholl, William C. Hahn, Matthew Meyerson, Neal I. Lindeman, Paul VanHummelen, and Laura E. MacConaill. Breakmer: detection of structural variation in targeted massively parallel sequencing data using kmers. *Nucleic acids research*, 43(3):e19–e19, 2014.

[104] Ruibang Luo, Thomas Wong, Jianqiao Zhu, Liu Chi-Man, Xiaoqian Zhu, Edward Wu, Lap-Kei Lee, Haoxiang Lin, Wenjuan Zhu, David W. Cheung, Hing-Fung Ting, Siu-Ming Yiu, Shaoliang Peng, Yu Chang, Yingrui Li, Ruiqiang Li, and Tak-Wah Lam. Soap3-dp: fast, accurate and sensitive gpu-based short read aligner. *PloS one*, 8(5), 2013.

[105] Jure Leskovec, Anand Rajaraman, and Jeffrey David Ullman. *Mining of massive datasets*. Cambridge University Press, 2014.

[106] Ben Langmead, Cole Trapnell, Mihai Pop, and Steven L Salzberg. Ultrafast and memory-efficient alignment of short dna sequences to the human genome. *Genome biology*, 10(3):R25, 2009.

[107] David J Lipman and William R Pearson. Rapid and sensitive protein similarity searches. *Science*, 227(4693), 1985.

[108] Jun Wang, Wei Wang, Ruiqiang Li, Yingrui Li, Geng Tian, Laurie Goodman, Wei Fan, Junqing Zhang, Jun Li, Juanbin Zhang, Yiran Guo, Binxiao Feng, Heng Li, Yao Lu, Xiaodong Fang, Huiqing Liang, Zhenglin Du, Dong Li, Yiqing Zhao, and Yujie Hu. The diploid genome sequence of an asian individual. *Nature*, 456(7218):60–65, 2008.

[109] NVBIO Smith-Waterman: `https://developer.nvidia.com/nvbio/`.

[110] Yoongu Kim, Weikun Yang, and Onur Mutlu. Ramulator: A fast and extensible dram simulator. *Computer Architecture Letters*, 15(1):45–49, 2016.

[111] Karthik Chandrasekar, Christian Weis, Yonghui Li, Benny Akesson, Norbert Wehn, and Kees Goossens. Drampower: Open-source dram power & energy estimation tool. *URL: http://www. drampower. info*, 22, 2012.

[112] H. Y. Cheng, W. C. Chien, M. BrightSky, Y. H. Ho, Y. Zhu, A. Ray, R. Bruce, W. Kim, C. W. Yeh, H. L. Lung, and C. Lam. Novel fast-switching and high-data retention phase-change memory based on new ga-sb-ge material. In *IEEE International Electron Devices Meeting (IEDM)*, 2015.

[113] NCSU-EDA. FreePDK45: `https://www.eda.ncsu.edu/wiki/FreePDK45:Contents`.

[114] Linda Wilson. International technology roadmap for semiconductors (itrs). *Semiconductor Industry Association*, 2013.

[115] Andrew B Kahng, Bin Li, Li-Shiuan Peh, and Kambiz Samadi. Orion 2.0: a fast and accurate noc power and area model for early-stage design space exploration. In *Proceedings of the Design, Automation & Test in Europe (DATE)*, 2009.

[116] M. H. Abu-Rahma, Y. Chen, W. Sy, W. L. Ong, L. Y. Ting, S. S. Yoon, M. Han, and E. Terzioglu. Characterization of sram sense amplifier input offset for yield prediction in 28nm cmos. In *IEEE Custom Integrated Circuits Conference (CICC)*, Sept 2011.

[117] 1000 genomes project: `ftp://ftp-trace.ncbi.nih.gov/1000genomes/ftp/technical/reference/`.

[118] Na12878. `ftp://ftp-trace.ncbi.nlm.nih.gov/giab/ftp/data/NA12878/`.

[119] Daichi Fuijiki, Arun Subramaniyan, Tianjun Zhang, Yu Zheng, Reetuparna Das, David Blaauw, and Satish Narayanasamy. GenAx: A Genome Sequencing Accelerator. *ACM/IEEE International Symposium on Computer Architecture (ISCA)*, June 2018.

[120] Subramanian S Ajay, Stephen CJ Parker, Hatice Ozel Abaan, Karin V Fuentes Fajardo, and Elliott H Margulies. Accurate and comprehensive sequencing of personal genomes. *Genome Research*, 21(9), 2011.

[121] Eric S Lander and Michael S Waterman. Genomic mapping by fingerprinting random clones: a mathematical analysis. *Genomics*, 2(3), 1988.

[122] Ruiqiang Li, Yingrui Li, Karsten Kristiansen, and Jun Wang. Soap: short oligonucleotide alignment program. *Bioinformatics*, 24(5), 2008.

[123] Heng Li, Jue Ruan, and Richard Durbin. Mapping short dna sequencing reads and calling variants using mapping quality scores. *Genome Research*, 18(11), 2008.

[124] Ruiqiang Li, Chang Yu, Yingrui Li, Tak-Wah Lam, Siu-Ming Yiu, Karsten Kristiansen, and Jun Wang. Soap2: an improved ultrafast tool for short read alignment. *Bioinformatics*, 25(15), 2009.

[125] Ben Langmead, Cole Trapnell, Mihai Pop, and Steven L Salzberg. Ultrafast and memory-efficient alignment of short dna sequences to the human genome. *Genome Biology*, 10(3), 2009.

[126] Ben Langmead and Steven L Salzberg. Fast gapped-read alignment with bowtie 2. *Nature Methods*, 9(4), 2012.

[127] Heng Li and Richard Durbin. Fast and accurate short read alignment with burrows–wheeler transform. *Bioinformatics*, 25(14), 2009.

[128] Heng Li and Richard Durbin. Fast and accurate long-read alignment with burrows–wheeler transform. *Bioinformatics*, 26(5), 2010.

[129] Wim Vanderbauwhede and Khaled Benkrid. *High-performance computing using FPGAs*. Springer, 2013.

[130] Advait Madhavan, Timothy Sherwood, and Dmitri Strukov. Race logic: A hardware acceleration for dynamic programming algorithms. *ACM/IEEE International Symposium on Computer Architecture (ISCA)*, 2014.

[131] Jeremie Kim, Damla Senol, Hongyi Xin, Donghyuk Lee, Mohammed Alser, Hasan Hassan, Oguz Ergin, Can Alkan, and Onur Mutlu. Genome read in-memory (grim) filter: Fast location filtering in dna read mapping using emerging memory technologies `https://people.inf.ethz.ch/omutlu/pub/GRIM-genome-read-in-memoryfilter_psb17-poster.pdf`, 2017.

[132] F. Zokaee, H. R. Zarandi, and L. Jiang. AligneR: A Process-In-Memory Architecture for Short Read Alignment inReRAMs. *IEEE Computer Architecture Letters*, 2018.

[133] C. Bo, V. Dang, E. Sadredini, and K. Skadron. Searching for potential grna off-target sites for crispr/cas9 using automata processing across different platforms. In *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 737–748, Feb 2018.

[134] Qing Guo, Xiaochen Guo, Ravi Patel, Engin Ipek, and Eby G Friedman. AC-DIMM: associative computing with STT-MRAM. In *ACM/IEEE International Symposium on Computer Architecture (ISCA)*, June 2013.

[135] Leonid Yavits, Shahar Kvatinsky, Amir Morad, and Ran Ginosar. Resistive associative processor. *IEEE Computer Architecture Letters*, 14(2), 2015.

[136] R. Kaplan, L. Yavits, R. Ginosar, and U. Weiser. A resistive cam processing-in-storage architecture for dna sequence alignment. *IEEE Micro*, 37(4):20–28, 2017.

[137] M. Imani, D. Peroni, A. Rahimi, and T. Rosing. Resistive cam acceleration for tunable approximate computing. *IEEE Transactions on Emerging Topics in Computing*, 2017.

[138] Mohesn Imani, Abbas Rahimi, Deqian Kong, Tajana Rosing, and Jan Rabaey. Exploring hyperdimensional associative memory. *IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2017.

[139] Bonan Yan, Zheng Li, Yaojun Zhang, Jianlei Yang, Hai Li, Weisheng Zhao, and Pierre Chor-Fung Chia. A high-speed robust nvm-tcam design using body bias feedback. In *Great Lakes Symposium on VLSI (GVLSI)*, 2015.

[140] E. Karl, Yih Wang, Yong-Gee Ng, Zheng Guo, F. Hamzaoglu, M. Meterelliyoz, J. Keane, U. Bhattacharya, K. Zhang, K. Mistry, and M. Bohr. A 4.6 GHz 162 Mb SRAM Design in 22 nm Tri-Gate CMOS Technology With Integrated Read and Write Assist Circuitry. *IEEE Journal of Solid-State Circuits*, 48(1), Jan 2013.

[141] Darsen Lu, Chung-Hsun Lin, Ali Niknejad, and Chenming Hu. Compact Modeling of Variation in FinFET SRAM cells. *IEEE Design & Test of Computers*, 27(2), 2010.

[142] S.H. Rasouli, K. Endo, and K. Banerjee. Variability Analysis of FinFET-based Devices and Circuits Considering Electrical Confinement and Width Quantization. In *International Conference on Computer-Aided Design (ICCAD)*, Nov 2009.

[143] V B Kleeberger, H Graeb, and U Schlichtmann. Predicting Future Product Performance: Modeling and Evaluation of Standard Cells in FinFET Technologies. In *Design Automation Conference (DAC)*, June 2013.

[144] C Y Lee and N K Jha. FinCANON: A PVT-Aware Integrated Delay and Power Modeling Framework for FinFET-Based Caches and On-Chip Networks. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, (99), 2013.

[145] Predictive Technology Model (PTM). *http://ptm.asu.edu/*.

[146] S. Sinha, G. Yeric, V. Chandra, B. Cline, and Yu Cao. Exploring sub-20nm FinFET Design with Predictive Technology Models. In *Design Automation Conference (DAC)*, June 2012.

[147] S.R. Sarangi, B. Greskamp, R. Teodorescu, J. Nakano, A. Tiwari, and J. Torrellas. VARIUS: A Model of Process Variation and Resulting Timing Errors for Microarchitects. *IEEE Transactions on Semiconductor Manufacturing*, 21(1), February 2008.

[148] Ulya R. Karpuzcu, Krishna B. Kolluru, Nam Sung Kim, and Josep Torrellas. VARIUS-NTV: A Microarchitectural Model to Capture the Increased Sensitivity of Manycores to Process Variations at Near-Threshold Voltages. In *International Conference on Dependable Systems and Networks (DSN)*, June 2012.

[149] D. Markovic, C. C. Wang, L. P. Alarcon, T.-T. Liu, and J. M. Rabaey. Ultralow-Power Design in Near-Threshold Region. *Proceedings of the IEEE*, 98(2), February 2010.

[150] L. Chang, R.K. Montoye, Y. Nakamura, K.A. Batson, R.J. Eickemeyer, R.H. Dennard, W. Haensch, and D. Jamsek. An 8T-SRAM for Variability Tolerance and Low-Voltage Operation in High-Performance Caches. *IEEE Journal of Solid-State Circuits*, (4), April 2008.

[151] Jim Jeffers and James Reinders. *Intel Xeon Phi Coprocessor High-Performance Programming*. Morgan Kaufmann, March 2013.

[152] L. Chang, D. J. Frank, R. K. Montoye, S. J. Koester, B. L. Ji, P. W. Coteus, R. H. Dennard, and W. Haensch. Practical Strategies for Power-Efficient Computing Technologies. *Proceedings of the IEEE*, 98(2), February 2010.

[153] R. G. Dreslinski, M. Wieckowski, D. Blaauw, D. Sylvester, and T. Mudge. Near-Threshold Computing: Reclaiming Moore's Law Through Energy Efficient Integrated Circuits. *Proceedings of the IEEE*, 98(2), February 2010.

[154] A. Tang, Y. Yang, C. Y. Lee, and N. K. Jha. McPAT-PVT: Delay and Power Modeling Framework for FinFET Processor Architectures Under PVT Variations.

IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 23(9), Sept 2015.

[155] M. C. Rinard. Unsynchronized techniques for approximate parallel computing. In *ACM Workshop on Relaxing Synchronization for Multicore and Manycore Scalability (RACES)*, 2012.

[156] Sasa Misailovic, Stelios Sidiroglou, and Martin C Rinard. Dancing with Uncertainty. In *ACM Workshop on Relaxing Synchronization for Multicore and Manycore Scalability (RACES)*, 2012.

[157] L. Renganarayana, V. Srinivasan, R. Nair, and D. Prener. Programming with relaxed synchronization. In *ACM Workshop on Relaxing Synchronization for Multicore and Manycore Scalability (RACES)*, 2012.

[158] Benjamin Recht Feng Niu, Christopher Ré, and Stephen J. Wright. Hogwild!: A lock-free approach to parallelizing stochastic gradient descent. In *Neural Information Processing Systems Conference (NIPS)*, 2011.

[159] Yen-Kuang Chen, J. Chhugani, P. Dubey, C. J. Hughes, Daehyun Kim, S. Kumar, V.W. Lee, A.D. Nguyen, and M. Smelyanskiy. Convergence of recognition, mining, and synthesis workloads and its implications. *Proceedings of the IEEE*, 96(5), 2008.

[160] V. Chippa, D. Mohapatra, and A. Raghunathan. Scalable effort hardware design: Exploiting algorithmic resilience for energy efficiency. *Design Automation Conference (DAC)*, 2010.

[161] V. Wong and M. Horowitz. Soft error resilience of probabilistic inference applications. In *Workshop on Silicon Errors in Logic-System Effects (SELSE)*, 2006.

[162] X. Li and D. Yeung. Application-Level Correctness and its Impact on Fault Tolerance. In *International Symposium on High Performance Computer Architecture (HPCA)*, 2007.

[163] D. D. Thaker, D. Franklin, J. Oliver, S. Biswas, D. Lockhart, T. Metodi, and F. T. Chong. Characterization of error-tolerant applications when protecting control data. In *International Symposium on Workload Characterization (IISWC)*, 2006.

[164] H. Cho, L. Leem, and S. Mitra. Ersa: Error resilient system architecture for probabilistic applications. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, 31(4), 2012.

[165] M. Herlihy, J. Eliot, and B. Moss. Transactional memory: Architectural support for lock-free data structures. In *International Symposium on Computer Architecture (ISCA)*, 1993.

[166] R. Rajwar and J. R. Goodman. Speculative lock elision: enabling highly concurrent multithreaded execution. In *International Symposium on Microarchitecture (MICRO)*, 2001.

[167] J. F. Martinez and J. Torrellas. Speculative synchronization: programmability and performance for parallel codes. *IEEE Micro Magazine*, 23(6), 2003.

[168] G. M. Baudet. Asynchronous iterative methods for multiprocessors. *Journal of ACM*, 25(2), 1978.

[169] F. Allen, M. Burke, R. Cytron, J. Ferrante, and W. Hsieh. A framework for determining useful parallelism. In *International Conference on Supercomputing (ICS)*, 1988.

[170] N. Shavit and D. Touitou. Software transactional memory. In *ACM Symposium on Principles of Distributed Computing (PODC)*, 1995.

[171] A. Dragojevik, P. Felber, V. Gramoli, and R. Guerraoui. Why STM can be more than a research toy. *Communications of the ACM*, 54(4), 2011.

[172] Richard M. Yoo, Christopher J. Hughes, Konrad Lai, and Ravi Rajwar. Performance evaluation of intel transactional synchronization extensions for high-performance computing. In *International Conference on High Performance Computing, Networking, Storage and Analysis (SC)*, 2013.

[173] Jeffrey Dean, Greg S. Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Quoc V. Le, Mark Z. Mao, Marc'Aurelio Ranzato, Andrew Senior, Paul Tucker, Ke Yang, and Andrew Y. Ng. Large scale distributed deep networks. In *International Conference on Neural Information Processing Systems*, pages 1223–1231, 2012.

[174] K Bergman, S Borkar, D Campbell, W Carlson, W Dally, M Denneau, P Franzon, W Harrod, J Hiller, and S Karp. Exascale computing study: Technology challenges in achieving exascale systems. *DARPA Information Processing Techniques Office (IPTO) sponsored study*, 2008.

[175] Daya Shanker Khudia, Babak Zamirai, Mehrzad Samadi, and Scott A Mahlke. Rumba: an online quality management system for approximate computing. *International Symposium on Computer Architecture (ISCA)*, 2015.

[176] Michael Ringenburg, Adrian Sampson, Isaac Ackerman, Luis Ceze, and Dan Grossman. Monitoring and debugging the quality of results in approximate programs. In *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2015.

[177] S. T. Chakradhar and A. Raghunathan. Best-effort computing: Re-thinking parallel software and hardware. In *Design Automation Conference (DAC)*, 2010.

[178] Chi Cao Minh, Jaewoong Chung, C. Kozyrakis, and K. Olukotun. Stamp: Stanford transactional applications for multi-processing. In *International Symposium on Workload Characterizationd (IISWC)*, 2008.

[179] Sasa Misailovic, Michael Carbin, Sara Achour, Zichao Qi, and Martin C. Rinard. Chisel: Reliability- and accuracy-aware optimization of approximate computational kernels. In *International Conference on Object Oriented Programming Systems, Languages, Applications (OOPSLA)*, 2014.

[180] M. de Kruijf, S. Nomura, and K. Sankaralingam. Relax: An architectural framework for software recovery of hardware faults. In *International Symposium on Computer Architecture (ISCA)*, 2010.

[181] A. Sampson, W. Dietl, E. Fortuna, D. Gnanapragasam, L. Ceze, and D. Grossman. Enerj: Approximate data types for safe and general low-power computation. In *Conference on Programming Language Design and Implementation (PLDI)*, 2011.

[182] M. Rinnard. Parallel synchronization-free approximate data structure construction. *USENIX Workshop on Hot Topics in Parallelism*, 2013.

[183] Jaewoong Chung, L. Yen, S. Diestelhorst, M. Pohlack, M. Hohmuth, D. Christie, and D. Grossman. Asf: Amd64 extension for lock-free data structures and transactional memory. In *International Symposium on Microarchitecture (MICRO)*, 2010.

[184] Ulya R. Karpuzcu, Ismail Akturk, and Nam Sung Kim. Accordion: Toward Soft Near-Threshold Computing. In *International Symposium on High Performance Computer Architecture (HPCA)*, 2014.

[185] C. Bienia, S. Kumar, J. P. Singh, and Kai Li. The parsec benchmark suite: Characterization and architectural implications. Technical Report TR-811-08, Princeton University, 2008.

[186] J. A. Stratton, C. Rodrigues, I. J. Sung, N. Obeid, L. W. Chang, N. Anssari, G. D. Liu, and W. W. Hwu. Parboil: A revised benchmark suite for scientific and commercial throughput computing. *Center for Reliable and High-Performance Computing*, 2012.

[187] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta. The SPLASH-2 Programs: Characterization and Methodological Considerations. In *International Symposium on Computer Architecture (ISCA)*, 1995.

[188] Gokcen Kestor, Srdjan Stipic, Osman S Unsal, Adrian Cristal, and Mateo Valero. Rms-tm: A transactional memory benchmark for recognition, mining and synthesis applications. In *4th Workshop on Transactional Computing*, 2009.

[189] Sasa Misailovic et al. Quality of Service Profiling. In *International Conference on Software Engineering (ICSE)*, 2010.

[190] I. Akturk, K. Khatamifard, and U. R. Karpuzcu. On Quantification of Accuracy Loss in Approximate Computing. In *12th Annual Workshop on Duplicating, Deconstructing and Debunking (WDDD)*, 2015.

[191] Kevin E Moore, Jayaram Bobba, Michelle J Moravan, Mark D Hill, David A Wood, et al. Logtm: log-based transactional memory. In *International Symposium on High Performance Computer Architecture (HPCA)*, volume 6, pages 254–265, 2006.

[192] C. Segulja and T. S. Abdelrahman. Architectural support for synchronization-free deterministic parallel programming. *International Symposium on High Performance Computer Architecture (HPCA)*, 2012.

[193] Armin Alaghi and John P. Hayes. Survey of stochastic computing. *ACM Trans. Embed. Comput. Syst.*, 12(2s):92:1–92:19, May 2013.

[194] Armin Alaghi, Weikang Qian, and John P Hayes. The promise and challenge of stochastic computing. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2017.

[195] Kyounghoon Kim, Jungki Kim, Joonsang Yu, Jungwoo Seo, Jongeun Lee, and Kiyoung Choi. Dynamic energy-accuracy trade-off using stochastic computing in deep neural networks. In *Proceedings of the 53rd Annual Design Automation Conference*, DAC '16, pages 124:1–124:6, New York, NY, USA, 2016. ACM.

[196] Peng Li, D.J. Lilja, Weikang Qian, K. Bazargan, and M.D. Riedel. Computation on stochastic bit streams digital image processing case studies. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 22(3):449–462, March 2014.

[197] M. Hassan Najafi, Peng Li, David J. Lilja, Weikang Qian, Kia Bazargan, and Marc Riedel. A reconfigurable architecture with sequential logic-based stochastic computing. *J. Emerg. Technol. Comput. Syst.*, 13(4):57:1–57:28, June 2017.

[198] D. Fick, G. Kim, A. Wang, D. Blaauw, and D. Sylvester. Mixed-signal stochastic computation demonstrated in an image sensor with integrated 2d edge detection and noise filtering. In *Proceedings of the IEEE 2014 Custom Integrated Circuits Conference*, pages 1–4, Sept 2014.

[199] N. Onizawa, D. Katagiri, W. J. Gross, and T. Hanyu. Analog-to-stochastic converter using magnetic tunnel junction devices for vision chips. *IEEE Transactions on Nanotechnology*, PP(99):1–1, 2015.

[200] Robert LiKamWa, Yunhui Hou, Julian Gao, Mia Polansky, and Lin Zhong. Redeye: Analog convnet image sensor architecture for continuous mobile vision. In *Proceedings of the 43rd International Symposium on Computer Architecture*, ISCA '16, pages 255–266, Piscataway, NJ, USA, 2016. IEEE Press.

[201] Peng Li and D.J. Lilja. A low power fault-tolerance architecture for the kernel density estimation based image segmentation algorithm. In *Application-Specific Systems, Architectures and Processors (ASAP), 2011 IEEE International Conference on*, pages 161–168, Sept 2011.

[202] B. Murmann. "ADC Performance Survey 1997-2016," [online]. Available: http://web.stanford.edu/ murmann/adcsurvey.html, 2016.

[203] Lukas Kull, T. Toifl, M. Schmatz, P. A. Francese, C. Menolfi, M. Braendli, M. Kossel, T. Morf, T. Meyer Anderson, and Yusuf Leblebici. A 90GS/s 8b 667mW 64x Interleaved SAR ADC in 32nm Digital SOI CMOS. In *Proceedings of the 2014 ISSCC*, 2014.

[204] W. T. Lin and T. H. Kuo. A 12b 1.6gs/s 40mw dac in 40nm cmos with 70db sfdr over entire nyquist bandwidth. In *2013 IEEE International Solid-State Circuits Conference Digest of Technical Papers*, pages 474–475, Feb 2013.

[205] Junjie Lu, Steven Young, Itamar Arel, and Jeremy Holleman. A 1 tops/w analog deep machine-learning engine with floating-gate storage in 0.13 μm cmos. *IEEE Journal of Solid-State Circuits*, 50(1):270–281, 2015.