First-Order Modeling Frameworks for Power-Efficient and Reliable Multiprocessor Systems

A Dissertation

Presented to

the faculty of the School of Engineering and Applied Science

University of Virginia

In Partial Fulfillment

of the requirements for the Degree

Doctor of Philosophy

Computer Science

by

Liang. Wang

May 2016

Approvals

This dissertation is submitted in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

Computer Science

Liang Wang

Approved:

Kevin Skadron (Advisor)

Mircea R. Stan

Gabriel Robins (Chair)

Pradip Bose

James Cohoon

Accepted by the School of Engineering and Applied Science:

Craig H. Benson (Dean)

May 2016

Abstract

As the semiconductor industry keeps evolving at the pace predicted by Moore's Law, computer system architects are facing increasing challenges from the three major design constraints: performance, power, and reliability. Thermal constraints from a reasonable cooling cost do not scale well as technology evolves. The dwindling scaling on threshold voltage leads to a slower pace of supply voltage scaling. These two effects lead to an increasing power density in current and future technology generations. Reliability has emerged as a primary design constraint due to the smaller feature size and generally lower supply voltage for electronic devices. Transient errors caused by high-energy particle strike and voltage noises are expected to increase significantly in frequency. Performance improvement becomes more challenging for future architectures with limitations set by the power and the resilience constraints.

Integration of accelerators to create heterogeneous processors is becoming more common for both power and performance reasons. However, this adds one more dimension to the design space that is already complex due to technology variants, system organizations, application's variability, and so on. Therefore, high-level models are essential for system designers to explore the design space and make decisions in a timely manner. Additionally, the three design constraints compete with each other. For example, resilience-aware techniques, such as DMR and TMR, are expensive in terms of power and performance, low-power designs usually come with a price of lower speed. Consequently, it requires system designers to make trade-offs by considering all the three design constraints at the same time.

To address these challenges, we (1) propose an analytical modeling framework called Lumos that is capable of modeling power and performance for heterogeneous architectures with hardware accelerators. Then we (2) use Lumos to explore the design space composed of CPU cores and accelerators, revealing important scaling trend for future heterogeneous architectures. We further (3) propose a rapid modeling framework to characterize resilience across a range of applications in DSP, and image processing domains; And finally we (4) propose an integrated framework to optimize energy-efficiency by trading off design constraints of power, performance and resilience.

Acknowledgments

First, I would like to thank my advisor Prof. Kevin Skadron for guiding me through my pursue of the Ph.D. degree in the graduate school. He helped me develop my research topic, continuously improve the quality of the work, present and publish the work as an independent researcher. He also influenced me to improve my soft skills, such as professional networking, collaboration and communication with external researchers. Last, but probably the most importantly, he encouraged me to finish and helped me to build a strong mind to survive through those desperate moments on my way to the degree. Without these advising and mentoring, both academic and non-academic, I would never reach where I am right now.

A significant part of this dissertation came from my year-long internship at IBM T. J. Watson Research Center and follow-up collaborations. I would like to express my gratitude to my manager Dr. Pradip Bose and my mentor Dr. Jude A. Rivers. As renowned researchers in computer architecture, they guided me through the whole project from research definition, problem solving, implementation, and presentation to upper management. From their insightful guidance, I have gained exclusive learning experience not only in academic research, but also in the project management, patent application, etc. I would like to thank Dr. Alper Buyuktosunoglu as well, who shed innumerable amount of insightful comments on all projects I worked on at IBM. I would like to thank Dr. Ramon Bertran, Dr. Augusto Vega, Dr. Chen-yong Cher, Dr. Meeta S. Gupta for their various collaborations, and making my internship a remarkable experience. Most importantly, this dissertation would not have been possible without their help.

I would also like to thank my committee, Prof. Mircea R. Stan, Prof. James Cohoon, Prof. Gabriel Robins, and Dr. Pradip Bose, for taking time to review my proposal and dissertation as

well as for giving me many valuable comments about this work.

I am grateful for having so many friends both inside and outside the CS/ECE departments. The members and alumni of Kevin's LAVA lab have helped me in numerous ways. Especially, I thank Dr. Runjie Zhang, Dr. Shuai Che, Dr. Michael Boyer, Dr. Ke Wang, Prof. Brett H. Meyer, Jack Wadden, Deyuan Guo, Chunkun Bo, Elaheh Sadredini, for invaluable discussions on research ideas, and generous sharing of useful information of all kinds. I want to thank Xinfei Guo for his assistant on circuit simulation setup with the latest PTM, which contributed to build Lumos framework. In addition, I want to thank my fellow graduate students Yuchen Zhou, Yanqing Zhang, Wei Wang, Wei Zhang for setting examples of excellent researchers and motivating me with peer-pressure.

I want to thank my parents and in-laws for their unconditional love and support. I thank my wife Xiaohui for her endless love and support. I am grateful for her constant encouragement, advice, and accompany, which leads me through the most stressful, depressing, and even desperate times in my life. I dedicate this dissertation to all of them.

This work was supported by the SRC under GRC task 1972.001, the NSF under grants MCDA-0903471 and CNS-0916908, and Defense Advanced Research Projects Agency (DARPA), Microsystems Technology Office (MTO) under contracts no. HR0011-13-C-0022. The views expressed are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

As the semiconductor industry keeps evolving at the pace predicted by Moore's Law [53, 54], computer system architects are facing increasing challenges from the three major design constraints: performance, power, and reliability.

First of all, performance has always been a primary constraint for system architects. However, it is getting increasingly hard to improve the performance, especially the performance of a single core. Chip-multiprocessors (CMP) have been widely accepted to be promising to deliver scalable performance over generations. Nevertheless, its effectiveness has been largely reduced due to physical constraints arising from recent technology scaling trends (this will be discussed in detail later). On the other hand, hardware specialization is another promising approach to achieve better performance in an efficient way. Unfortunately, the heterogeneity among conventional cores and the specialized hardware increases the design complexity significantly, not to mention the substantially higher engineering cost associated with the customized hardware.

Secondly, power consumption has been widely accepted as another first-class constraint ever since early this century [10]. Thermal constraints from a reasonable cooling cost set the total system power, also known as the "Power Wall," which does not scale well as the technology evolves. The recent trend in technology scaling poses even more challenges on power constraints. Threshold voltage scales down slowly in current and future technology generations to keep leakage power under control. The dwindling scaling on threshold voltage leads to a slower pace of supply voltage scaling. Since the switching power scales as CV^2f, Dennard Scaling can no longer be maintained, and power density keeps increasing. Furthermore, the cooling cost and on-chip power delivery implications limit the increase of a chip's thermal design power (TDP). As Moore's Law continues to double transistor density across technology generations, total power consumption will soon exceed TDP, and if high supply voltage must be maintained, future chips will only support a small fraction of active transistors, leaving others inactive, a phenomenon referred to as "dark silicon" [26, 84].

Last but not least, reliability has been evolved to be a primary constraint for system designers, which traditionally is a serious concern only for mission-critical systems, such as unmanned aerial vehicles (UAVs), or high-end server systems, such as IBM POWER series processors. The transistor density keeps doubling every eighteen months, as Moore's Law predicts. The growing density results in a higher probability of manufacturing imperfection and raises the likelihood of a chip to fail. Additionally, technology scaling keeps delivering smaller feature size and reduced supply voltage for electronic devices. Single-event upsets (SEUs) caused by high-energy particle strike are exacerbated at lower voltages, due to the critical charge (Q_{crit}) required to cause a latch or SRAM bit flip diminishes correspondingly. Similarly, transient errors caused by voltage noise also increase in frequency under low-voltage operations. To make the situation even worse, errors are expensive to detect and recover in terms of power and performance. Redundancy-based techniques, such as dual-modular redundancy (DMR) and triple-modular redundancy (TMR), suffer from considerable overheads in system power and area. On the other hand, checkpoint-based techniques have non-negligible run-time overhead, and could potentially prevent forward progress if the soft error rate (SER) is too high. Meanwhile, extra energy consumed by recovery hurts the overall energy efficiency of the system. This leaves computer systems with worse performance as well as even more pressure on an already stringent power budget.

Integration of accelerators to create heterogeneous processors is becoming more common for both power and performance reasons. However, this adds one more dimension to the design space that is already complex due to technology variants, system organizations, applications variability, and so on. As a result, high-level models are essential for system designers to explore the design space and make decisions in a timely manner. Additionally, the three design constraints compete with each other. For example, resilience-aware techniques, such as DMR and TMR, are expensive in terms of power and performance; low-power designs are implemented at a price of lower speed. Therefore, it requires system designers to make trade-offs by considering all the three design constraints at the same time.

To address these challenges, we propose a set of frameworks to model power-efficiency, resilience, and performance for future heterogeneous multi-core systems. The proposed frameworks are flexible and easily integrated to perform holistic analysis regarding all design constraints. Our approaches consist of the following major research tasks:

1. Design and implement an analytical modeling framework that is capable of modeling both power and performance for heterogeneous architecture with hardware accelerators under different technology variants.

2. Explore the design space composed of conventional CPU cores and hardware accelerators, revealing potential trends of system designs for future heterogeneous architectures.

3. Propose an approach to analyze and evaluate accelerator-rich heterogeneous architectures exploiting an advanced optimization algorithm.

4. Design and implement a rapid resilience modeling framework based on application-level fault injections, then characterize resilience across a range of applications in DSP, and image processing domains.

5. Propose an integrated framework to optimize energy-efficiency by trading off design constraints of power and resilience in the context of real-time embedded processing.

1.1 Lumos: An Analytical Modeling Framework for Design Space Exploration on Heterogeneous Computer Architecture

Quantitative models for power and performance play an essential role in computer architecture research. As a result, researchers have proposed enormous amount of models through decades' research within the field, they vary in complexities and accuracies. Theoretically, direct measurements on a real machine are considered as the most accurate modeling approach, however, it comes with the limitation that quite a few architectural parameters can be changed. It is almost impossible to use this approach to explore innovative architectures nor to experiment with new features on existing architecture. The next level of approaches exploits architecture simulators to adopt power and performance of the target computer systems. Although simulators are popular for architectural research and development in the last couple of decades [2, 7, 10], they suffer from orders of magnitude slower simulation throughput than native execution. And it is prohibitively expensive to use simulation-based approaches to do a broad exploration in a large design space.

Analytical models based on Amdahl's Law [1] have gained popularity recently [26, 35], primarily for the interest of design space exploration of future computer architectures. These efforts in early-stage exploration are not aimed at proposing specific architectural features. Instead, their goals are trying to find important implications for architecture designs. For example, in [26], authors showed that an increasing amount of die area will be inactive, a.k.a. "dark silicon," due to the more stringent relative power budget as the manufacturing technology advancing to the next couple of generations. The observation advocated a shift of focus from multi-core scaling to more power-efficient computing units (e.g. hardware accelerators).

In Chapter 2, we propose an analytical modeling framework, called "Lumos", that is capable of modeling the power and the performance of conventional CPU cores, as well as hardware accelerators. Using "Lumos", we explore the design trade-offs among CPU cores and various accelerators. We show that despite the performance benefits from operating CPU cores at a substantially lower supply voltage, it is even more promising to include hardware accelerators as the building block in heterogeneous systems. Furthermore, in Chapter 3, we design a systematic approach to analyze accelerator-rich heterogeneous architectures with regard to domain-specific and general-purpose applications. From our extensive design space exploration, we show that reconfigurable accelerators are preferable over dedicated accelerators when overall performance across a set of applications is of particular interest.

1.2 AFI: Application-Level Fault Injection

Due to technology scaling, reliability has emerged as a first-order design constraint for mainstream computer systems. The shrinking feature size makes circuits more vulnerable to permanent and transient faults. Permanent faults are caused by manufacturing imperfection as well as the aging effect in transistors. Transient faults, also known as soft errors, are due to high-energy particle strikes [4] and voltage droops [58]. Both types of faults are important from the perspective of system reliability. However, in this dissertation, we mainly focus on transient faults since they are more common.

Fortunately, a given pair of a machine architecture and an application offers significant masking of effects in the face of spurious bit-flips. As a result, the raw soft error rate (SER) of processor chips – even though it may increase with device density and lower voltages – is at least partially offset by large degrees of masking. For a given machine, different applications or phases within an application workflow present different degrees of masking. It is also true that despite errors in a particular program output, consuming applications can often actuate actions accurately, thereby providing error-free outcome at the end. For example, an image processing and recognition algorithm can tolerate numerous pixel errors in the input data that may have been produced by an earlier image-scanning step.

To quantify the SER with regard to various applications, in Chapter 4, we present a framework based on a software-implemented fault injection (SWIFI) tool, called AFI. It features a runtime-independent triggering mechanism to enable flexible parallel experiment runs, as well as a mechanism to characterize only the region-of-interest (ROI) of an application through source code annotation. The tool is built on top of standard debugging interface and performance counters which are available in most of the modern operating systems and processors. AFI is currently implemented on AIX$^{\text{TM}}$for POWER7 processor for demonstration purpose. It can be easily ported to other platforms such as X86 on Linux. Using AFI, We characterize the transient error tolerance of a set of applications from DSP and image processing domains. Various error injection models are used to see if the relative ordering of applications gets changed while measuring resilience metrics. Additionally, we show that error tolerance is a strong function of the compiler optimization level. This provides a static optimization knob for application scientists in the quest to optimize performance/watt while meeting resilience targets. Finally, we explain changes in the degree of error tolerance by analyzing simple workload metrics like the dynamic instruction frequency mix.

1.3 Resilience-aware Optimization for Power-Efficiency with Real-Time Constraints

A powerful control parameter for power management of embedded processor systems is dynamic voltage-frequency scaling (DVFS). However, soft error rates (SERs) are known to increase sharply as the supply voltage is scaled downward [70]. Hence, in order to preserve system resilience levels, it is important to apply voltage scaling carefully, keeping in mind the varying levels of vulnerability to SER within an application's execution profile. On the other hand, overclocking or turbo-boosting (with higher voltages applied if/as necessary) to meet the real-time deadlines, comes at the cost of higher power (or current) density and temperature (as well as higher gate-oxide field stress), which results in higher hard-failure rates.

In Chapter 5, we consider a class of embedded systems that require high levels of power-performance efficiency while meeting mission-critical reliability specifications and real-time performance targets. Such systems require an energy optimization protocol that is cognizant of the variable resiliency needs and properties of the executed application. A representative example of such an embedded system of interest is a single unmanned aerial vehicle (UAV) or a swarm of such UAVs. These are typically engaged in remote sensing of ground images, for the purposes of reconnaissance, object recognition/tracking and tactical response. We propose PEARL, a novel software modeling framework that enables users to: (a) *statically* prepare application workflows for energy-optimized resilience; and (b) explore *run-time* deployment options in targeted embedded systems. The overall goal is to maximize performance per watt, while meeting real-time deadlines and resilience-related constraints (dictated by effective SER and maximum power dissipation). PEARL stands for Power Efficient And Resilient embedded processing with reaL-time constraints.

1.4 Summary

By finishing these research tasks, I contribute to a set of modeling frameworks with regard to three important design factors: power, performance, and resilience. Lumos/Lumos+ provide mechanisms for system designers to rapidly explore design points of future heterogeneous architectures, and make early decisions before spending significant efforts on a single path. AFI helps application developer to better understand applications' resilience. System designers can also benefit from resilience characterization by applying application-specific tuning algorithms. Finally, we demonstrate an optimization scenario where trade offs are made among power, performance, and resilience to achieve the best energy efficiency while still maintain the real-time deadline.

Chapter 2

Lumos: An Analytical Modeling Framework for Multiprocessors

Threshold voltage scales down more slowly in current and future technology nodes to keep leakage power under control. In order to achieve fast switching speed, it is necessary to keep transistors operating at a considerably higher voltage than their threshold voltage. Therefore, the dwindling scaling on threshold voltage leads to a slower pace of supply voltage scaling. Since the switching power scales as CV^2f, Dennard Scaling can no longer be maintained, and power density keeps increasing. Furthermore, cooling cost and on-chip power delivery implications limit the increase of a chip's thermal design power (TDP). As Moore's Law continues to double transistor density across technology nodes, total power consumption will soon exceed TDP. If high supply voltage must be maintained, future chips will only support a small fraction of active transistors, leaving others inactive, a phenomenon referred to as "dark silicon" [26, 84].

Since dark silicon poses a serious challenge for conventional multi-core scaling [26], researchers have tried different approaches to cope with dark silicon, such as "dim silicon" [36] and customized accelerators [79]. Dim silicon, unlike conventional designs working at nominal supply, aggressively lowers supply voltage close to the threshold to reduce dynamic power. The saved power can be used to activate more cores to exploit more parallelism, trading off per-core performance loss with better overall throughput [27]. However, with near-threshold supply, dim silicon designs suffer from diminishing throughput returns as the core number increases. On the other hand, customized accelerators are attracting more attention due to their orders of magnitude higher power efficiency than general-purpose processors [16, 32]. Although accelerators are promising in improving performance with less power consumption, they are built for specific applications, have limited utilization in general-purpose applications, and sacrifice die area that could be used for more general-purpose cores. Poorly-utilized die area would be very costly if there are more efficient solutions, in terms of opportunity cost. Therefore, the utilization of each incremental die area must be justified with a concomitant increase in the average selling price (ASP).

To investigate the performance potential of dim cores and accelerators, we have developed a framework called *Lumos*, which extends the well-known Amdahl's Law, and is accompanied by a statistical workload model. We investigate two types of accelerators, application-specific logic (ASIC) and RL. When we refer to RL, we generically model both fine-grained reconfigurability such as FPGA and coarse-grained such as Dyser [31]. We also assume the reconfiguration overhead is overwhelmed by sufficient utilization of each single configuration. In this chapter, we show that

- Systems with dim cores achieve up to 2x throughput over conventional CMPs, but with diminishing returns.
- Hardware accelerators, especially RL, are effective to further boost the performance of dim cores.
- Reconfigurable logic is preferable over ASICs on general-purpose applications, where kernel commonality is limited.
- A dedicated fixed logic ASIC accelerator is beneficial either when: 1) its targeted kernel has significant coverage (e.g. twice as large as the total of all other kernels), or 2) its speedup over RL is significant (e.g. 10x-50x).

This work has been published in IEEE Micro [87].

2.1 Related Work

The power issue in future technology scaling has been recognized as one of the most important design constraints by architecture designers [56, 84]. Esmaeilzadeh et al. performed a comprehensive design space exploration on future technology scaling with an analytical performance model in [26]. While primarily focusing on maximizing single-core performance, they did not consider lowering supply voltage, and concluded that future chips would inevitably suffer from a large portion of dark silicon. In [9], Borkar and Chien indicated potential benefits of near-threshold computing with aggressive voltage-scaling to improve the aggregate throughput. We evaluate near-threshold in more detail with the help of Lumos calibrated with circuit simulations. In [36], Huang et al. performed a design space exploration for future technology nodes. They recommended dim silicon and briefly mentioned the possibility of near-threshold computing. Our work exploits circuit simulation to model technology scaling and evaluates in detail the potential benefit of improving aggregate throughput by dim silicon and near-threshold computing. In short, Lumos allows design space exploration for cores running at near-threshold supply factoring in the impact of process variations.

Another set of studies has evaluated the benefit of hardware accelerators as a response to dark silicon. Chung et al. studied the performance potentials of GPU, FPGA, and ASIC in [16], regarding physical constraints of area, power, and memory bandwidth. Although a limited number of applications were studied in their work, our work corroborates that reconfigurable accelerators, such as FPGA, are more competitive than dedicated ASICs. Wu and Kim did a study across several popular benchmark suites regarding the targets to be accelerated in [91]. Their work suggested a large number of dedicated accelerators are required to achieve a moderate speedup due to the minimal functional level commonality in benchmark suites like SPEC2006. This is consistent with our observation that dedicated fixed logic accelerators are less beneficial due to limited utilization across applications in a general-purpose workload, and the importance of efficient, reconfigurable accelerators. In [81], Tseng and Brooks build an analytical model to study trade-offs between latency and throughput performance under different power budgets and workloads. However, the model lacks the support of voltage and frequency scaling, especially near threshold and the capability of hardware accelerator performance modeling, limiting the design space explorations of future heterogeneous architectures. In [96], Zidenberg et al. propose MultiAmdahl model to study the optimal allocations to each type of computational units, including conventional cores and u-cores. The MultiAmdahl model cannot support voltage scaling and near threshold effects. Therefore, its design space exploration capability is limited to heterogeneous systems under dark/dim silicon projections. Lumos takes a broader view of these issues, by providing a highly configurable tool for exploring the design space composed of conventional cores, various forms of accelerators, and low-voltage operation, with various power constraints and various workload characteristics.

2.2 Lumos Framework

The design space of heterogeneous architectures with accelerators is too huge to use extensive architectural simulation in a feasible way. To enable rapid design space explorations, we build Lumos, a framework putting together a set of analytical models for power-constrained heterogeneous many-core architectures with accelerators. Lumos extends the simple Amdahl's Law model by Hill and Marty [35] with physical scaling parameters calibrated by circuit simulations with a modified Predictive Technology Model [12]. Lumos is composed of:

1. a technology model for inter-technology scaling;
2. a model for voltage-frequency scaling;
3. a model evaluating power and performance of conventional cores;
4. a model evaluating power and performance of accelerators;
5. a model for workloads and applications.

2.2.1 Technology Modeling

Technology scaling is built based on the modified PTM, which tunes transistor models with more up-to-date parameters extracted from recent publications [12]. Two types of technology variants are investigated, a high-performance process and a low-power process. The primary difference is that low-power processes have a higher threshold, as well as a higher nominal supply voltage, than high-performance processes for every single technology node, as shown in 2.1.

Table 2.1: Nominal supply voltage (V_{nom}) and threshold voltage (V_t) for each PTM technology variants. Voltage units are volt (V).

		$45nm$	$32nm$	$22nm$	$16nm$
High	V_{nom}	1.0	0.9	0.8	0.7
Perf.	V_t	0.424	0.466	0.508	0.505
Low	V_{nom}	1.1	1.0	0.95	0.9
Power	V_t	0.622	0.647	0.707	0.710

Inter-technology scaling mainly deals with scaling ratios for the frequency, the dynamic power, and the static power. From the circuit simulation, we compare the frequency, the dynamic power, and static power at the nominal supply for each of technology nodes. We assume CPU cores will have the same scaling ratio as the circuit we simulated. As shown in Table 2.2, the scaling factors of high-performance variants are normalized to 45nm, while the scaling factors of low-power variants are normalized to their high-performance counterparts.

Table 2.2: Technology scaling for high-performance (HP) processes and low-power (LP) processes. The area scaling is assumed to be ideal so that the area of a single core shrinks by 2x per technology node.

Tech.	Freq.	Switch	Static
(nm)		Power	Power
45	1.0	1.0	1.0
32	0.95	0.49	0.31
22	0.79	0.21	0.12
16	0.66	0.09	0.13

(a) HP, normalized to 45nm

Tech.	Freq.	Switch	Static
(nm)		Power	Power
45	0.27	0.30	0.0084
32	0.28	0.32	0.042
22	0.26	0.34	0.23
22	0.26	0.40	0.49

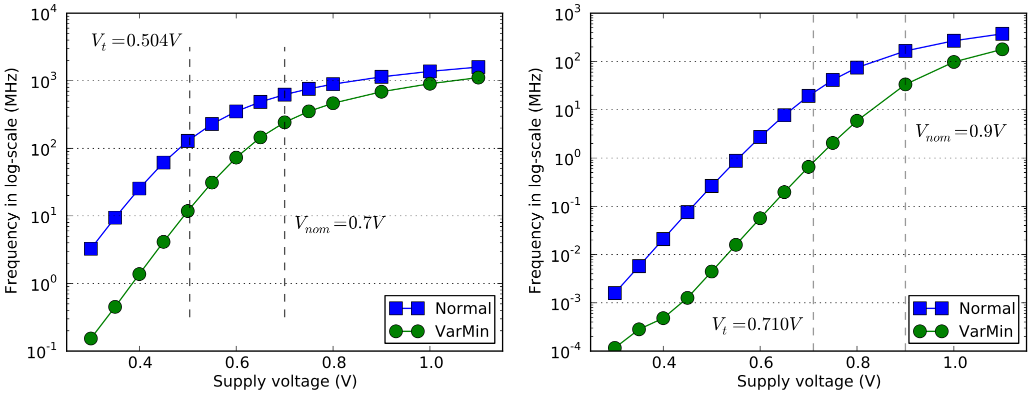
(b) LP, normalized to HP counyerparts

2.2.2 Frequency Modeling

Voltage-to-frequency scaling is modeled by interpolating empirical results from circuit simulations. We use a 32-bits ripple carry adder (RCA) to emulate the core frequency changes subject to various supply voltages. The adder is synthesized with a standard-cell library [38] at 45nm. Sizes of transistors in the post-synthesized net-list are scaled to other technology nodes. These circuit netlists are then simulated using a modified version of Predictive Technology Model (PTM) for the bulk devices. Note that Lumos is not limited to the bulk devices, but can be extended to other technologies by applying the same RCA characterization with the target technology library. More specifically, Lumos can be extended with minimal efforts to support FinFET devices by using the latest PTM from [75].

The switching speed of transistors scales exponentially to the threshold voltage when it is operating at near-threshold voltage; therefore, the working frequency of a circuit in the near-threshold region is extremely sensitive to the threshold voltage. To investigate the fluctuation of frequency subject to process variations, the test circuit has been simulated with Monte-Carlo analysis, assuming a standard Gaussian distribution with 3-σ on the threshold voltage of transistors.

The results for 16nm are shown in Figure 2.1. For the high-performance process plotted in Figure 2.1a, the maximum performance penalty due to process variations is less than 10% when the circuit is operated at higher voltages around $1V$. However, when supply voltage approaches the threshold, frequency penalty increases significantly to more than 90%, which means an order of magnitude slowdown on frequency. A similar trend is observed in Figure 2.1b for the low-power process, though the low-power process suffers from a larger percentage of frequency loss than the high-performance process even at a higher supply voltage, e.g. 1V.



(a) High-performance with $V_t = 0.504V$.

(b) Low-power with $V_t = 0.710V$.

Figure 2.1: Voltage-frequency scaling characterization on a 32-bits ripple carry adder (RCA) simulated at 16nm, subjecting to process variation (VarMin). V_t is the threshold voltage, and V_{nom} is the nominal supply voltage.

2.2.3 Core Modeling

2.2.3.1 Power

Core power consumption is modeled as two major sources, dynamic power due to transistor switching and static power due to leakage. Therefore, per-core power is given by

$$P_{\text{total}} = P_{\text{dynamic}} + P_{\text{leakage}} \tag{2.1}$$

Generally speaking, dynamic power is given by

$$P_{\text{dynamic}} = \alpha \cdot C_{eff} \cdot V_{dd}^2 \cdot f \tag{2.2}$$

where α is the activity factor, C_{eff} is the effective capacitance, V_{dd} is supply voltage, and f is the core running frequency. We assume a constant activity factor and effective capacitance when the core frequency is scaled with various supply voltages. Therefore, dynamic power changes quadratically to supply voltage and linearly to frequency. According to [11], the static power of a system is given by

$$P_{\text{leakage}} = V_{dd} \cdot N \cdot k_{design} \cdot \hat{I}_{leak} \tag{2.3}$$

where V_{dd} is supply voltage, N is the number of transistors, k_{design} is a device-specific constant, and $\hat{I}_{leak}$ is the normalized per-transistor leakage current. We assume the normalized per-transistor leakage current is proportional to the leakage current of a single transistor, which , according to [64], is given by

$$I_{leak} = I_0 \cdot 10^{\frac{V_{gs}-V_t+\eta V_{ds}}{S}} \cdot \left(1 - e^{-\frac{V_{ds}}{V_{th}}}\right) \tag{2.4}$$

where V_t is the threshold voltage, η is the drain-induced barrier lowering factor (DIBL), V_{th} is the thermal voltage, and n is a process-dependent constant. The thermal voltage at room temperature is around $28mV$, which is far less than the supply voltage of interest in this project, therefore, $e^{-V_{ds}/V_t} \approx 0$. Because the transistor is at its static state when considering the static leakage power, V_{gs} and V_{ds} is roughly proportional to the supply voltage. As a result, the above equation can be reduced to

$$\hat{I}_{leak} \propto 10^{\frac{V_{dd}}{\hat{S}_{leak}}} \tag{2.5}$$

where $\hat{S}_{leak}$ is the aggregate scaling coefficient derived from fittings to the simulated results.

2.2.3.2 Performance

Lumos uses aggregate throughput under physical constraints as the primary performance metric. For the aggregate throughput performance (pf_{s}), we model it with Amdahl's Law as shown in Equation 2.6

$$\text{Speedup} = \frac{1}{\frac{1-\rho}{S_{\text{serial}}} + \frac{\rho}{n \cdot S_{\text{parallel}}}} \tag{2.6}$$

where ρ is the parallel ratio of the studied workload, S_{serial} is the serial part speedup over a baseline core, S_{parallel} is the per-core speedup when the system works in parallel mode, and n is the number of active cores running in parallel. For S_{serial}, only one core is active. In this case, the core is boosted to 1.3x nominal supply to achieve the best single core performance, denoted as $\hat{pf}_{\text{c}}$. When the system is in parallel mode, both n and S_{parallel} are determined by supply voltage. First, Lumos picks the optimal frequency with a given supply voltage; Second, it calculates per-core power consumption including both the switching power and the leakage power according to the frequency and the supply. Finally, the number of active cores is the minimum restricted by the overall power and area budgets, which is given by

$$n(v) = \min(\frac{P}{p(v)}, \frac{A}{a}) \tag{2.7}$$

where $p(v)$ is the per-core power as a function of supply voltage, P and A are system budgets of power and area, respectively. As a result, Equation 2.6 can be rewritten as

$$\text{Speedup} = 1 \bigg/ \frac{1-\rho}{\frac{\hat{pf}_{\text{c}}}{pf_0}} + \frac{\rho}{n(v) \cdot \frac{pf(v)}{pf_0}} \tag{2.8}$$

where pf_0 is the performance of a single baseline core, $pf(v)$ is the per-core performance as a function of supply voltage,

When considering process variation, performance loss of a single core leads to the poor throughput of a symmetric many-core system, in which the slowest core dictates the overall performance of the whole system. Adaptive Body Bias (ABB) is not considered here, but is important future work. However, the worst-case variation we used here sets an upper bound on the benefits of ABB. As a result, the parallel performance of a system is confined to the core with the worst performance, denoted as $pf(v)_{min}$. As for the power, we assume no fine-grained per-core level power management mechanisms; therefore, all cores contribute to the total power consumption through the whole period that the system is in parallel mode. We use the total mean power ($\overline{p}$) to estimate the per-core power, so Equation 2.7 is rewritten to

$$n(v) = \min(\frac{P}{\overline{p}}, \frac{A}{a}) \tag{2.9}$$

When considering process variations, the overall speedup in Equation 2.8 is rewritten to

$$\text{Speedup} = 1 \Bigg/ \frac{1-\rho}{\frac{\hat{p}f_c}{pf_0}} + \frac{\rho}{n(v) \cdot \frac{pf(v)_{min}}{pf_0}} \tag{2.10}$$

2.2.3.3 Un-core Components

Un-core components, such as network-on-chip (NoC) and memory controllers (MCs), have important implications on power and performance of a modern processor. Detailed models ([40, 67]) are too fine-grained to achieve fast evaluation, and unsuitable for exploration tasks of large design space explorations that Lumos have been targeted. Alternatively, Lumos takes advantage of implicit approaches to model the power and performance impacts of un-core components. Lumos reserves 50% of both the total thermal design power (TDP) and the die area for un-core components. The performance impact on the multi-core scaling has been lumped into the ρ parameter in Equation 2.6.

2.2.3.4 Baseline

A Niagara2-like in-order core is chosen as a single-core baseline design. The characteristics of this baseline core at 45nm are obtained from McPAT [48] and summarized in Table 2.3.

Table 2.3: Characteristics of a single Niagara2-like in-order core at 45nm, obtained from McPAT [48].

Frequency (GHz)	Dynamic Power (W)	Leakage Power (W)	Area (mm^2)
4.20	6.14	1.06	7.65

2.2.4 Accelerator Modeling

We model the speedup and the power consumption of RL and customized ASIC for a given kernel by u-core parameters (η,ϕ). "u-core" is a term proposed by Chung et. al. [16] to represent any single computational unit that is not a conventional CPU core. Examples of "u-core" include dedicated hardware accelerators and reconfigurable accelerators. η[1] is the power relative to a single, basic in-order core, and ϕ is the performance relative to the same baseline in-order core. We assume that u-cores are only used to accelerate kernels that are ideally parallelized. Therefore, we model the relative performance of a u-core proportional to its area:

$$\text{Power}_{ucore} = \eta \cdot \frac{A_{ucore}}{A_0}, \quad \text{Perf}_{ucore} = \phi \cdot \frac{A_{ucore}}{A_0}$$

where A_{ucore} is the area of a u-core, and A_0 is the area of a single baseline in-order core.

2.2.5 Workload Modeling

Figure 2.2 illustrates the workload model, where a workload is composed of a pool of applications. Each single application is divided into two parts, serial and parallel. Part of an application can be also partitioned into several computing kernels. These kernels can be accelerated by various computing units, such as multi-core, possibly dim CPU cores, an RL accelerator, or a dedicated ASIC accelerator. We add two more parameters to model the relationship between applications and kernels:

Presence (λ) A binary value to indicate whether or not a kernel present in an application.

Coverage (ε) The time consumption in percentage for a kernel when the whole application is running with a single base line core.

To model these two parameters, We have profiled the PARSEC benchmark suite [6] using Valgrind [59]. We use native input set for all applications except for x264, due to that Valgrind failed

[1] In the original paper of [16], μ is used to denote relative performance of u-cores. However, μ is commonly used as the mean in statistics. Therefore, we use η as an alternative to avoid confusions.

to capture source annotations for x264. We extract top kernels for every single application. Their presence and coverage, which are plotted in Figure 2.3, show trends:

- Kernels, such as library calls, have a larger probability of presence, but lower coverage in a single application;
- Kernels, such as application-specific routines, have very limited presence, but higher coverage once they present;
- The majority of kernels are application-specific.

In addition to general-purpose benchmark suites like PARSEC, we have observed a similar trend on domain-specific benchmarks suite such as SD-VBS [83] and Minebench [57]. Based on these observations, we propose a statistical approach to model kernels' speedup (η), presence (λ),

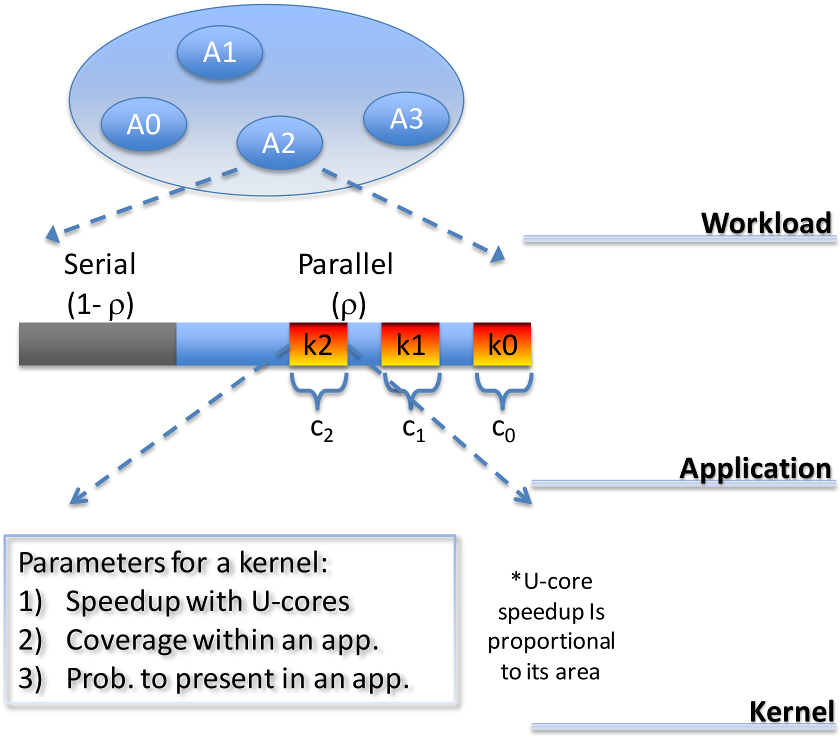


Figure 2.2: A workload is composed of applications. An application may have several computing kernels, which can be accelerated by many-core parallelization, RL, and ASIC.

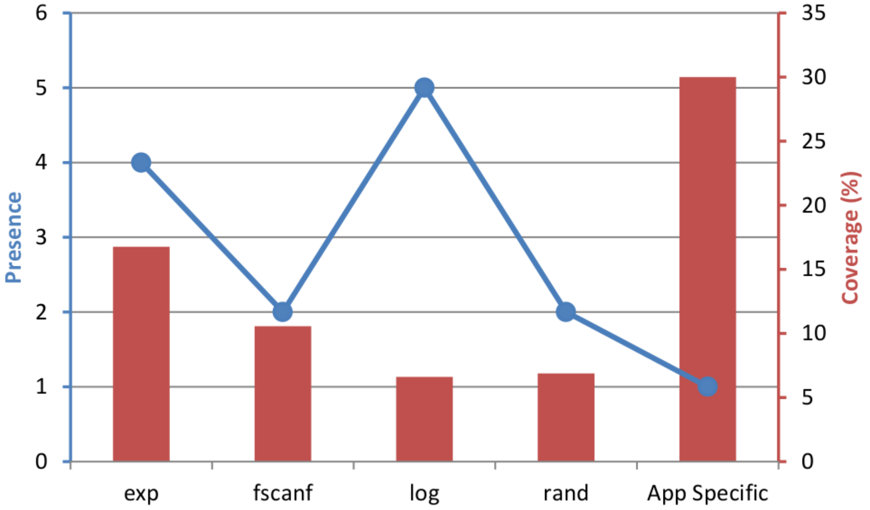


Figure 2.3: Kernels coverage and presence. Kernels are extracted from PARSEC applications, excluding x264. The coverage percentages of *exp*, *fscanf*, *log*, and *rand* are averaged across their all occurrences. The coverage percentage of the "app specific" is the average of all application-specific kernels.

and coverage (ε), which is shown in Equation 2.11.

$$\begin{cases} \eta & \sim \mathcal{N}(\mu_s, \sigma_s^2) \\ \lambda & = B(1, p) \qquad \text{where } p = PDF(\eta) \\ \varepsilon & \sim \mathcal{N}(\mu_c, \sigma_c^2) \end{cases} \tag{2.11}$$

Lumos samples the speedup (η) of kernels from a normal distribution. Speedup values of common kernels (e.g. library calls) are close to the mean of the distribution. On the other hand, speedup values of application-specific kernels are sampled from both tails of the distribution. The left tail represents kernels that are hard to accelerate, such as control-intensive ones. The right tail denotes kernels that are highly efficient on accelerators, such as compute-intensive streaming ones. Lumos samples the presence (λ) of these kernels from a Bernoulli distribution where the distribution parameter equals to $PDF(\eta)$. Lumos then samples the coverage (ε) from an independent normal distribution. We justify coverage parameters using values extracted from PARSEC. We extract speedup values of RL accelerators by normalizing reported data from recent publications in the RL research community. Finally, we assume a fixed performance scaling ratio of 5x between an ASIC accelerator and its counterpart RL implication. Table 2.4 summarizes values of all parameters used in Lumos's workload model.

Table 2.4: Summary of statistical parameters for kernel modeling

μ_s RL	μ_s ASIC	σ_s RL	σ_s ASIC	μ_c	σ_c
40x	200x	10	50	40%	10%

Lumos exploits a priority-based approach to map kernels to accelerators and conventional cores. Dedicated ASIC accelerators are considered to have the highest priority and followed by reconfigurable accelerators. If neither types of accelerators are available, a kernel will be mapped to conventional CPU cores.

2.2.6 System Configuration

Lumos models the basic system as a symmetric multi-core system composed of in-order cores. We assume the power and the area of the last-level cache (LLC) remains a constant ratio to the whole system. According to [48], we take the assumption that un-core components attribute to 50% of both the total thermal design power (TDP) and the die area. The bandwidth of the memory subsystem is assumed to be sufficient for future technology nodes by applying advanced techniques such as 3D-stacking, optical connections, multi-chip modules, etc. In the rest of this section, the power and the area of a system only refer to core components.

We have modeled three systems with different configurations of power and area by extracting data from commodity processors fabricated in 45nm. They include a small system representing desktop level processors [18], a medium system for normal server processors [19], and a large system that represents the most aggressive server throughput processors [20]. One more parameter is introduced as the power density of a system, reflecting the maximum power allowed for a unit area. It is calculated by dividing the area from the TDP. Table 2.5 summarizes detailed parameters of system configurations for Lumos.

Within an application, we assume that each kernel takes a considerable amount of computation

Table 2.5: System configurations. All numbers are for core components, assumed to be 50% of the total TDP and the die area.

System Type	Area (mm^2)	TDP (W)	P. Density (W/mm^2)[†]	# of Cores[*]
small	107	33	0.308	14
medium	130	65	0.5	17
large	200	120	0.6	26

[*] Computed based on the single core area at 45nm from Table 2.3.
[†] P. density is defined as maximum power allowed for a unit area.

time for a single run. Therefore, we can ignore various overhead in reconfiguration, as well as setting up the execution context. We leave the detailed overhead modeling and the modeling of transient kernel behaviors as future work.

2.2.7 Discussions

Instead of detailed cycle-accurate simulation, Lumos takes an analytical approach with abstractions to model heterogeneous computer architectures composed of cores and accelerators. Due to its fast evaluation time, Lumos is able to explore a large design space that is prohibitively expensive for simulation-based approaches. However, the drawback is that it is not as accurate as simulation-based approaches for a specific design. Therefore, the best use case of Lumos is to explore design implications at early stages or provision future designs subject to the technology scaling. Be complementary to each other, Lumos can help to prune down the design space for detailed simulation, while characterizations from simulation-based approaches benefit Lumos with better modeling accuracy.

2.2.8 Lumos Release

Lumos is composed of a set of Python scripts that read in technology-specific values described in previous sections, as well as results from circuit simulations. Typically, Lumos needs a workload description as its input for analysis. This description is encoded in XML format, enumerating all applications that compose the workload. Each application is described by a parallel fraction, as well as possible kernels with their coverage in percentage. Each kernel is described by u-core parameters described in section 2.2.5. We have pre-compiled a couple of descriptions for kernels and workloads following distributions as described in section 2.2.5. Lumos provides APIs to generate these XML descriptions by user-specified distribution parameters. Lumos requires the user to specify the physical constraints of a system, such as thermal design power (TDP) and area, then define systems by explicitly allocating area resources to any accelerators. Lumos allocates the rest of area to conventional cores to form a heterogeneous many-core system. When all these are ready, Lumos evaluates workload performance of the defined system or brute-force search system configurations to find the one with the best TDP-constrained performance. More details on its usage can be found at `http://liangwang.github.com/lumos/tree/v0.1`. Lumos is released under a BSD open-source license.

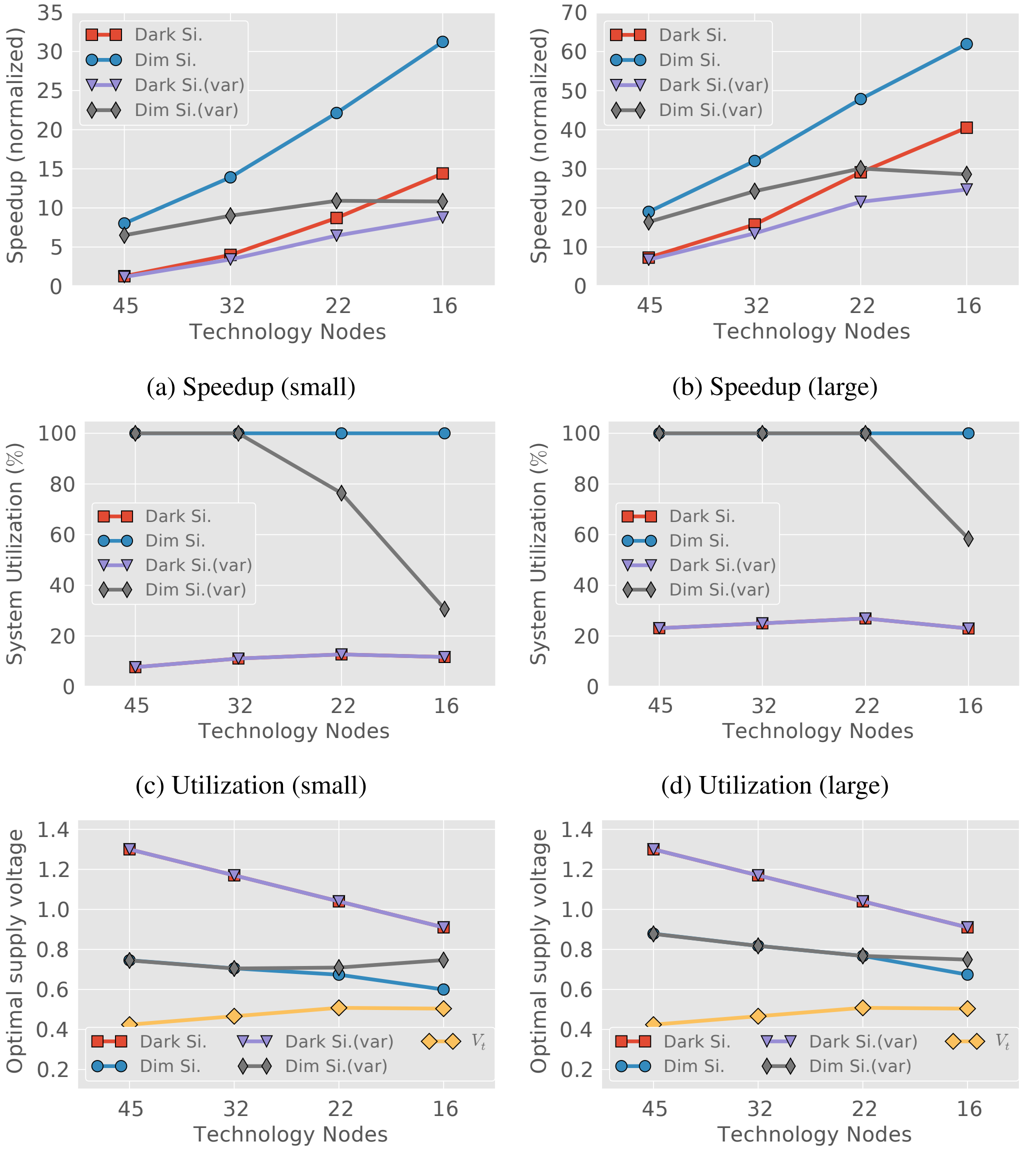
2.3 Design Space Exploration

2.3.1 Effectiveness of Dim Silicon with Near-threshold Operation

Unlike dark silicon, which we define as maximizing the frequency of cores to improve the overall throughput at the expense of potentially turning some off, dim silicon aims to maximize chip utilization to improve overall throughput by exploiting more parallelism. Dark silicon systems apply the highest supply voltage, which is assumed to be 1.3x the nominal supply voltage in Lumos, to cores in parallel mode; while dim silicon systems scale down supply and pick the optimal voltage according to the overall throughput. Figure 2.4 shows the comparisons between dark silicon (dark Si.) and dim silicon (dim Si.). .

For performance, dim silicon beats dark silicon with up to 2X throughput improvement at 16nm. When variation comes into play, however, both systems experience throughput loss compared to non-variation cases respectively. Dim silicon systems suffer even more compared to dark silicon, revealing more vulnerability of dim silicon to process variations. Dim silicon can utilize more cores to achieve high utilization, up to 100% for the small system and the large system. Similarly, utilization of dim silicon systems is sensitive to the process variation. Being aware of variation, the small system with dim silicon sees the utilization drop to as low as 20%, quite close to the utilization obtained by dark silicon at the same technology node.

Although dim cores manage to deliver higher throughput, they suffer from diminishing returns



(a) Speedup (small)

(b) Speedup (large)

(c) Utilization (small)

(d) Utilization (large)

(e) Supply voltage (small)

(f) Supply voltage (large)

Figure 2.4: Dark silicon vs. dim Silicon, regarding the throughput based speedup, die utilization, and the supply voltage with optimal throughput. Two system budgets (see Table 2.5) are compared: a small-budget system plotted on the left, and a large-budget system plotted on the right.

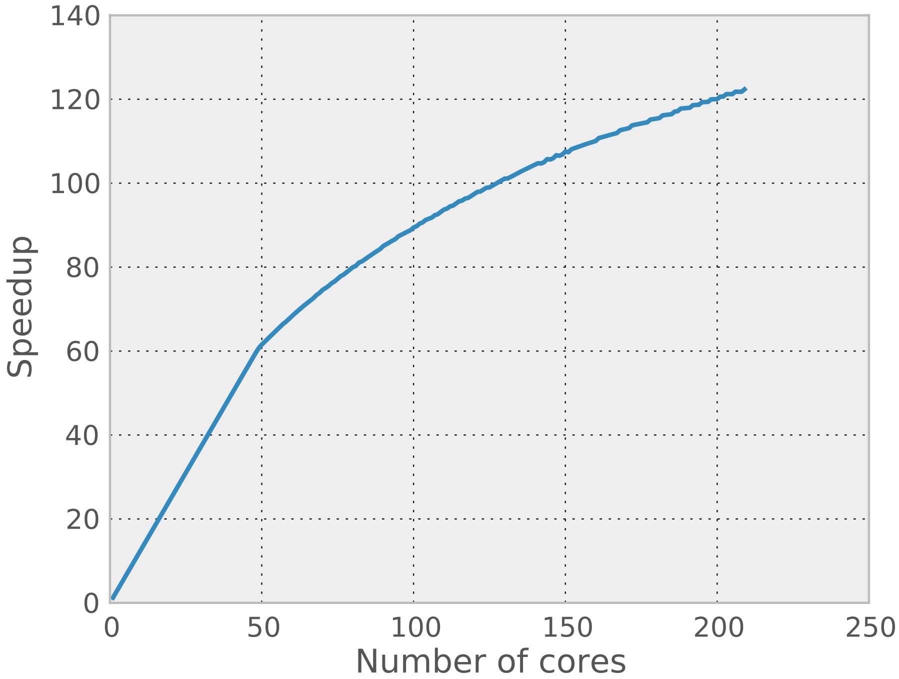
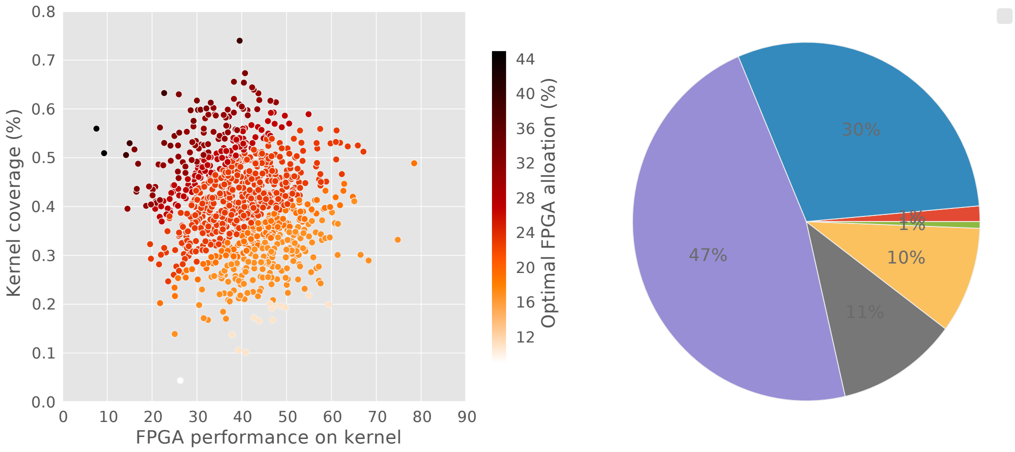


Figure 2.5: The overall throughput-based speedup of in-order (IO) cores at 16nm with high-performance (HP) process. The system is configured with large budget of 200 mm^2 in area and 120 W in TDP. Diminishing returns are observed starting around 50 active cores.

as the number of active cores increases. As shown in Figure 2.5, a system with the large budget starts to experience diminishing performance gains when the number of active cores passes over 50. When more than 50 cores are active at the same time, the system has to lower its supply voltage to stay within the power budget. Therefore, the lower per-core frequency of dim cores compromises the performance improvement from more cores. As a result, it is not cost-effective to lower supply voltage aggressively to reach the optimal throughput. The limited performance improvement from dim cores provides an opportunity to allocate some of the die area to more efficient customized hardware, such as RL, or ASICs.

2.3.2 Dim Silicon with Reconfigurable Logic

In this section, we explore the design space that includes conventional CPU cores and accelerators built on reconfigurable logic. Our goal is to find the optimal area allocation for RL that achieves the highest overall throughput on a given application. We start by looking at the scenario that every application in the workload has only a single kernel with a considerable amount of coverage, e.g. an application-specific kernel. To do this, we generate a pool of kernels from a population of distribution described in Table 2.4, as well as applications associated with each of generated kernels. To search for the optimal RL allocation, we adopt a brute-force approach by sweeping RL allocations with a step size of 1% of the total area budget. As shown in Figure 2.6, despite various kernel characteristics of each application, a majority of applications flavor RL allocations that are less than 30% (Figure 2.6a and 2.6b). This is because RL manages to accelerate the kernel significantly even with a small area allocation. As a result, the application performance is limited by the performance that dim silicon delivers on the rest part of the application, according to Amdahl's Law.



(a) Optimal area allocations on RL

(b) Distributions of optimal RL allocations

Figure 2.6: Optimal area allocations on reconfigurable logic (RL) accelerators. Derived from exhaustive search, optimal RL allocations for each application are plotted in 2.6a. In this plot, each of dots represents an application that has been characterized by the coverage of the kernel (Y-axis) and the kernel's u-core performance parameter (X-axis). The colors of dots indicate the optimal RL allocation in percentage for the given application, the darker the color the higher the optimal allocation. On the other hand, the distribution of optimal allocations is illustrated by a pie-chart in 2.6b. Percentage numbers within the pie-chart suggest the distribution of each category, while the percentage numbers around the pie-chart indicate the range of optimal RL allocations within each category.

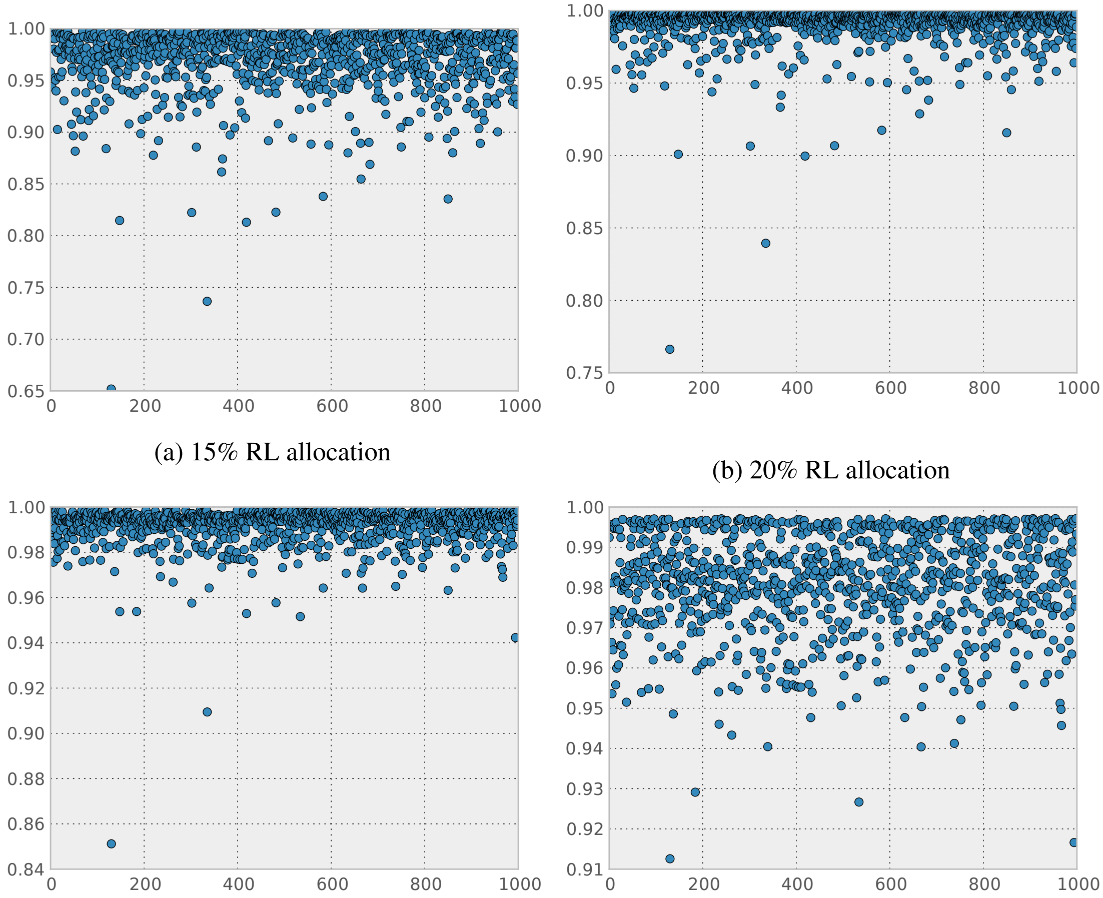
Considering a system with certain die area dedicated to RL, some applications may suffer from performance penalties when the allocation is not large enough. We compare the performance of a system to the best speedup ever achieved by varying the area allocation of RL. Figures in 2.7 plot the normalized speedup of systems with RL allocation of 15%, 20%, 25%, and 30%, respectively. Systems with smaller RL allocations, such as 15%, 20%, and 25%, have close-to-optimal performance for a majority of applications, with a worst case performance loss of 35% with 15% allocation. On the other hand, the system with 30% RL allocation only experiences less than 5% penalties for most applications, and less than 10% for the worst-case. This indicates that the speedup is somewhat insensitive to RL allocation within a reasonable range.

We vary statistical parameters of speedup and coverage, with five other alternatives listed in Table 2.6. The sensitivity study on these alternatives is summarized in Figure 2.8. As the coverage drops down from high to low profile, the optimal area allocation drops down as well. This is because the lower the coverage of a single kernel, the less significant the kernel speedup at any allocation. As the RL speedup goes from fast to slow, the optimal area allocation goes up. This is because the less the speedup of the RL on kernels, the more area it requires to achieve a comparable performance.

Table 2.6: Parameters for alternative distributions of speedup and coverage.

	Speedup			Coverage	
	Fast	Medium	Slow	High	Low
μ	80	40	20	0.4	0.2
σ	20	10	5	0.1	0.05

Then we take a look at a more realistic scenario, where applications follow characteristics defined by Equation 2.11. We choose ten synthetic kernels with speedup performance sampled evenly from $\mu - 3\sigma$ through $\mu + 3\sigma$, representing various kernel performance and kernel's probabilities of presence in a specific application. We refer to these kernels with number 0 through 9. The accelerator speedup increases from kernel 0 through kernel 9, while the probability of presence peaks at kernel 4 and 5 and bottoms at kernel 0 and 9 (i.e. normal distribution). We draw a sample popula-



(a) 15% RL allocation

(b) 20% RL allocation

(c) 25% RL allocation

(d) 30% RL allocation

Figure 2.7: The performance of systems with reconfigurable logic accelerators. For each application, we plot the performance of the given system relative to the system of the optimal RL allocation for the application. Four systems with alternative RL allocations are shown at 15% in Figure 2.7a, 20% in Figure 2.7b, 25% in Figure 2.7c, and 30% in Figure 2.7d. X-axis is the index number of an application within the workload. Y-axis is the relative performance achieved by the given system configuration relative to the performance achieved by the system with the optimal RL allocation. Note that each of dots is normalized to different baseline since the optimal RL allocation is application-specific (see Figure 2.6).

tion of 500 applications. For each application, the sum of all kernels' coverage follows the given coverage distribution. We use the mean of speedups on all applications as the performance metric. We do a brute-force search to find the optimal RL allocation.

The result suggests an optimal RL allocation around 20%, in Figure 2.9. A larger allocation on RL limits the number of available dim cores to accelerate the non-kernel parts of the application.

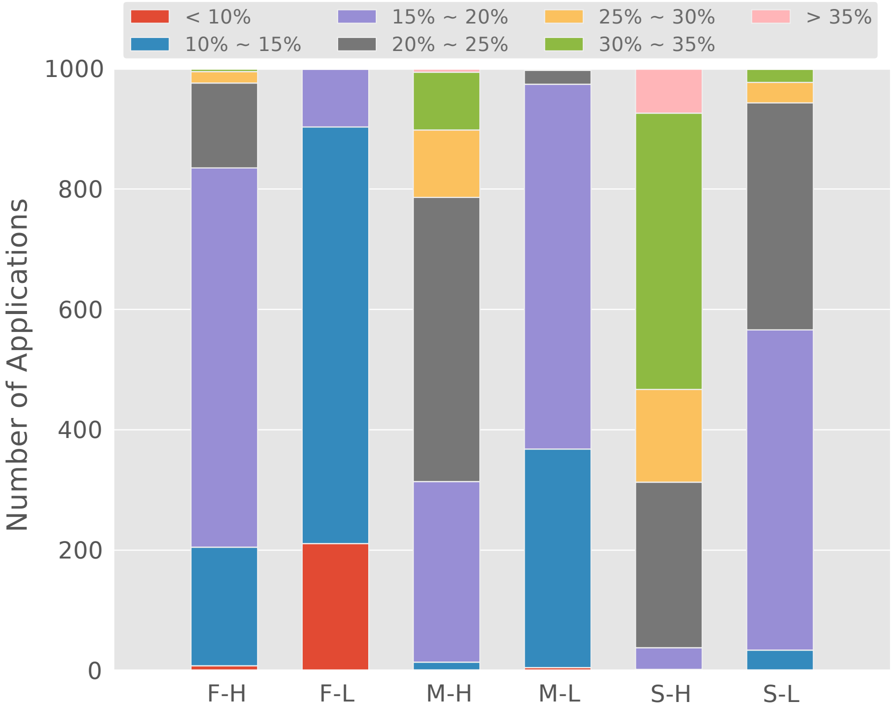


Figure 2.8: Optimal RL allocations with alternative distributions for speedup and coverage. Distributions for speedup cover three scenarios from slow to fast, while distributions for coverage cover two scenarios from low to high. Parameters for these distributions are detailed in Table 2.6.

Therefore, it delivers worse overall performance according to Amdahl's Law. For net speedup comparison of RL vs. dim silicon, see Figures 2.11-2.12 in following sections.

In summary, across a range of kernel characteristics, a relatively small area allocation on RL, e.g. 20%-30%, is good enough to achieve the optimal throughput that is almost 3x larger than a non-RL system of only dim cores.

2.3.3 Dim Silicon with ASICs

We use the same synthesized workload as described in the last subsection to study the performance implications of systems equipped with various fixed logic ASIC accelerators. Although specialized accelerators can achieve tremendous speedup for application-specific kernels, limited overall coverage of its targeted kernel within a general-purpose workload leads to limited overall performance benefit for the workload as a whole. However, accelerators for library call kernels tend to

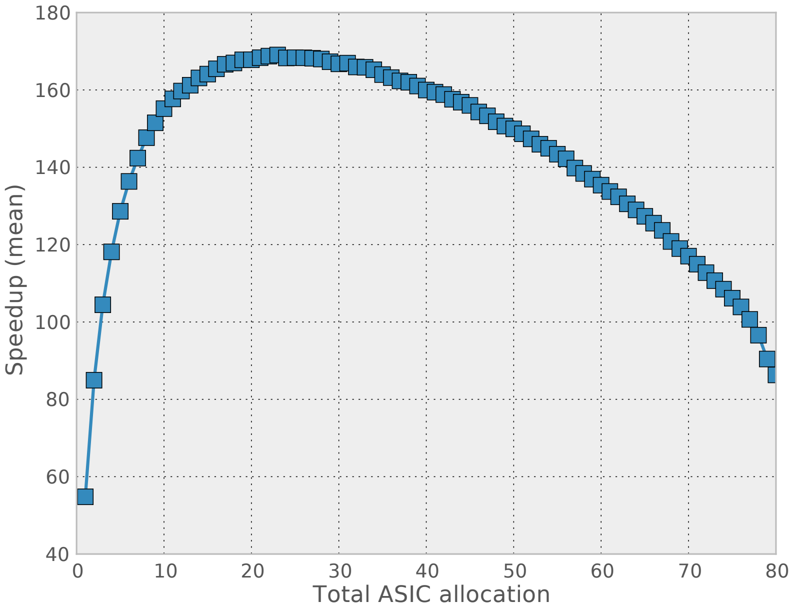
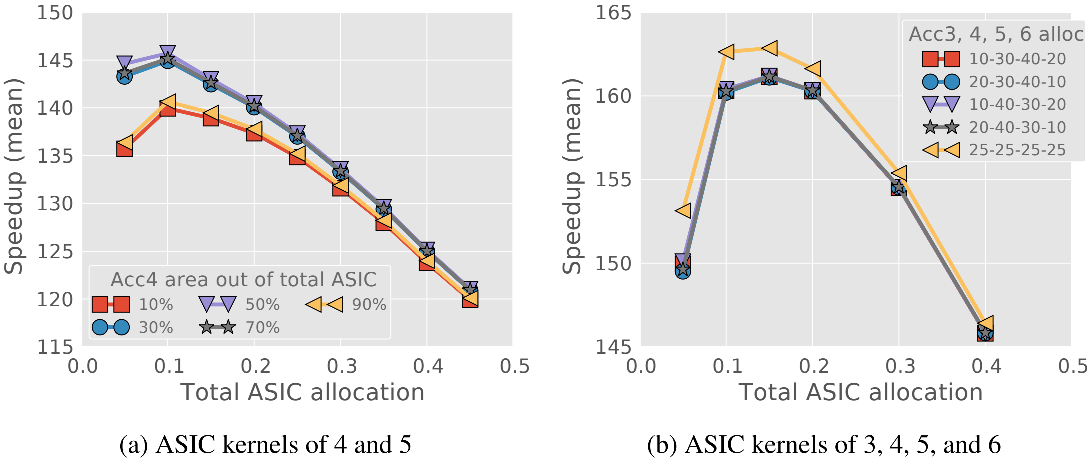


Figure 2.9: Speedup of various RL allocations. Optimal performance achieved at around 20% RL allocation.

be more beneficial in the overall speedup of the workload due to their higher overall coverage. We plot potential allocations of two, three, and four accelerators for library call kernels in Figure 2.10. As the number of accelerators increases, the best achievable performance increases as well, from



(a) ASIC kernels of 4 and 5

(b) ASIC kernels of 3, 4, 5, and 6

Figure 2.10: Speedup of a system composed of dim silicon and ASICs. We show configurations with tow accelerators 2.10a and four accelerators 2.10b. The X-axis is the total allocation to ASIC kernels, relative to the die area budget. The legend labels show the allocation of a kernel relative to the total ASIC allocation.

140 to 155. With a given number of accelerators, the system achieves the best performance when area allocations to each accelerator are evenly distributed. The total area allocation to all hardware accelerators is limited to less than 20%.

2.3.4 Dim Silicon with Accelerators(RL and ASIC) on General-Purpose Workload

In the previous two subsections, we have shown the benefits of accompanying conventional but dim cores with RL and ASIC accelerators, respectively. Both of them exhibit performance improvement over a baseline system composed of dim cores only. However, it is not necessary to achieve a better performance by combining both types of U-cores. As shown in Figure 2.11, the best performance is achieved with RL-only system organization. This counter-intuitive result comes from two reasons:

- In Lumos, the accelerator performance scales proportionally to the area it is allocated. In addition, we have a conservative assumption on the performance ration between ASIC and its

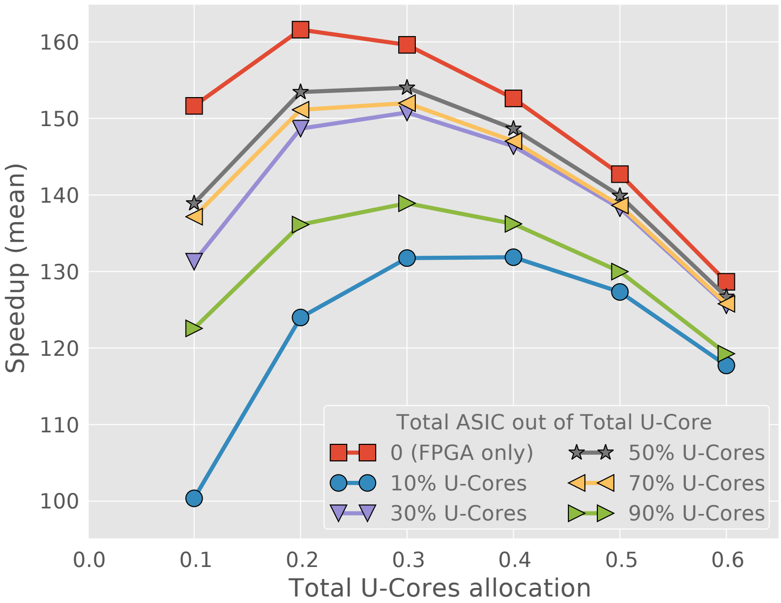


Figure 2.11: Speedup of a system composed of dim silicon, RL, and ASICs. The X-axis is the total allocation to U-cores, including both RL and ASIC accelerators, relative to the die area budget. Legend labels indicate the total allocation of ASIC accelerators, relative to the total U-cores allocation. We assume an evenly distributed allocation among ASIC accelerators, since it shows the best performance in the previous analysis.

counterpart RL implementation.(see Section 2.3.6 for the impact of alternative performance ratios). With these assumptions, RL will be more powerful than the ASIC implementation as long as the RL allocation is adequately larger than the ASIC. Moreover, a large RL allocation is justified by the high utilization achievable across multiple kernels. Consequently, the system ends up with an RL-only configuration.

- With a general-purpose workload, the average coverage of a kernel is small, due to either small coverage among applications (library-call kernels) or rare presence (application-specific kernels). This exaggerates the cost of a single ASIC accelerator that is only helpful for a specific kernel. As a result, the RL implementation is more favorable due to its versatility across kernels.

2.3.5 Benefit of ASIC Accelerators

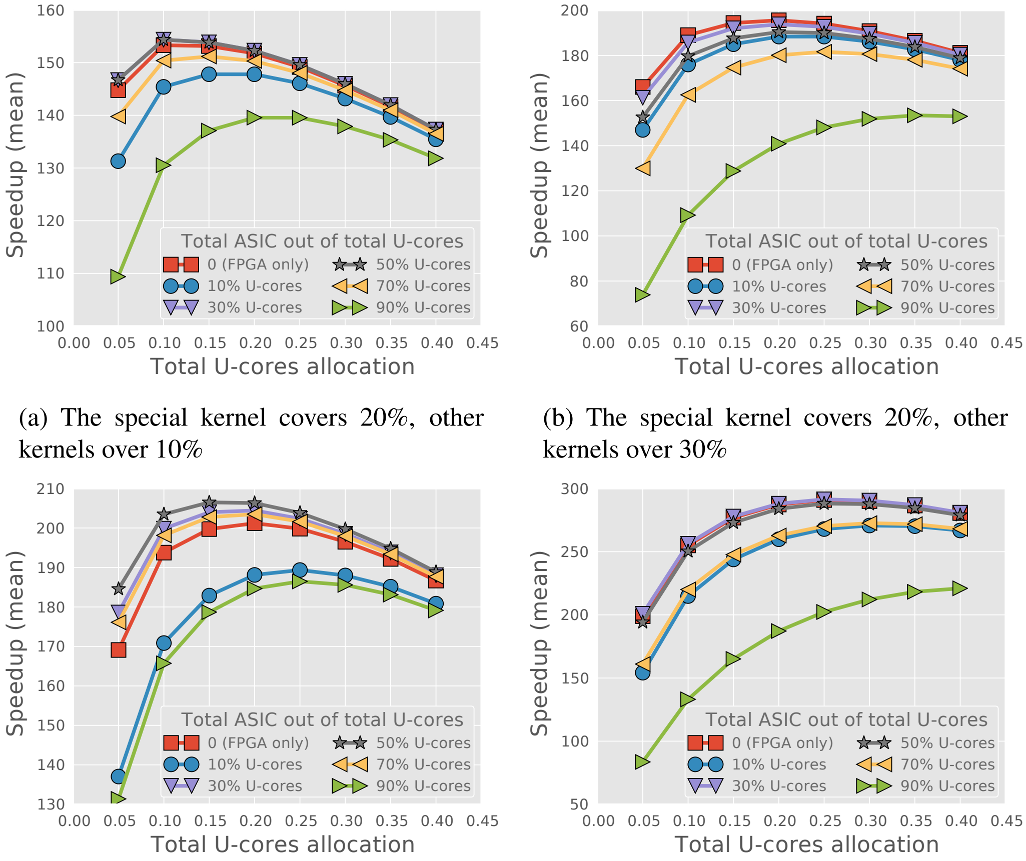
Although RL works better with general-purpose workloads, an ASIC accelerator becomes beneficial when its targeted kernel is common enough across applications in a workload. To capture this scenario, we add one special kernel to the set of ten kernels we have used in previous analyses. The coverage of the special kernel is fixed across all applications, as well as the total coverage of all other kernels. We generate workloads by varying the coverage of the special kernel and all other kernels.

Figure 2.12 shows results of 10% and 30% coverage of all other kernels. One of the most significant observations is that the dedicated ASIC accelerator is not beneficial at all until its targeted kernel covers more than the total of all other kernels within an application, e.g. in the case where the special kernel has an average coverage of 40% while others cover 10% on average (Figure 2.12c). This observation is consistent with commercial MPSoC designs such as TI's OMAP4470 [37], which has dedicated accelerators for only image processing, video encoding/decoding, graphics rendering, since these functions are expected to be quite common in the target mobile device workloads.

2.3.6 Sensitivity of ASIC Performance Ratio

Lumos assumes that a fixed logic accelerator provides 5x better performance than the RL accelerator with the same size. This assumption could be quite conservative, especially when it is hard to amortize the reconfiguration overhead over RL's running time of a specific kernel. To study the sensitivity of the ASIC performance ratio, we investigate alternative performance ratios of 10x and 50x using a similar workload setup in the previous subsection.

In Figure 2.13a, the coverage of the special kernel is fixed at 20%, while the total coverage



(a) The special kernel covers 20%, other kernels over 10%

(b) The special kernel covers 20%, other kernels over 30%

(c) The special kernel covers 40%, other kernels over 10%

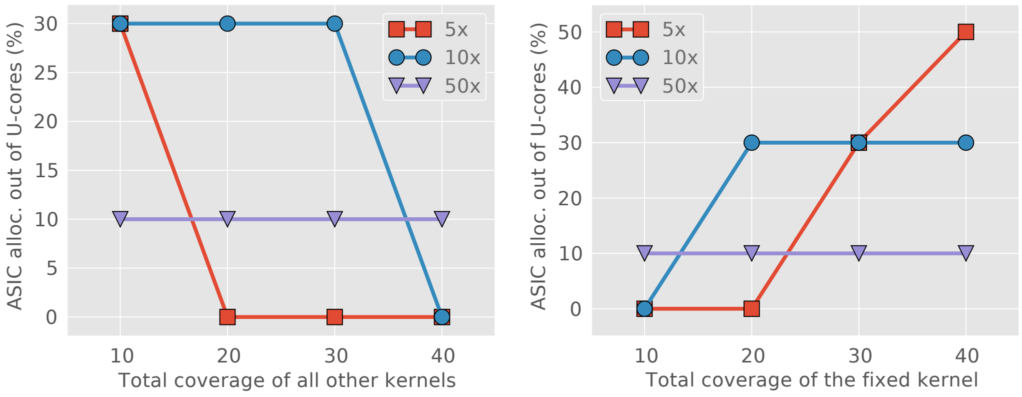
(d) The special kernel covers 40%, other kernels over 30%

Figure 2.12: To study the benefit of a dedicated ASIC accelerator, we add one special kernel to the set of ten kernels used before. We vary the total coverage of all other kernels at 10% (left plots) and 30% (right plots), while setting the coverage of the special kernel at 20% (upper plots) and 40% (lower plots). The results show that the dedicated ASIC accelerator in only beneficial when its targeted kernel has a dominant coverage with applications, e.g. in Figure 2.12c.

of all other kernels varies from 10% through 40%. When the performance of the dedicated ASIC accelerator is merely 5x better than RL, the system ends up with zero allocations to the dedicated accelerator, unless the coverage of its targeting kernel (20%) is larger than the total coverage of all other kernels (10%). However, when the performance of the dedicated accelerator boosts to 50x better than RL, the dedicated accelerator will always hold its place with 10% allocation out of all u-cores area. When the performance ratio is 10x, the dedicated accelerator is beneficial unless the total coverage of all other kernels is as large as 40%, overwhelming the special kernel's coverage of 20%.

In Figure 2.13b, the total coverage of all other kernels is fixed at 20%, while the special kernel's coverage varies from 10% through 40%. It is similar that the dedicated accelerator is preferred when there is a huge gap between either: 1) the performance of the dedicated and the reconfigurable accelerator (e.g. 50x), or 2) the coverage of the dedicated kernel and the total coverage of all other kernels (e.g. 2x larger).

In summary, the benefit of ASIC accelerators depends on its relative throughput to RL accelerators on the same kernel. With a large throughput gap (e.g. 50x), ASIC accelerators are beneficial even when the targeted kernel has a limited presence across applications (e.g. as low as 10%). Otherwise, RL is preferable, as long as reconfiguration overhead can be neglected.



(a) 20% coverage of the fixed kernel

(b) 20% coverage of all the other kernels

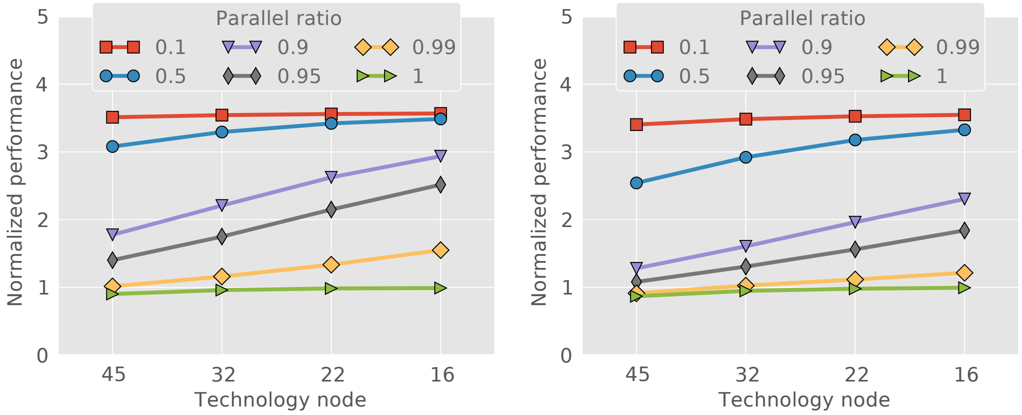
Figure 2.13: Sensitivity study on ASIC performance ratio.

2.3.7 Alternative Serial Cores

Although massive parallelism has been observed in several computing domains, such as high-performance computing, there are many applications with a limited parallel ratio. In this case, adding a “beefy” out-of-order (O3) core is more beneficial, especially when dim cores get diminishing returns on throughput. We extract the performance of the O3 core from SPEC2006 scores of a Core i7 processor and calculate its power and area using McPAT, which are normalized and summarized in Table 2.7.

Table 2.7: Characteristics of an O3 core at 45nm, normalized to the in-order core at the same technology node.

Perf.	Power	Area
2.2x	3.5x	3.46x



(a) Large chip with area of $200mm^2$

(b) Small chip with area of $107mm^2$

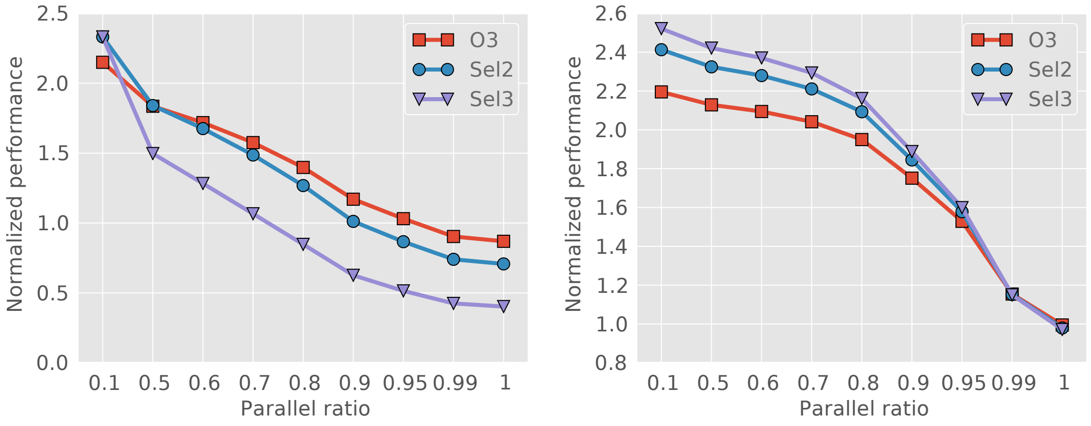
Figure 2.14: Relative performance by using O3 core to run the serial part of an application. System budget is large in 2.14a and small in 2.14b, as characterized in Table 2.5. Y-axis is the performance normalized to the system at the same technology node, which only includes in-order cores and uses one of the in-order cores as the serial core.

We assume the same inter-technology scaling factors as we have used for in-order cores in all previous analyses. We also assume the serial core is gated off in parallel mode to save power for dim cores. As shown in Figure 2.14, even with an embarrassingly parallel application, the die area investment on dedicated O3 core is still beneficial. In the case of ideal parallelism, the performance loss by introducing an O3 core is around 14% at 45nm. The performance loss reduces to less than 1% at 16nm, due to diminishing performance returns from a larger number of throughput cores, and lower percentage area impact of one O3 core.

Alternatively, the serial code can be executed by the best core selected from several out-of-order candidates, as proposed in [56]. In this work, Najaf et al. observed a 10% performance improvement by selecting from two alternative out-of-order cores, compared to an optimal-in-average out-of-order core. We model core-selectability with assumptions of 10% better performance but twice the area as an out-of-order core. To study the potential limits, we model an alternative organization of core-selectability with a total of three O3 cores to be selected, and assume another 5% performance boost by introducing more selections (a total of 15% over one O3 core "group"). We use the small system budget summarized in Table 2.5. As shown in Figure 2.15a, the original core-selectability (Sel2) shows less benefit as long as the parallel ratio is larger than 50% at 45nm. At 16nm (Figure 2.15b), the original core-selectability outperforms the conventional core with almost all parallel ratios. The core-selectability in the alternative organization (Sel3) delivers worse throughput at 45nm unless the application is almost serial with a parallel ratio of 10%. However, when it comes to 16nm, the alternative core-selectability is better for almost all parallel ratios. This is, again, because the number of throughput cores is so large that more in-order cores suffer from diminishing performance returns. Therefore, the parallel performance penalty is limited when trading off a couple of in-order cores for single thread performance.

2.4 Conclusions

In this chapter, we describe *Lumos*, a framework for exploring the performance of heterogeneous systems operating at near-threshold (e.g. with dim cores). We show that dim cores manage to provide a moderate speedup up to 2x over conventional CMP architecture (suffering from dark silicon issue with further technology scaling). However, the poor per-core frequency of dim cores leads to diminishing returns in throughput, creating an opportunity for more efficient hardware accelerators



(a) 45nm

(b) 16nm

Figure 2.15: Performance comparison among systems implementing the serial core as an out-of-order (O3) core, core-selectability of two O3 cores (Sel2), and core-selectability of three O3 cores (Sel3). The system budget is small from Table 2.5. Y-axis is the performance normalized to the system at the corresponding technology node, which only includes in-order cores and uses one of in-order cores as the serial core.

such as RL and ASIC. Lumos models the general-purpose workload via a statistical approach. We show that RL is more favorable for general-purpose applications, where commonality of kernels is limited. A dedicated ASIC accelerator will not be beneficial unless the average coverage of its targeted kernel is twice as large as that of all other kernels or its speedup over RL is significant (e.g. 10x-50x). However, it is hard to identify common function-level hotspots in real-world applications, and this is even true with domain-specific applications. In fact, one of the most important conclusions from this chapter is the need for efficient, on-chip, RL resources that can be rapidly reconfigured to implement a wide variety of accelerators.

Chapter 3

Lumos+: Design Space Exploration for MPSoCs

Integration of accelerators to create heterogeneous processors is becoming more common for both power and performance reasons. In most systems today, a design with only conventional CPU cores cannot meet performance targets without exceeding power constraints. Fixed-function accelerators can provide dramatic speedups over CPU cores, but sacrifice flexibility and programmability. Reconfigurable logic (e.g. FPGAs) is more flexible, so that the same hardware block can accelerate various functions. Another advantage of reconfigurable logic is that hardware can be updated to fix bugs, further optimize performance or power, or adapt to changing kernel characteristics. However, the overheads of reconfigurability reduce the performance and power-efficiency compared to fixed-function logic.

Various fixed-function and reconfigurable accelerators have been recently proposed in both industry [62] and academia [14, 16, 73, 92]. The increasing variety of both types of accelerators leads to an exploding design space in selecting how many CPU cores, fixed-function accelerators, and how much reconfigurable logic to provision, as well as the choice of optimal operating voltage. Furthermore, heterogeneous processors must serve increasingly diverse workloads. An efficient, automated *pre-RTL* design-space search methodology is needed to help identify the best configuration.

The Lumos [87] design-space exploration framework is a first-order analytical modeling framework for exploring the design space of accelerator-rich heterogeneous architectures, in order to determine the best mix of cores, fixed-function accelerators (FF-Accs), and reconfigurable logic (RL). Lumos extends existing hardware models from [35] and [16] with a statistical workload model and a fine-grained voltage-frequency scaling model calibrated by circuit simulation. Lumos+ extends Lumos with *GAopt*, a genetic search heuristic, to find the accelerator allocation that achieves the best throughput for diverse workloads. Lumos+ also adds modeling of the memory hierarchy, more detailed modeling of the reconfigurable logic, and more detail in the workload model. Finally, we propose a performance metric, *volatility*, to evaluate performance sensitivity to workload variations.

In addition to the GAopt search method, the primary contributions of this chapter are:

1. To show the importance of performance *volatility* as an evaluation metric for heterogeneous architectures. An architecture with low *volatility* achieves close-to-optimal performance across a majority of applications, even though it may not be best for any single application.

2. For general-purpose applications with a large number of kernels with diverse characteristics, we show that systems equipped with reconfigurable logic achieve the best overall performance and lowest volatility.

This work will be appeared in DAC 2016.

3.1 Related Work

Hardware accelerators have attracted a great interest from both industry and academia. For example, Catapult [62] is an FPGA-based reconfigurable accelerator for Microsoft's Bing service; In [34], personal intelligent assistant kernels from Sirius suite are accelerated by FPGA implementations; Diannao [14] is an application-specific accelerator implemented for common operations in Neural Networks. All these have reported substantial performance and efficiency improvements over conventional many-core systems, strongly motivating heterogeneous architectures with accelerators to cope with increasingly stringent power constraints.

Recently, researchers have proposed infrastructures to understand the architectural design tradeoffs introduced by hardware accelerators. Aladdin [73] extracts accurate models from applications' execution traces, enabling explorations of designs on dedicated accelerators for any given application. On the other hand, PARADE [17] takes advantage of high-level synthesis (HLS), and integrates the accelerator models with a cycle-accurate simulator (Gem5). As a result, PARADE can simulate end-to-end applications running on a user-defined accelerator-rich architectures. Lumos+, which extends Lumos [87], is complementary to Aladdin and PARADE in two aspects: first, we use analytical models for power and performance, trading off precision for fast evaluation time; secondly, we explore the design space at a higher level to find the optimal system configurations given a number of accelerator candidates. Our work can benefit from Aladdin's more accurate power-performance characterization of accelerators, and our work can narrow down system configurations for PARADE's more detailed but time-consuming evaluations.

3.2 Lumos+

In the Lumos+ model, a processor consists of conventional general-purpose cores of various size and performance levels, and optionally, one or more hardware accelerators. For simplicity, we assume there is only one reconfigurable block that can be shared in time or space (the latter is left for future work). Various figures of merit can be optimized, but in this chapter, we maximize performance subject to power and area constraints. For each candidate hardware organization, voltage, and hence frequency, are determined so that the power constraint is met. Performance is determined according to the frequency and a model for performance as a function of workload characteristics.

3.2.1 Technology Scaling Model

We employ circuit simulations to determine energy-delay characteristics of a given technology process and derive a map of frequency and power as a function of voltage. This is achieved by simulating a 32-bit ripple-carry adder using SPICE, for various supply voltages. The complexity of an adder makes it a better approximation for logic with multiple critical paths than a simple inverter chain. We use the predictive technology model (PTM) from [75].

3.2.2 Performance Model

To model the performance of a kernel running on cores, we take a similar approach to the validated model in [26]. This model extends Amdahl's law by factoring cache hit rate and latency into the multi-core performance scaling.

$$Perf = N\frac{freq}{CPI_{exe}}\mu$$

where N is the number of active cores subject to the power constraint, CPI_{exe} is the effective cycles-per-instruction (CPI) excluding stalls due to cache accesses, which are considered separately in core utilization (μ). μ is calculated by factoring in the average memory access latency, percentage of memory instructions, and miss rates.

To model the speedup and the power consumption of a hardware accelerator (FF-Acc or RL), Lumos+ uses a pair of parameters (η, ϕ) for each kernel, following [16,87]. ϕ is the power efficiency normalized to a single, baseline core, and η is the relative performance normalized to the same baseline. The cost of data movement is lumped implicitly into the performance parameter (η) of an accelerator. In general, hardware accelerators are much more power-efficient than cores. The area allocated to a hardware accelerator also tends to be small. We do not yet model complex task graphs so that different accelerators can be in use concurrently; in applications we have studied, such task-level parallelism is small enough not to affect the eventual design. Although the CPU cores can together be power-limited and need to run at lower voltage for a parallel task, an accelerator will rarely be power limited. The goal of Lumos+ is to explore the best potential configuration of cores and accelerators, so we assume the memory bandwidth is always sufficient. With these assumptions, we model the relative performance of an accelerator proportional to its area, which is a simplifying first-order approximation suitable for tasks with plentiful parallelism, but overlooking synchronization and data-transfer overheads for more complex algorithms.

For simplicity, Lumos+ assumes that the power and the area of all un-core components remain a constant ratio to the whole system. For this study, we study processor design for data-center workloads and, following [48], assume that 50% of both the total thermal design power (TDP) and the die area are available for processing units, with the rest allocated to memory controllers, I/O, etc. We use the Oracle SPARC T4, a representative server-class design, with TDP and area of $120W$ and $200mm^2$, to set our power and area budgets (i.e., 60W and $100mm^2$). For the CPU cores, the Niagara2-like in-order core is used, as it is the latest design supported by McPAT and a close predecessor to SPARC T4. At the baseline 45nm, a single core consumes $7.2W$ and takes $7.65mm^2$. We scale these according to McPAT rules for more recent nodes.

3.2.3 Workload Model

A workload consists a set of N kernels $(K_1, K_2, \ldots, K_N)$ and M applications $(A_1, A_2, \ldots, A_M)$. For each application A_i, the normalized execution time of the kernel K_j using a single baseline core is denoted as t_{ij}. $t_{ij} = 0$ if application A_i does not include kernel K_j. We use S_i^a to denote the speedup of the application A_i by a given system configuration, and calculate S_i^a as:

$$S_i^a = \frac{1}{\sum_{j=1}^{M} \frac{t_{ij}}{s_j}} \tag{3.1}$$

where s_j is the speedup of the kernel K_j achieved by the given system configuration. In the case that more than one computation units are available for a given kernel (K_j), its speedup is defined as the highest performance achieved among all computation units:

$$s_j = \max(s_j^{\text{MP}}, s_j^{\text{RL}}, s_j^{\text{FF}}) \tag{3.2}$$

where s_j^{MP} is the speedup achieved by many core parallelization, s_j^{RL} is the speedup achieved by RL, and s_j^{FF} is the speedup achieved by an FF-Acc. Finally, the speedup of a workload is defined as a weighted average of speedup achieved for every applications within the workload.

In order to compose state-of-the-art representative workloads, we derive applications from kernels in the Sirius suite [34]. This suite consists of kernels extracted from Sirius, an open end-to-end standalone speech and vision based intelligent personal assistant (IPA). It includes speech recognition, image matching, natural language processing, and intelligent question-and-answer systems. Finally, for this study, we adopt the speedup of reconfigurable accelerators from reported data in [34]. To obtain speedups of FF-Accs, we scale up the speedup by a constant ratio from the RL implementations of the same size. We use 5–40x as conservative and aggressive values, as they represent the range of performance ratios from literature.

3.2.4 Reconfiguration Overhead

The performance model of the reconfigurable accelerator described in the previous section implicitly ignores the reconfiguration overhead. It can be justified by the assumption that each kernel within an application takes sufficient time to dominate the time spent on setting up the execution context. Unfortunately, this is not always the case. To consider the reconfiguration overhead of a kernel within an application, we introduce a pair of parameters to denote the number of reconfiguration operations (N^{rc}) and the associated overhead (T^{rc}). The power overhead of reconfiguration tends to be trivial compared to the overall system power budget. As a result, we only focus on the latency overhead associated with reconfiguration operations. According to the cost model of FPGA reconfiguration in [60], we model the latency overhead as proportionally to the size of the reconfigurable accelerator. After factoring in the overhead, the runtime of the kernel K_j in the application A_i by reconfigurable acceleration can be expressed as:

$$t_{ij}^{\mathrm{RL}} = \frac{t_{ij}}{s_j^{\mathrm{RL}}} + N_{ij}^{\mathrm{rc}} \cdot T_{ij}^{\mathrm{rc}} \tag{3.3}$$

where s_j^{RL} is the speedup achieved by a reconfigurable accelerator. N_{ij}^{rc} is the reconfiguration count of the kernel K_j when it presents in the Application A_i. Its value is at least 1 if t_{ij} is none-zero, because the reconfigurable fabric has to be reconfigured at least once when the targeted kernel is invoked at the first time. Finally, T_{ij}^{rc} is the reconfiguration overhead of the kernel K_j within the application A_i. T_{ij}^{rc} is normalized to the length of the application executed by a single baseline core.

3.3 Domain-Specific Applications

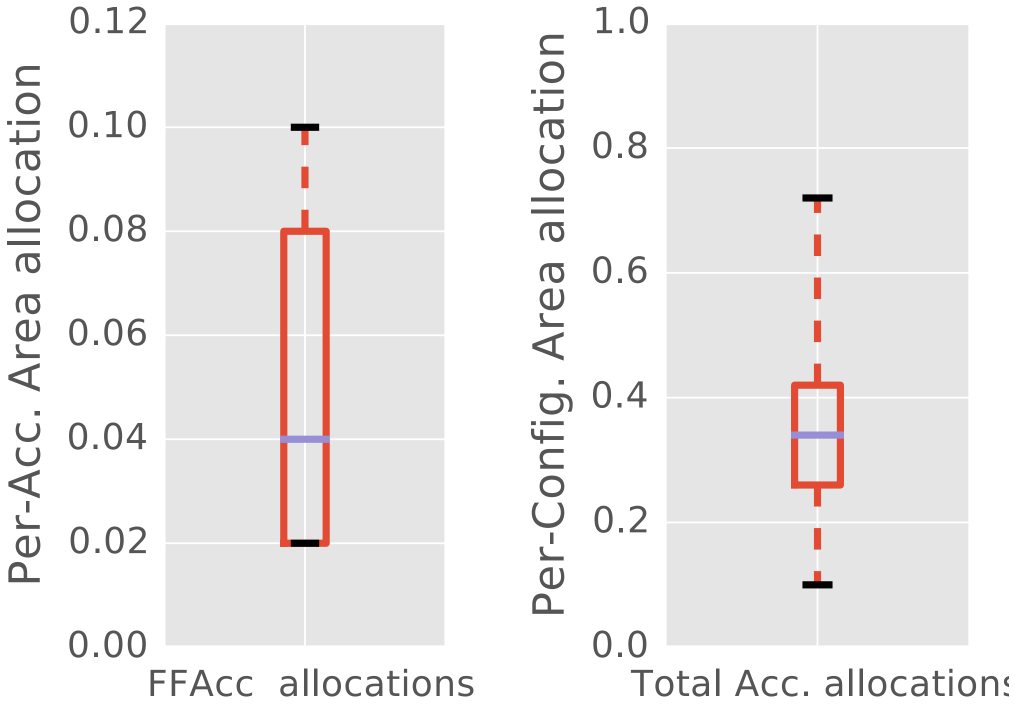
We define domain-specific applications to be applications that share a small set of kernels, with these kernels likely to present in each application. To conduct the design space exploration on domain-specific applications, we synthesize a set of applications using all seven kernels from the

Sirius suite [34]. We use a synthetic kernel, “serial”, to model the serial computation of applications. The “serial” kernel can only be executed on a single conventional core. Further, we introduce another synthetic kernel, “coreonly”, to lump together computations that can be accelerated only by many-core parallelization but not any types of hardware specialization. We employ a Monte-Carlo sampling approach to generate 10,000 applications from a pool composed of seven Sirius kernels and the two special synthetic kernels that are only applicable for conventional cores. In this section, we first describe a simple brute-force search scheme, named *BFopt*; then we use *BFopt* to explore the best system configuration among synthetic applications as just described.

3.3.1 BFopt

Since the number of FF-Acc candidates is small, it is possible to conduct a brute-force search on the design space to find the optimal system configuration that delivers the best performance. We call this scheme *BFopt*. To make it more practical, we employ two heuristics to trim down the space of system configurations. First, we limit the area allocation for FF-Accs. This is because the throughput of an FF-Acc is generally much better than many-core parallelization and reconfigurable accelerators. A small FF-Acc is good enough to speedup its targeted kernel significantly. In our study, we set the upper limit of area allocations on an FF-Acc to be 10% of the total system area budget. Second, we assume that conventional cores cannot be eliminated completely. In reality, an application typically has some portion that can only be executed by conventional cores (e.g. the “serial” and the “coreonly” synthetic kernels). In our study, we set the area allocated to cores to be at least 20% of total the overall processing area budget.

As a result, we set up *BFopt* in such a way that the area allocation of each FF-Acc ranges from 0 to 10% with a step size of 2%, while the area allocation of RL ranges from 0 to 70% with a step size of 10%. After removing candidates that lead to less than 20% of total area budget for cores, the effective search space is around 1.4 million configurations. Thanks to the fast evaluation of *Lumos+*, the brute-force search takes minutes rather than days on standard servers.



(a) FF accelerators.

(b) FF-Accs. and RL

Figure 3.1: Area allocation with *BFopt* for accelerators with a domain-specific workload, with an FF-Acc speedup of 40X. The box extends from the lower to upper quartile values of the data, with a line at the median, and the whiskers indicating min/max of the data. (a) distribution of per-accelerator area allocations on FF-Accs within these optimal configurations. Allocations on FF-Accs range from 2% to 10% with a step size of 2%. (b) distribution of total accelerator allocations (including RL and FF-Accs) of each optimal configuration.

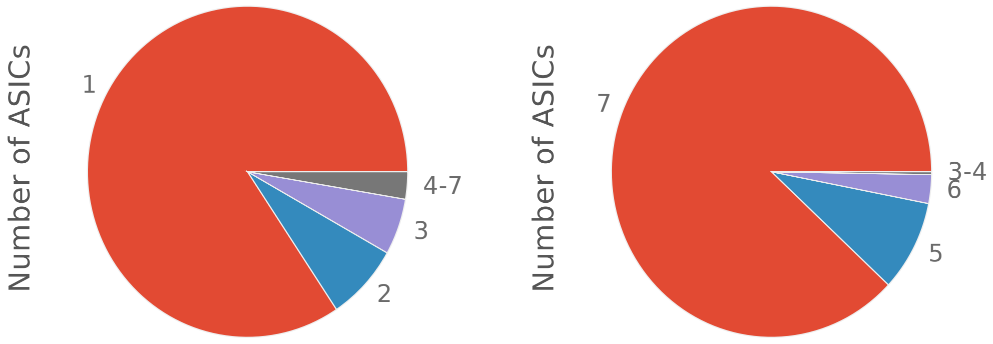
3.3.2 Area Allocations on Accelerators

As a validation to the heuristics we applied to *BFopt*, we first look at the area allocated to accelerators across optimal configurations for 10,000 synthetic domain-specific applications. For each application, we invoke *BFopt* to find the optimal configuration of area allocations on accelerators and cores.

We use a box-plot to show the results, where the box extends from the lower to upper quartile values of the data, with a line at the median, and with whiskers indicating min/max of the data. As shown in Figure 3.1a, when the performance advantage of an FF-Acc is as large as 40x over its RL counterpart, the optimal area allocations for a majority of FF-Accs are less than 8%, validating the first heuristic that *the optimal area allocation for an FF-Acc is small*. In Figure 3.1b, the overall accelerator allocations of each optimal configuration are mostly in the range of 15% to 45%, with outliers reaching up to 70%. Even considering these outliers, the total allocations are still well below 80%, corroborating the second heuristic that conventional cores cannot be eliminated completely (i.e., more cores are provisioned than the heuristic baseline of 20%). When the performance advantage of FF-Acc's is 5x, we observe a similar shape of the resulting data set. We omit plots of 5x for succinctness.

3.3.3 Number of Dedicated Accelerators

As dedicated accelerators are generally more expensive to design and manufacture, it is critical to understand the optimal number of FF-Accs to achieve the best performance in a cost-effective way. We study the same set of optimal configurations obtained from the previous analysis, and use a pie chart to summarize the number of FF-Acc's in each optimal configuration. In Figure 3.2a, where the performance ratio of FF-Accs to RL is 5x, most of the optimal configurations end up with no more than one dedicated accelerator. This is because an RL accelerator with a large allocation outper-



(a) Perf. ratio: 5x

(b) Perf. ratio: 40x

Figure 3.2: Number of dedicated ASIC accelerators in optimal configurations, plotted in pie charts.

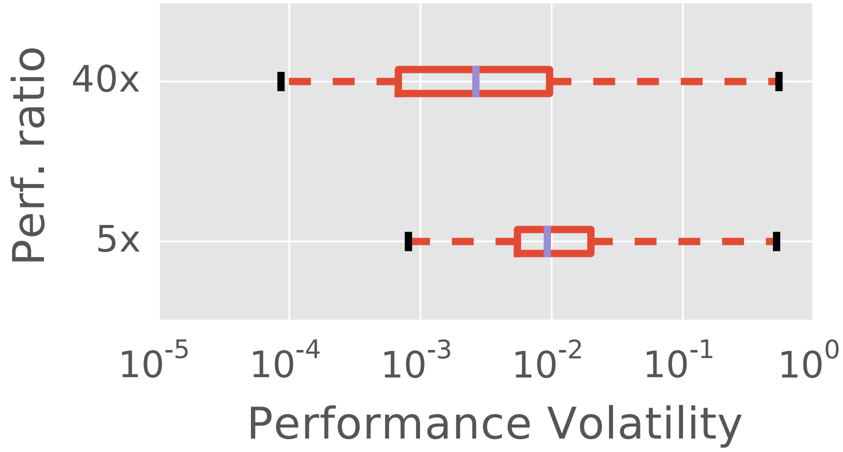


Figure 3.3: Distribution of performance volatility of optimal configurations for ASIC performance ratio of 5x, and 40x. The box-plot setting is the same as described in Figure 3.1.

forms the corresponding dedicated accelerator with a small area allocation, and the large allocation on RL is justified by its flexibility in acceleration across a range of kernels. However, when the performance ratio of the FF-Acc is large enough, e.g. at 40x, a small allocation on is more than enough to deliver a substantial kernel speedup, leaving non-accelerating kernels (e.g. "coreonly" and "serial") the new performance bottleneck. Therefore, as plotted in Figure 3.2b, a majority of optimal system configurations include dedicated accelerators for all seven Sirius kernels, and avoid any RL allocations at all.

3.3.4 Performance Volatility

The overall throughput of a heterogeneous system, especially when it is loaded with FF-Accs, depends heavily on the frequency of its targeted kernels in each application. More specifically, the system performs best on applications in which the targeted kernels are heavily weighted, and suffers in applications within which the targeting kernels have lower weights. To study the performance heterogeneity of a system configuration across applications, we propose a new metric, *volatility*, to describe the performance stability of a heterogeneous system across a workload. *Volatility* ($\mathcal{V}$) is defined as the sum of "square residuals" of system performance across all applications within a workload:

$$\mathcal{V} = \frac{1}{N}\sum_{i=1}^{N}\left(1 - \frac{s_i^a}{S_i^{\text{opt}}}\right)^2 \tag{3.4}$$

where s_i^a is the performance of the system on application A_i, S_i^{opt} is the performance of the system that achieves the highest throughput on application A_i, and $1 - s_i^a/S_i^{\text{opt}}$ is the performance "residual" for the application A_i. The value of $\mathcal{V}$ ranges from 0 to 1. The smaller the $\mathcal{V}$, the higher likelihood of a system configuration to deliver close-to-optimal (if not optimal) performance for applications within a workload. For example, when $\mathcal{V} = 0.01$, due to the square of "residual", it implies that the configuration performs within 90% of the optimum on an average basis. We show the distribution of performance volatility of optimal configurations from previous analysis in Figure 3.3. The performance volatility values of optimal configurations in both cases (5x and 40x) are as low as 0.01. This is because most of the optimal configurations for domain-specific applications are similar in the number of accelerators as well as the type of accelerators. A configuration that is optimal for one application performs close-to-optimal for other applications as well, resulting in a low performance volatility across the workload.

3.4 General-Purpose Applications

Next we extend our exploration to general-purpose applications. In contrast to domain-specific applications, where the set of kernel behaviors is limited and can be more carefully characterized, general-purpose applications vary significantly in the variety of potential kernels, meaning that applications can differ substantially.

We can use the same approach to synthesize applications for our design space exploration. In addition to the special kernels of "coreonly" and "serial," we create a pool of 100 kernels whose RL speedups are evenly sampled from the range determined by the seven Sirius kernels. Then we synthesize 500 applications by randomly sampling kernels out from this pool for each application. We first explore designs with applications that have 15 kernels per-each. Then we show a sensitivity study on applications that have 10 and 20 kernels.

Because the number of kernel candidates across these general-purpose applications is so large, the prior exhaustive search mechanism is no longer practical. We propose a heuristic search using a genetic algorithm to find the best allocation of cores, FF-Accs, and RL.

3.4.1 GAopt

Genetic algorithms (GAs) are search heuristics that mimic the process of natural selection. To solve a problem using GAs, one first needs to define an "individual" and initialize a *population* as the start point. The GA then applies a combination of the user-defined *mutation* and *crossover* operations to breed a "child" generation from the "parent." At the end of each iteration, the new child generation goes through a *selection* process to choose the best "individual" judged by a user-defined *fitness* function. This genetic evolution process iterates until either the new population has converged or a preset maximum of iterations has been reached. To solve our area allocation problem, we set up a GA as follows:

1. An individual is defined as a vector of $[a_0, a_1, \dots, a_n]$, where a_0 is the area allocation for the reconfigurable block, and a_1 through a_n are the area allocations for the n FF-Acc candidates. Any of the $[a_0 \dots a_n]$ can be zero. In this study, we assume that each kernel can have its own FF-Acc, and there are n kernels. Note that the area allocation on conventional cores is implied as $1 - \sum a_i$.

2. We initialize the population by generating individual vectors with random area allocations, and discard vectors that imply a less-than-20% area allocation for the cores. This is consistent with the second heuristic presented in Section 3.3.1.

3. The *crossover* operation just selects two random dedicated accelerators and exchanges their allocations between the two individuals. The *mutation* operation selects a new, random allocation for each accelerator.

4. The *fitness* function of each individual is the performance of the candidate system configuration on the given application.

Pop. size	Size of new generation	number of generations	Crossover prob.	Mutation prob.
300	600	100	0.6	0.3

Table 3.1: Parameters for *GAopt* using DEAP framework

5. The *selection* process select the *k* best individuals for the next generation, where *k* is the population size. This is chosen empirically to optimize the speed and accuracy of GAopt.

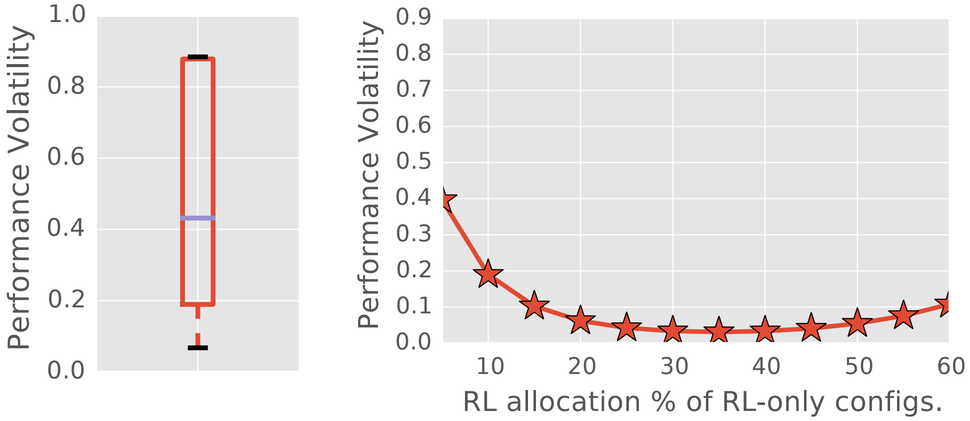
We implement *GAopt* using the DEAP framework [29], and use parameters summarized in Table 3.1. The first three parameters in Table 3.1 are chosen to ensure the *GAopt* to find the optimal results in a reasonable amount of solving time. In our experiments, *GAopt* finds no better configuration with a larger population size or more generations. The last two probability parameters adopt the typical values suggested by the DEAP framework.

We compare *GAopt* results against *BFopt* on application-specific applications studied in the previous section. The results obtained from *GAopt* match closely to the results from *BFopt*, with the worst-case relative performance of 0.98 and 0.99 for performance ratio of 5x and 40x, respectively. In some cases, *GAopt* is even slightly better than *BFopt*. This is because *BFopt* searches for a more coarse-grained space due to our pruning heuristics. *GAopt* solves for the optimal configuration for a single application in 262.91s on a server with an Intel Xeon X5550 processor using a single thread, while *BFopt* spends 8027.66s to solve for the same application on the same server. *GAopt* achieves a 30x speedup. For a workload consisting of 500 applications, BFopt is prohibitively expensive, while GAopt completes in 1.5 days.

3.4.2 Importance of Reconfigurable Accelerators

Heterogeneous architectures with FF-Accs can provide substantial performance and power-efficiency improvements over conventional multicore organizations. However, the high development, qualification, and chip-area cost of a dedicated accelerator cannot be justified unless the target kernel is prevalent across applications, so only a limited number of accelerators can be provisioned. Furthermore, general-purpose workloads may need a wide range of accelerators, and very few FF-Accs are likely to receive high utilization. Even its speedups are lower, RL is often a better use of area, because it can provide speedups and power-efficiency benefits for a wide range of kernels.

To explore this tradeoff space for general-purpose applications, we characterize the distribution of performance volatility across 500 applications in a box-plot on the left in Figure 3.4, using the same settings as described in Figure 3.1. Each of the 500 applications are synthesized by randomly selecting 15 out of the 100 kernels. Performance ratio of FF-Accs is 40x, since RL has already dominated allocations on accelerators in the optimal configurations when the ratio is 5x (see Section 3.3.3). The plot suggests that most of the optimal configurations have a performance volatility larger than 0.2. This suggests that the performance of these configurations are worse than the optimal by more than 40% on average. Then we look at the performance volatility of systems composed of only reconfigurable accelerators. In the right plot, X-axis indicates the area allocation on reconfigurable accelerators, in terms of percentage to the total area budget. It shows that configurations with a considerable amount of allocation on a reconfigurable accelerator have a much lower performance volatility (the minimum volatility is 0.031 when RL accounts for 35% of total area budget). These configurations may not give the optimal speedup for any single application, but performance is close to the optimal for a majority of applications. More specifically, the configuration with RL



(a) Per-application optimal configurations

(b) RL-only configurations

Figure 3.4: Performance volatility comparison between per-application optimal configurations and RL-only configurations. Performance ratio of FF-Accs is 40x.

allocation of 35% achieves at least 80% of the optimal performance for 80% of applications. In the worst case, the configuration still achieves 65% of optimal performance, showing much better performance potential than configurations that include both RL and FF-Accs.

Next, we explore the optimal area allocation for the RL. We test this for three workloads, one with 10 randomly selected kernels per application, one with 15, and one with 20. From the previous analysis, we learn that RL-only configurations are sufficient in most cases. Therefore, we focus on RL-only configurations in this study. The remainder of the area is therefore dedicated to CPU cores.

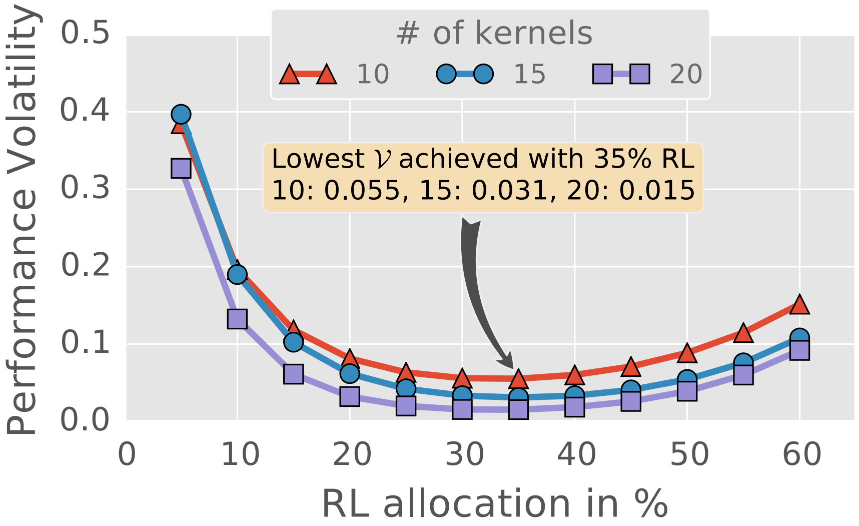


Figure 3.5: Performance volatility comparison between per-application optimal configurations and RL-only configurations. Performance ratio of FF-Accs is 40x.

As shown in Figure 3.5, the lowest performance volatility is achieved when 35% of chip area is allocated for RL. (The remaining 65% of area supports 55 CPU cores, at 0.86V.) As the number of kernels per-application increases from 10 through 20, the minimum performance volatility decreases from 0.055 to 0.015, suggesting that RL-only configurations provide better performance stability across applications as the number of kernels per application increases.

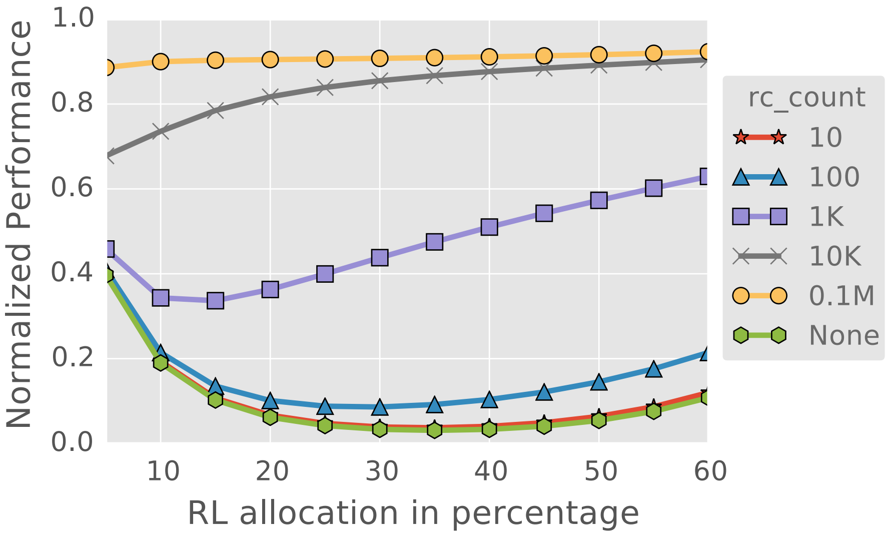


Figure 3.6: Impact of reconfiguration overhead on performance volatility of RL-only configurations, as a function of the number of reconfiguration options per application. “None” means that reconfiguration overhead is ignored.

3.5 Reconfiguration Overhead

One drawback to RL is that reconfiguration overhead can compromise overall performance significantly. To study the impact of reconfiguration overhead to the performance volatility of RL-only configurations, We use the same set of general-purpose synthetic applications that have been used in the last section. We only show the results for applications that average 15 kernels per each, and we obtain similar conclusions for applications with 10 and 20 kernels. As shown in Figure 3.6, when the reconfiguration count is small (e.g. rc_count=10), its impact on performance volatility is minimal. As the reconfiguration count increases, the RL allocation in the system that achieves the lowest performance volatility decreases. This is because a larger RL requires more time to reconfigure, compromising its performance benefit over a smaller RL. When the reconfiguration count is embarrassingly high (e.g. 0.1M), the performance volatility of systems with RL-only accelerators are constantly high across all RL allocations, suggesting that systems loaded with RL-only accelerators perform sub-optimally across target applications.

The negative performance impact of reconfiguration overhead advocates mitigation techniques in order to make reconfigurable accelerators beneficial. These overheads can be mitigated at the application or architecture level. At the application level, it is very important to design RL-friendly algorithms that minimize either the amount of reconfiguration or the time of each single reconfiguration process. At the architecture level, it may be preferable to reduce the area of any particular accelerator on the reconfigurable block, so that multiple accelerators can be instantiated concurrently, allowing for greater reuse without intervening reconfiguration operations. The effectiveness of this approach will be limited by the diversity of kernels, but temporal locality in execution may still allow for significant reuse, for the same reasons that instruction caches are effective for CPUs. Partial reconfiguration can help exploit temporal locality, by dynamically adapting the mix of kernels currently instantiated on the RL to match application use patterns, replacing one kernel at a time. Compile-time or run-time prediction can also allow pre-fetching of kernels, so that they can be installed before they are needed. Partial reconfiguration also allows accelerators to grow and shrink as needed.

3.6 Conclusions

In this chapter, we use an analytical framework called Lumos+ to explore the design space of heterogeneous architectures composed of hardware accelerators. With the help of a genetic algorithm based search heuristic, we show that dedicated accelerators are only beneficial when their performance premiums are large enough (e.g. 40x) to compensate for limited programmability, or when only a few kernels need acceleration. However, systems equipped with reconfigurable logic are promising to achieve close-to-optimal performance consistently across applications with a wide range of kernel characteristics, an important implication for rapidly evolving, general-purpose workloads.

Chapter 4

AFI: Application-Level Fault Injection

Traditionally, system reliability is only a serious concern for mission-critical systems, such as unmanned aerial vehicles (UVAs), or high-end server systems, such as IBM POWER series processors. While this convention still holds, reliability has become to one of the first-class design constraints for almost all modern computer systems. Moore's Law continues its pace to doubling the number of transistors on a chip. This results in a higher probability of a chip to fail. To make it even worse, technology scaling keeps delivering smaller feature size and lower supply voltage for devices. Single-event upset (SEU) caused by high energy particle strike are exacerbated at lower voltages, due to the critical charge (Q_{crit}) required to cause a latch or SRAM bit flip diminishes correspondingly. Similarly, transient errors caused by voltage noise also increase in incidence rate under low-voltage operations.

Errors are expensive to detect and recover. Redundancy-based techniques, such as dual-modular redundancy (DMR) and triple-modular redundancy (TMR), suffer from considerable amount of overheads in system power and area. On the other hand, checkpoint-based techniques have non-negligible run-time overhead, and could potentially prevent forward progress if the soft error rate (SER) is too high. Meanwhile, extra energy consumed by recovery hurts the overall energy efficiency of the system. This leaves system designer with worse performance as well as even more pressure on already stringent power budget.

Fortunately, a given machine-application pair offers significant masking of effects in the face of spurious bit-flips; so, the raw soft error rate (SER) of processor chips – even though it may increase with device density and lower voltages – is at least partially offset by large degrees of masking. For a given machine, different applications (or phases within an application workflow) present different degrees of masking.

It is also true that despite errors in a particular program output, consuming applications can often actuate actions accurately, thereby providing error-free outcome at the end. For example, an image processing and recognition algorithm can tolerate numerous pixel errors in the input data that may have been produced by an earlier image-scanning step.

Overall, the different levels of error tolerance in applications can be exploited by using dynamic voltage-frequency settings in such a manner as to optimize the net workflow performance/watt, without violating system resilience constraints [86]. Therefore, it is important to characterize and understand the inherent degree of transient error tolerance across the range of application workloads of interest in a particular embedded system domain.

To quantify the SER with regard to various applications, we present a framework based on a software-implemented fault injection (SWIFI) tool, called AFI. It features a runtime-independent triggering mechanism to enable flexible parallel experiment runs, as well as a mechanism to characterize only the region-of-interest (ROI) of an application through source code annotation. The tool is build on top of standard debugging interface and performance counters that are available in most of modern operating systems and processors. AFI is currently implemented on AIX$^{\text{TM}}$for POWER7 processor for demonstration purpose. It can be easily ported to other platforms such as X86 on Linux (we will add this support later).

This work has been published in IISWC 2014 [85].

4.1 Related Work

Soft error resilience has been studied over several decades (dating back to 1978 [97]) with a large body of more recent work at various levels of design hierarchy, such as logic level [21, 74], micro-architectural level [43, 89], and architectural level [8, 49, 50, 55]. For example, Mukherjee et. al.

proposed the concept of architectural vulnerability factor (AVF) in [55] to quantify the resilience of various architectural components. In [43], Kim et. al. studied soft error sensitivity of functional blocks using software simulated fault injections into the RTL model of a microprocessor (picoJava-II); Though showing significant masking effects (e.g. more than 85% reported by Wang et. al. in [89]), none of these prior works specifically considered application's algorithmic masking effects. Direct measurement based SER resilience characterization works [5,44,97] do not separate out the *MD* and *AD* components of the derating stack.

More recently, there has been an increasing interest in studying resilience at the application level for low-cost reliability solutions. SWAT/mSWAT [33, 46, 47, 68] take advantage of application's abnormal behavior, referred to as symptoms, to identify faulty units using light-weight diagnosis algorithms. In [80] Thomas et. al. propose the term EDC describing outcomes that deviate significantly from the error-free outcomes of an application. They use LLFI, an LLVM-based injection framework, to study EDC across six benchmarks from MediaBench. As mentioned in [90], LLFI injects errors at LLVM IR level, which may not be accurate for errors resulting in crash. AFI uses ptrace facility to inject errors directly into applications' architectural state, therefore does not have the inaccuracy issue. Besides, AFI captures EDC via SDC-related outcomes under relaxed result checking mechanism (e.g. SNR comparisons for SAR kernels in our study - see Section 4.4.1), and we have studied a more extensive set of applications for application-level resilience.

There are some early attempts on software implemented fault injectors by changing the value of architectural states. FERRARI, proposed in [41], also exploits the *ptrace* debugging facility just as we do. However, FERRARI waits for a timer interrupt to inject errors, where execution time would be shifted if there is any resource sharing contentions. Xception, proposed in [13], requires dedicated hardware support of debugging register. AFI only relies on commonly available features from operating system and processor hardware, therefore can be adapted to a broad platform. GOOFI-2, a recent tool presented in [77], integrates three injection "backends" in a single framework. One of the backends, the exception-based injection is the most similar approach to AFI. However, this approach changes target applications by adding an instrumentation function. While AFI keeps the ROI of target applications intact to avoid any noise to the fault characterization.

4.2 Application Level Resilience

4.2.1 SER Taxonomy

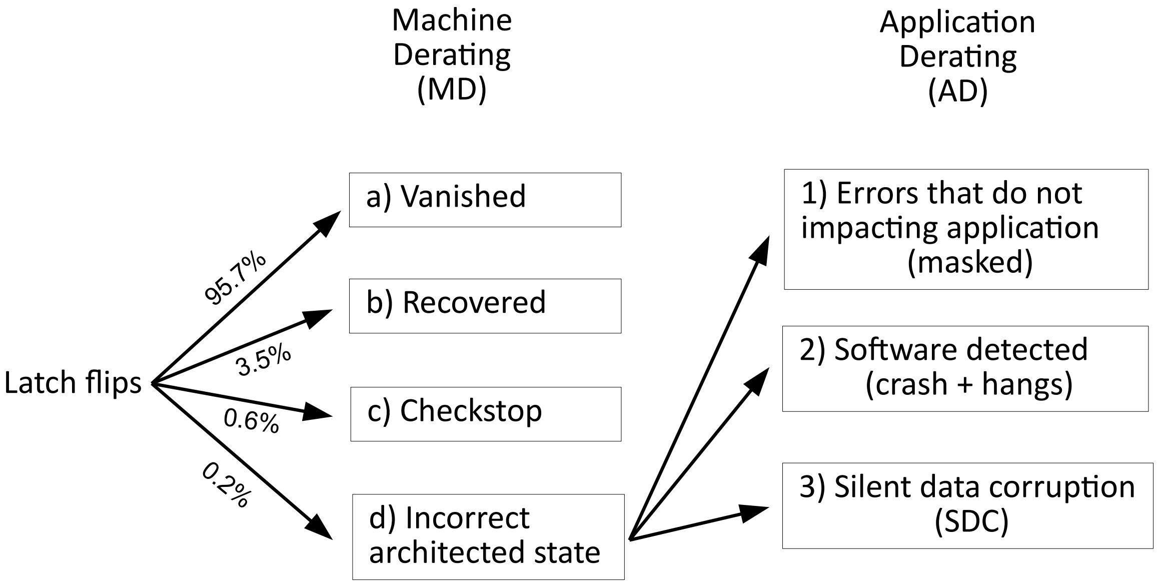

Machine Derating (MD)
Application Derating (AD)
Latch flips
95.7%
3.5%
0.6%
0.2%
a) Vanished
b) Recovered
c) Checkstop
d) Incorrect architected state
1) Errors that do not impacting application (masked)
2) Software detected (crash + hangs)
3) Silent data corruption (SDC)

Figure 4.1: SER taxonomy

Figure 4.1 depicts our adopted taxonomy [5,44] with respect to the classification of fault effects at the machine and application-level, under the influence of a single particle strike event (commonly referred to as single event upset or SEU). Following assumptions relevant in current technology nodes, an SEU is considered to be synonymous with a single latch bit-flip. Such taxonomy also applies in the context of fault injection experiments conducted during pre- or post-silicon soft error rate (SER) testing and evaluation activities. An SEU may result in one of four outcomes (as shown in the left segment of Figure 4.1), one of which results in the corruption of the architectural state visible to the executing application or workload. Note that in this particular description, we limit our attention to latch-level *single* bit-flips, assuming that SRAM elements (e.g. register files) are ECC-protected as in current generation POWER server-class processor [76]. SEUs affecting data sitting in such an SRAM buffer would be corrected in-place and so would be 100% masked.

Let us denote the probability of getting to an incorrect architectural (register or memory) state on a given fault injection or incidence by P_{md}, where *md* signifies machine or micro-architectural derating (or masking). In response to a random flip of a latch bit state (from 1 to 0 or vice versa), while an application is running, most of the time (e.g. with a probability of 0.957 for a measures case in the POWER6 design [5, 44]) there is absolutely no effect - i.e. no harm is done to the application. The application-visible architectural state can be corrupted with only a small probability P_{md} (=0.002 in the case cited in Figure 4.1).

An architectural state bit change can *potentially* result in silent data corruptions (SDC). However, given that an unwanted state bit change has occurred, the probability P_{ad} of that causing an actual SDC at the program output can also be rather small (e.g. typically somewhere between 0.05 to 0.2, although very rarely as high as 0.5). Here, *ad* stands for application derating, and refers to the masking offered by the application as a corrupted architectural state tries to propagate and eventually affect the correctness of the final program output. Given that the raw probability (P_{raw}) of a random latch bit-flip (within a given processor core) caused by high energy cosmic particle incidence (over a period of say 1 hour) is after all a very small number to begin with, one can see that the net probability P_{net} of an SDC (i.e., $P_{net} = P_{raw} * P_{md} * P_{ad}$) can indeed be very, very small. For example, suppose $P_{raw} = 10^{-3}$, in a lower-end embedded processor context; $P_{md} = 0.002$, $P_{ad} = 0.05$ (for a particular SER-hardened application A, measured over an execution duration of 1 hour). Then, $P_{net}[A] = 10^{-3} * 0.002 * 0.05 = 10^{-7}$ is the probability of an SER-induced SDC with the application A running for one hour. Over a period of a billion (i.e. 10^9) hours, the expected number of SDC errors seen by a continuously running application A would have been 100 ($= 10^{-7} * 10^9$). By definition, therefore, the failure rate of application A, running on the given processor (core) is 100 FITs (where FIT is a unit of measurement abbreviated from the term: failures in time).

For a system composed of 1000 processor cores, the net SDC FITs would be 100,000, resulting in an SDC MTBF of $10^9/10^5 = 10,000$ hours, which is slightly more than 1 year. For short-duration, mission-critical airborne flights, this about 1 year SDC-MTBF might seem to be quite acceptable. However, if the application is not specifically constructed to be SER-resilient, and if it has a larger P_{ad} value (e.g. 0.5), then the SDC-MTBF in this scenario would be about 36 days. This may still be barely acceptable for short-duration mission-critical systems, but for a long duration space mission (e.g. satellite-borne permanent space stations), an SDC-MTBF of 36 days would quite possibly be deemed to be unacceptable. Note also, that the above illustrative MTBF figures are constrained to SDC. The mean-time between application interrupts (MTBAI), which includes consideration for all detected, uncorrected errors (DUE) would be much smaller.

4.2.2 Evaluating a Target System for SER

Figure 4.2 shows the target application domain of interest in this chapter. The system of interest consists of a network of embedded processor systems that wirelessly communicate with each other, while also being connected to a ground-based server system (possibly offered as a cloud-based service) through an internet-enabled wireless communication protocol. Our focus of attention in this chapter is an individual embedded processor system that is engaged in a real-time image processing, recognition and response actuation mission. It may be noted, that depending on the size and scale of the supporting cloud infrastructure, the per-core SDC FIT value discussed earlier would be unacceptable in the design of such a cloud server or data center. For example, if we are talking about a million-core extreme scale data center, the SDC-FITs would, in the above case, be $100 * 10^6 = 10^8$, implying an SDC MTBF of 10 hours which is definitely unacceptable. So, what is acceptable for a single many-core embedded processor system, may be totally unacceptable in the context of the

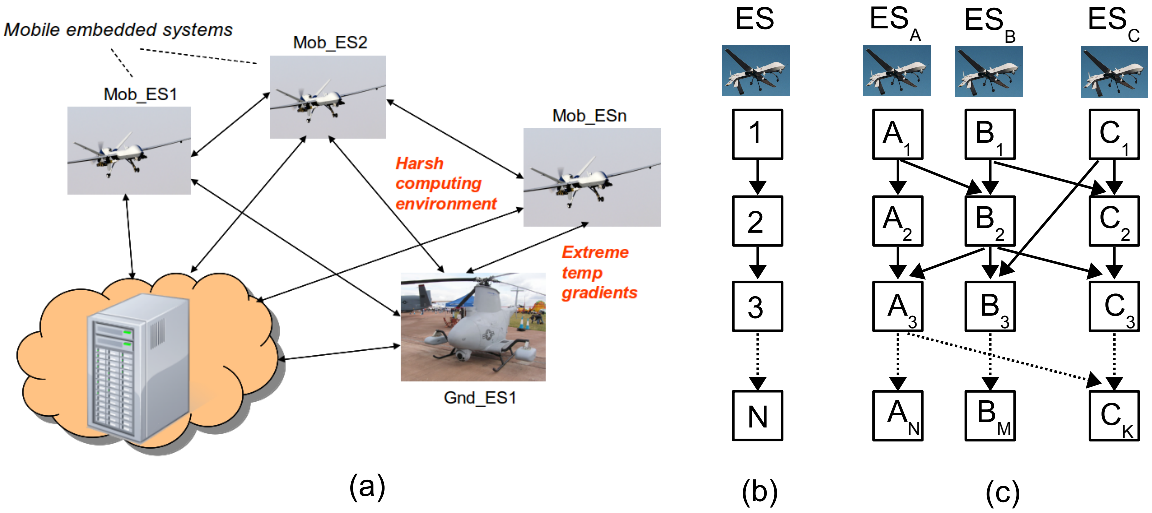


Figure 4.2: Cloud-backed airborne embedded system network, which operates under harsh conditions

large-scale supporting data center. Depending on the target system-level FITs and the size of the system, the per-core and per-chip FITs target can be deduced and this then becomes a strict design constraint.

To accurately predict system FIT values for a specific application workload in a pre-silicon setting, the associated *MD* and *AD* factors must be estimated with reasonable precision. Pre-silicon system-level methods for estimating the *MD* factor are found in prior works like AVF [55] and Phaser [65]. In this work, we focus only on the *AD* component of the system derating stack. The *MD* factor is actually also application-dependent, but the application sensitivity is significantly smaller than that observed in *AD* calculations. Therefore, in trying to quantify the inherent error tolerance of the PERFECT application suite, we limit out attention to the *AD* factor in this chapter.

The Application Fault Injection (AFI) tool, which we describe in Section 4.3, provides an estimate for the application-level derating (P_{ad} shown in Figure 4.1) and when combined with the machine-level derating values, the overall FIT estimation bound tightens to a more realistic bound. Typically, the combined (*MD*+*AD*)-driven FITs can easily decrease by a factor of 10, relative to an overly conservative system in which only the *MD* component is considered. As such, leveraging the masking effects provided inherently by the application (modulo the level of compiler optimization used) is a promising approach in low power embedded processor system research.

4.3 Application Fault Injection

The core component of application fault injection (AFI) is a software-implemented fault injector (SWIFI). Figure 4.3 shows the high-level block diagram of the fault injector in AFI, which exploits the standard debugging interface available in all POSIX operating systems, like Linux and Unix. The injector emulates hardware faults that are propagated into application level by alternating program-visible architectural states, such as general-purposed registers. The behavior of injector is fully parameterized and configurable to emulate various fault models such as single bit-flip in a register, multiple bit-flips in a register, multiple bit-flips among registers, etc.

AFI provides a mechanism by which a parent process can launch and control the execution of

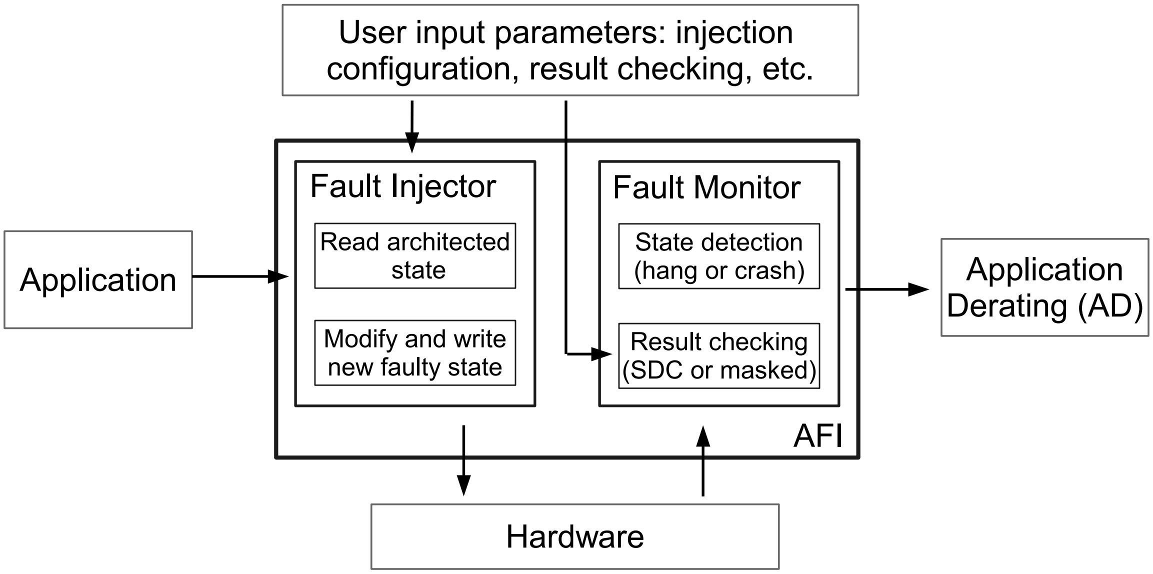

User input parameters: injection configuration, result checking, etc.
Fault Injector
Read architected state
Modify and write new faulty state
Fault Monitor
State detection (hang or crash)
Result checking (SDC or masked)
Application
Application Derating (AD)
AFI
Hardware

Figure 4.3: Block diagram of Application-Level Fault Injection framework (AFI). AFI includes two major components: 1) a fault injector which launches an application and injects error guided by user input, and 2) a fault monitor which determines the outcome of injected error into one of four outcomes as masked, SDC, crash, and hung.

another process – namely the application execution process that is targeted for fault injection. It randomly choose the location within architectural states of each injection uniformly to inject bit-flips. While, users can also specify a specific location (e.g. register number) to investigate application's resilience to a specific register. Within the chosen location, the application fault injector then flips either a single bit at or multiple consecutive bits from a random bit location. The parent process can examine and change the architectural register and memory state of the monitored application execution process.

AFI is accelerated by running fault injection trials in parallel. Typically, AFI needs to run thousands of injection trials to achieve statistical significance. This adds up to a significant amount of time to finish all trials for a single characterization run. Conventional approaches use clock-timer to trigger fault injection. When running injection trials in parallel, CPU time is not reliable since the execution time of the application will be stretched due to resource sharing among multiple running instances of the application. Instead of relying on CPU time, AFI exploits dynamic instruction count to trigger its injection. AFI takes advantage of performance counters that are available in most of modern processors (e.g., POWER7), and triggers its injection when an overflow occurs on dynamic instruction count by a pre-defined value (either user-specified or randomly generated by AFI).

AFI is further augmented to selectively inject faults into region-of-interest (ROI) of an application through source-level annotation. One of the most useful cases is to rule out the initialization and finalization stages of applications from fault characterization. For most of the applications, there application stages usually involve with file I/O operation to read/write data from/to files. They do take considerable of time especially for kernel-like benchmark applications where the total execution time of the main loop is relatively short. Therefore, they contribute to a lot of noise to the fault characterization when only the main loop is of interest.

AFI has been integrated with a user-friendly graphic user interface to configure fault injection experiments as well as visualize results. This provides an intuitive way for user to quickly explore the resilience characteristics of target applications. Advanced users can also take advantages of the scripting facility to automate large amount of experiments for design space explorations.

Pseudo-random fault injection experiments were made into the architectural register space in each controlled experiment. AFI then determines the outcomes of each such injection. Possible outcomes are defined as follows:

Masked The bit that is flipped via the injection has no effect on the program execution profile, including the final program output or final data memory state. These fault injections are categorized as being fully "masked".

SDC The injected bit-flip results in a silent data corruption (SDC) i.e. the program output or the final data memory state shows an error, when compared to the fault-free, "golden" run of the program.

Crash The injected error results in a program crash, where the operating system terminates the program due to a detected runtime error (e.g. divide by zero exception or segmentation fault, illegal memory reference, etc.).

Hung The injected bit-flip results in a "hung" state, where there is no forward progress of the program execution; effectively, in practice such a situation would require a user-initiated program termination or even a physical machine reboot.

4.4 Experiment Setup

We have implement AFI framework on a POWER7 machine running a Unix-class operating system, called AIX^{TM}. AFI takes advantage of hardware performance monitor API (*pmapi*) provided by AIX to interface the performance counters. All applications are compiled using IBM XL C compiler suite.

4.4.1 PERFECT Suite

To demonstrate the capability of AFI, we choose a set of application from a newly developed PERFECT Suite, which is a class of mobile embedded applications that are currently under study in the DARPA-sponsored PERFECT program.[1] A key application domain of interest is that of mobile embedded systems of the type captured by Figure 4.2 and associated discussion at the beginning of this chapter. The suite is divided into four major domains. Each domain contains three kernels that represent segments of computational importance for a representative application on the domain. Applications are detailed as follows:

4.4.1.1 PERFECT Application1 (PA1)

PA1 applications are image processing related kernels that work on massive data sets. The three kernels are:

2dconv is a kernel applying Gaussian filter over the input image or signals to reduce noise in images. The input image uses 32-bit unsigned pixels with a dynamic range of 16 bits per pixel.

[1] PERFECT stands for Power Efficiency Revolution For Embedded Computing Technologies.

dwt53 is a discrete wavelet transform kernel similar to the Fourier transform, except that it captures both frequency and temporal information. *pa1.dwt53* can be used as a pre-conditioner to data compression in areas of image and video processing.

histo is histogram equalization which increases the contrast of an image by spreading the most frequent intensity values over a larger range.

The results of these kernels are presented in the form of an integer matrix. The correctness of the results is verified by comparing the output matrix against the golden outputs.

4.4.1.2 Space-Time Adaptive Processing (STAP)

Radar systems on an airborne platform must often mitigate the impact of ground clutter (i.e., radar returns from the ground or structures), which can overwhelm other signals of interest. Airborne movement further complicates the mitigation process because radar returns are spread in the Doppler dimension. Space-time adaptive processing (STAP) algorithms address clutter by adaptively computing and applying filters in both the spatial (angular) and temporal (Doppler) domains. The included three kernels are:

oprod calculates outer product to estimate co-variance for a given range cell and Doppler bin in order to calculate the associated adaptive weights.

syssol is a linear system solver taking the output of *stap.oprod* to generate weight using QR decomposition.

iprod calculates inner product to apply adaptive weighting to the snapshot vectors to form the final STAP output.

All three kernels have the results in the form of an array of complex numbers. To evaluate the results, these kernels have embedded signal-to-noise ratio (SNR) calculation against the golden output. To verify the correctness of the results, SNR values are compared against non-injection runs.

4.4.1.3 Synthetic Aperture Radar (SAR)

Synthetic aperture radar (SAR) is a radar-based imaging modality capable of producing high-resolution imagery from an airborne platform. Rather than utilizing a large aperture to achieve high resolution, SAR synthesizes a large aperture using platform motion and then forms an image using data corresponding to the pulses acquired over the synthetic aperture. The amount of data needed for image formation is a function of the radar system parameters, desired image resolution, and coverage area. The included three kernels are:

pfa1 and pfa2 are two interpolation kernels related to the polar format algorithm (PFA), which is a Fourier-based approach. The *sar.pfa1* is a PFA range interpolation, while *sar.pfa2* is a PFA azimuth interpolation.

bp is the back-projection algorithm [23,24,30], which is a method of image formation that directly integrates a contribution from each pulse into each pixel.

Similar to STAP kernels, SAR kernels present the results as an array of complex numbers. The correctness is verified via SNR comparisons against non-injection runs.

4.4.1.4 Wide Area Motion Imagery (WAMI)

Modern imaging sensors produce a very large amount of data from which useful information must be extracted. The included three kernels are:

debayer is the debayer kernel that takes a Bayer array image with one sampler per pixel as input, and returns an image with RGB samples per pixel where the missing colors have been estimated via interpolation. The algorithm is detailed in [51]. The result of *debayer* is verified via exact match of each pixels. Any single mismatch of a pixel will be characterized as wrong results.

lucas is an image alignment kernel which moves and warps a template image to minimize the difference between the template and the original image. The algorithm is detailed in [3].

Domain	Application	Result fidelity metric
PA1	2dconv dwt53 histo	bit-to-bit correctness to predefined golden outputs
STAP	oprod syssol iprod	signal-to-noise ratio (SNR)
SAR	pfa1 pfa2 bp	signal-to-noise ratio (SNR)
WAMI	debayer	pixel-by-pixel comparison to predefined golden outputs
	lucas	bit-to-bit comparison on checksums
	cd	pixel-by-pixel comparison with acceptable mismatch threshold of 0.1%

Table 4.1: Result checking mechanism of PERFECT applications

The result of *lucas* is presented as an 6x6 Hessian matrix of floating point numbers with 7 significant bits. The correctness of the results is verified via exact match to the golden output matrix.

cd is a change detection kernel using Gaussian mixture model as described in [78]. The results are compared to the golden output pixel-by-pixel. The result is considered to be incorrect when more than 0.1% of pixels mismatch the golden output.

The result checking mechanism of all benchmarks within PERFECT Suite is summarized in Table 4.1

4.4.2 Statistical Significance

In order to achieve statistical saturation of the masking rate, we have done a sensitivity study on the number of faults injected using well known benchmarks such as bzip2, matrix multiply (mmm), and conjugate gradient (CG). As shown in Figure 4.4, the masking rate saturates after 3000 injections. Therefore, we choose 3000 injections for the rest of our studies.

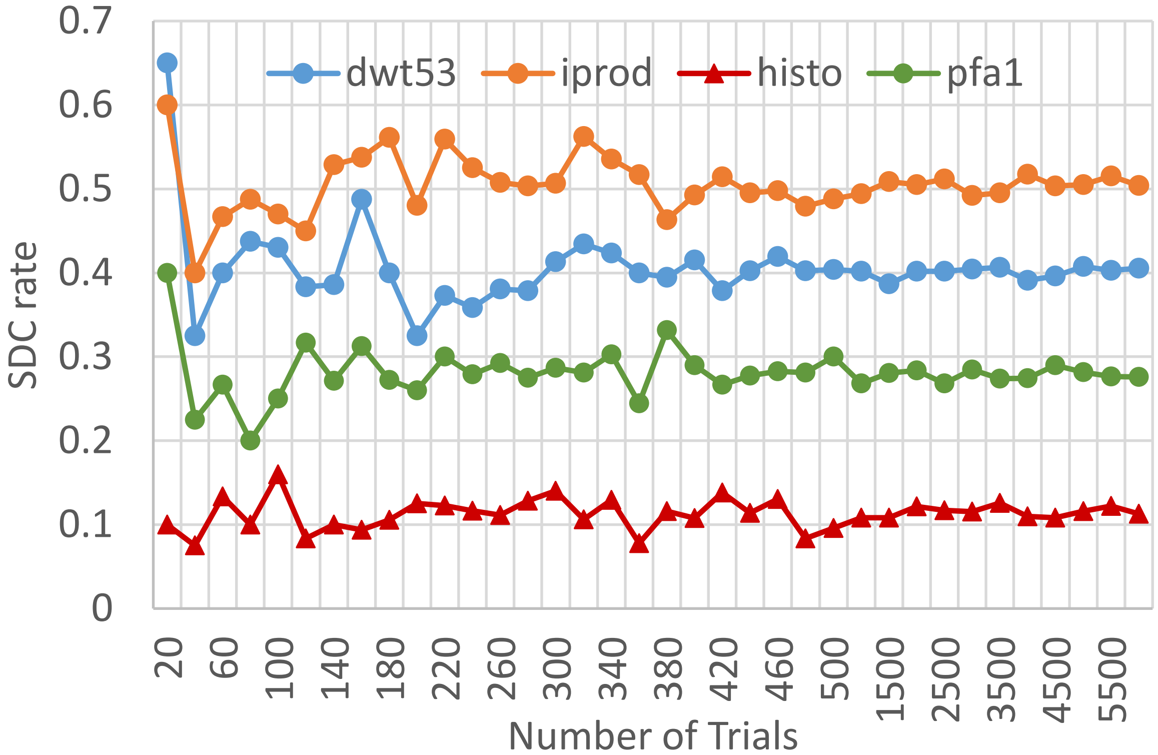


Figure 4.4: Sensitivity study on the number of injections to achieve statistical stability. X-axis is the number of injections, while Y-axis shows the masking rate of corresponding experiments. Three well known benchmarks are choose as conjugate gradient (CG-A) from NAS benchmark suite, bzip2 from SPEC2006, and matrix multiply (mmm).

4.5 Resilience Characterization

We show a couple of resilience characterization cases to demonstrate the capability of AFI framework.

4.5.1 Case I: Single Bit Errors

A single bit error in architectural states, e.g. general-purposed register file, is one of the most used fault model to emulate a single-event upset (SEU) induced by high-energy particle strike. Using AFI, we inject a single bit-flip into a general-purposed register that is randomly chosen at runtime. As shown in Figure 4.5, the SDC rate in response to the bit-flip varies widely across applications, ranging from 2% in the case of *lucas* and *histo* to as large as around 50% in the case of *iprod*. There are two general observations that we make in connection with the reported (potential) SDC manifestation rate (or probabilities) within an application. These observations actually apply to both general purpose and floating point register injection experiments (discussed later in this section).

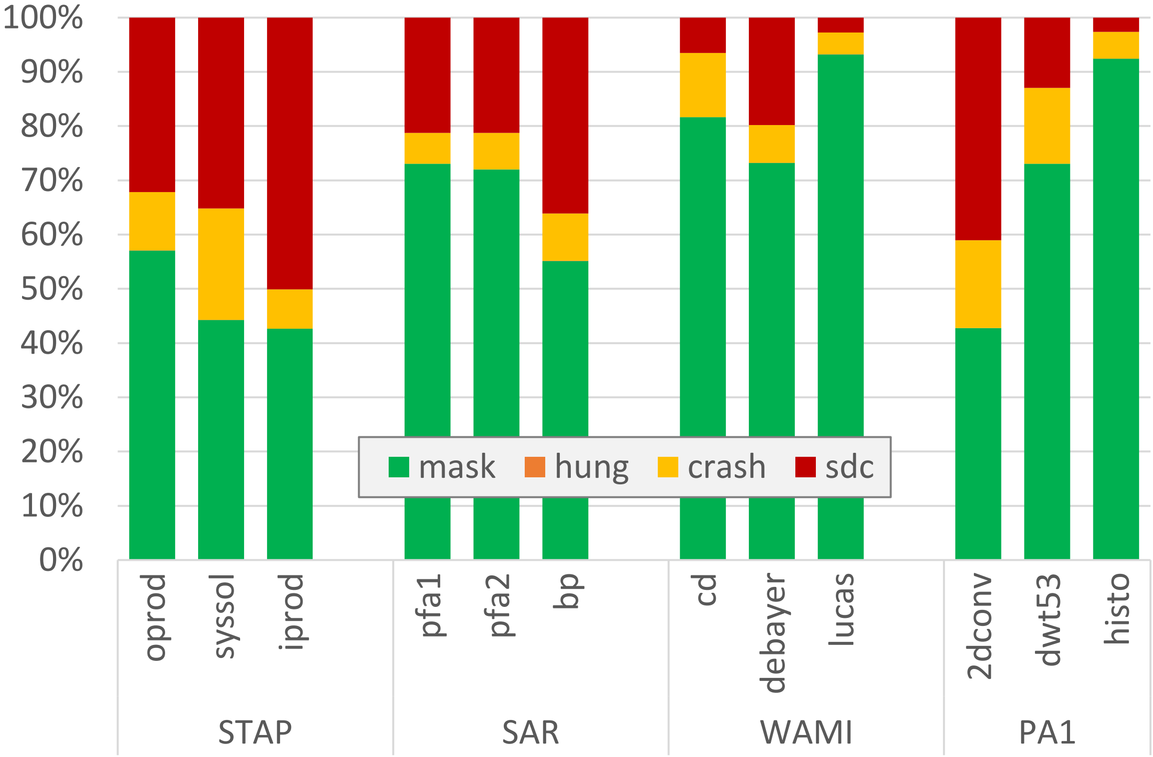


Figure 4.5: General-purpose register injections

Register Usage : The more registers used in the binary, the less resilient the application is. For example, in the main loop of *stap.iprod*, the binary code generated by the compiler uses most of the available architectural registers. Therefore, a single bit error within the architectural register space is more likely to affect application correctness.

Interpretation of SDC : It must be noted that although we refer to SDC rates of an application - we actually limit our attention only to the *AD* factor, in order to focus on the application-specific sensitivity aspect. As discussed before, multiplying by the *MD*-related probability and the raw SER incidence probability yields the net real probability of SDC error - which is a much, much smaller number than the *AD*-related probability alone.

AFI is also capable of injecting bit-flips into other architectural stats, such as floating point registers, as well. As shown in Figure 4.6, when injecting a single bit-flip in floating point architectural state, the SDC rate, in general, are much smaller than injections in general-purpose architectural state. This is because an error in floating point architectural state only affects floating point instructions, which are only a faction of total instruction mix. In particular, floating point registers are not involved in address computation related instructions - unlike general purpose registers.

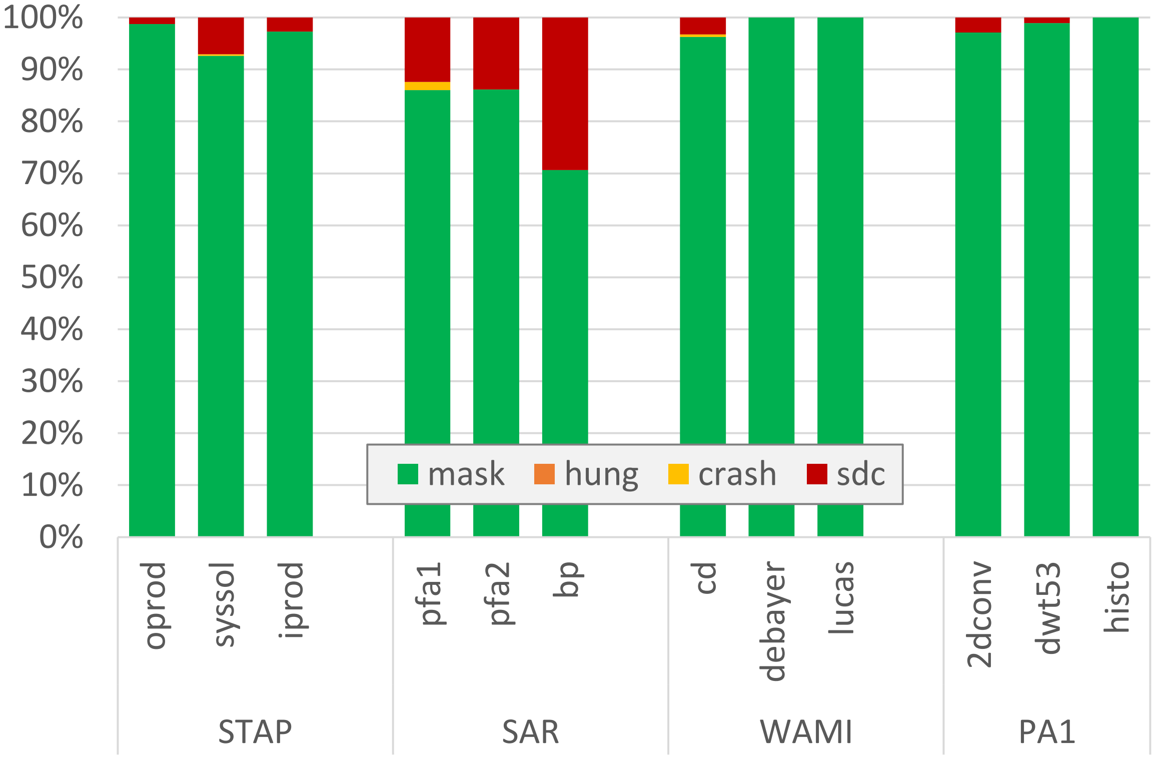


Figure 4.6: Floating point register injections

We can see from Figure 4.7, SAR applications (*pfa1, pfa2, bp*) have relatively high floating point operation percentages, therefore, they have the highest SDC rates among the floating point applications under consideration. There are two exceptions for the correlation between SDC rates and floating point instruction percentage, *cd* and *2dconv*. In the case of *cd*, the final stage of the computation generates residual checking values as the application's output, which masks most of the incorrectness in previous calculations. While for *2dconv*, all floating point operations are dedicated to calculate the accumulating sum using a single variable, so that the generated binary only uses very few floating point registers. As a result, the overall SDC rate is limited even though a

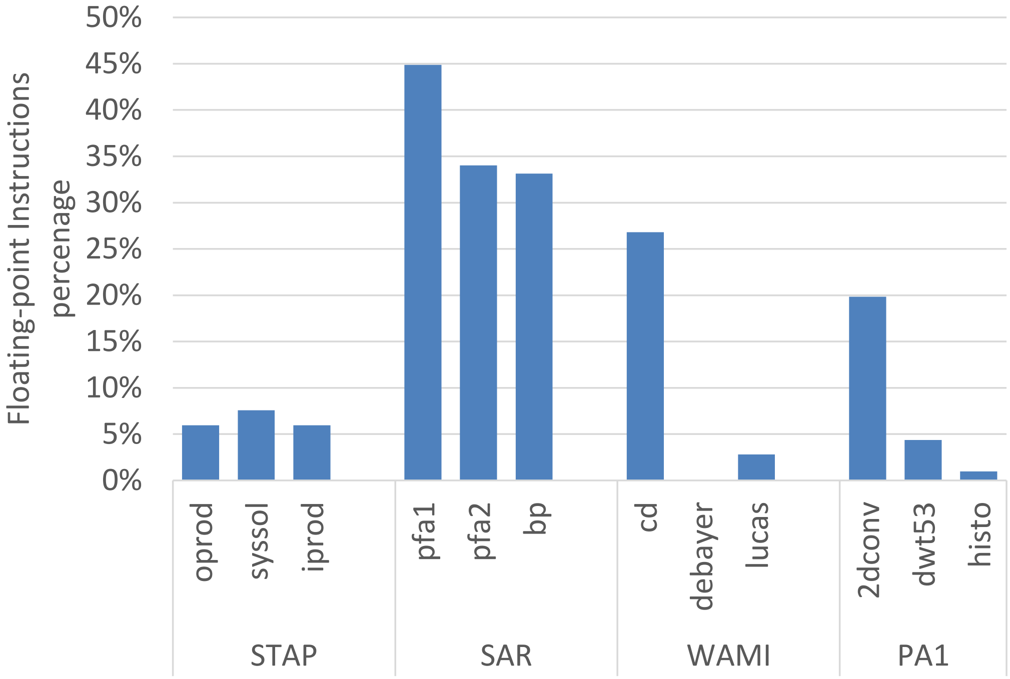


Figure 4.7: Floating point instruction percentage

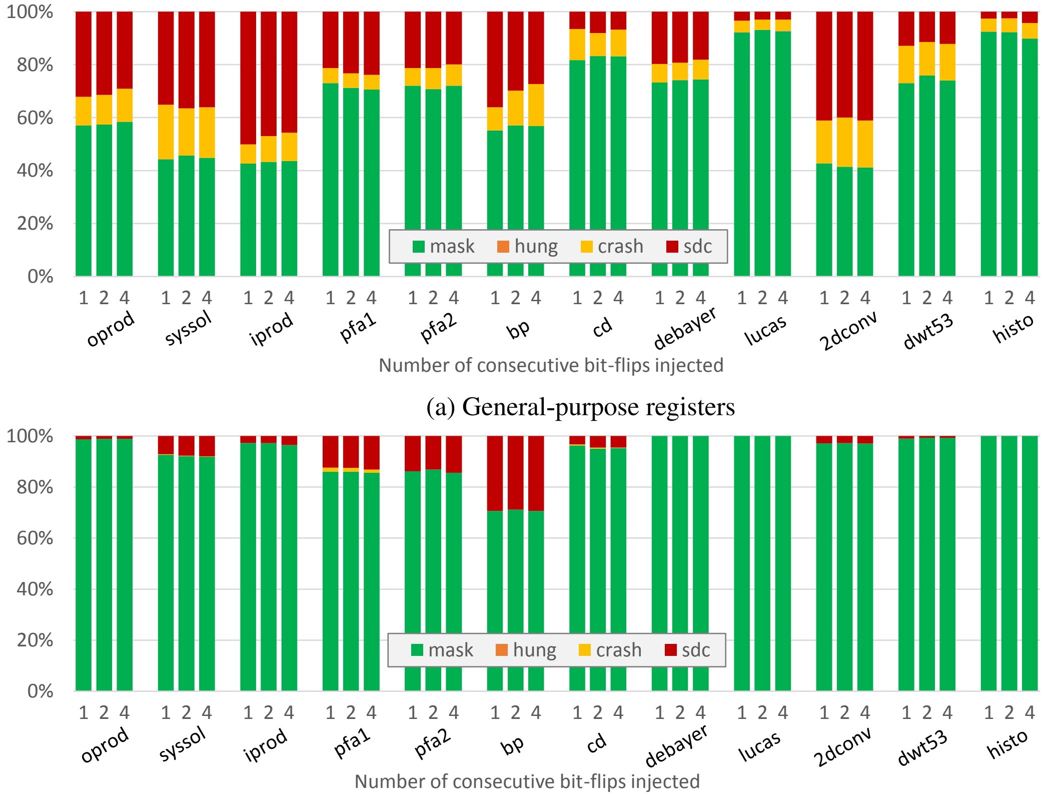
considerable number of instructions uses floating point operation.

4.5.2 Case II: Multiple Bit Errors

The multiple bit error injection models were primarily inspired by SEMU physical trend data, as shown in Figure 4.12. Also, recent statistical fault injection based analysis by Cho et al. [15] shows that random single bit-flips at the latch level (corresponding to an SEU) may not result in just a single bit-flip corruption at the architectural register state. Regarding to these impacts on architectural states, AFI supports different multiple bit error injection models.

We first look at a straightforward multi-bit injection model, where error bits are consecutive within a single architectural register. In this injection model, AFI randomly chooses a single architectural register and independently a random bit location, then flips multiple bits starting from the chosen location in the chose register.

In Figure 4.8, we show resilience results for all applications by injecting bit-flips into general-purpose registers (Figure 4.8a) and floating point registers (Figure 4.8b), with a single bit, 2 bits,



(a) General-purpose registers

(b) Floating point registers

Figure 4.8: Multi-bit error injections.

and 4 bits, respectively.

For GPRs in Figure 4.8a, in general, we do not see significant changes in masking rates. This is because when errors hit a critical register, it almost always leads to some form of errors, no matter how many bits the error is. Besides, out of these unmasked trials, we can observe a higher SDC rate with 1-bit error injection while a higher crash rate with 4-bit injection.This is because with a single bit-error, the affected address pointers used by an application may still be legitimate but point to a wrong place (hence SDCs). However, with 4-bit errors, the affected address pointers will more likely point to an illegal address (hence crash). We observe a similar trend for FPR injections (Figure 4.8b). The masking rates do not change much across 1-, 2- or 4-bit errors. Since the crash and hang trials are very rare with errors in FPRs, the SDC rate is roughly equal to $1 - mask\_rate$. As a result, SDC rates do not vary too much with multi-bits errors in FPRs.

We then look at an alternative multiple bit error injection model: for an N-bit error injection experiment, we randomly choose N consecutive architectural registers and an independently random bit location. Then, AFI flips N consecutive bits starting from the chosen location within each of these N registers. In this model, a total of N^2 bits are flipped in a single experiment. This injection model is used to mimic the single event multi-bit upset (SEMU) event, where a single particle strike can flip multiple bits within a register file region.

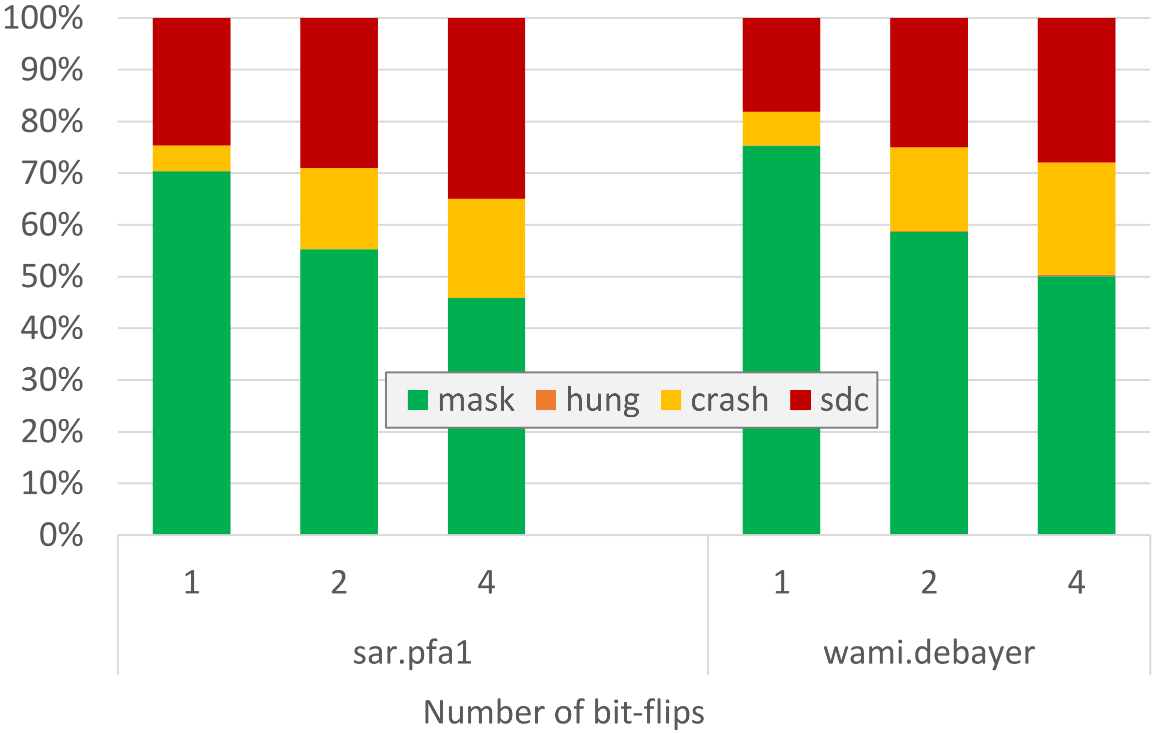


Figure 4.9: Results for alternate multi-bit error injection model (GPRs) to mimic the SEMU case.

We choose two applications: *sar.pfa1* and *wami.debayer*. For each application, we inject 1, 2, and 4 bits errors into general-purpose architectural register state using the alternative multiple bit error injection model. As shown in Figure 4.9, as the number of error bits increases from 1 through 4, masking rates of both applications decrease significantly while crash and SDC rates increase significantly. This is because more registers are affected under the alternative multiple bit error injection model, which increases the likelihood that a critical architectural register is corrupted.

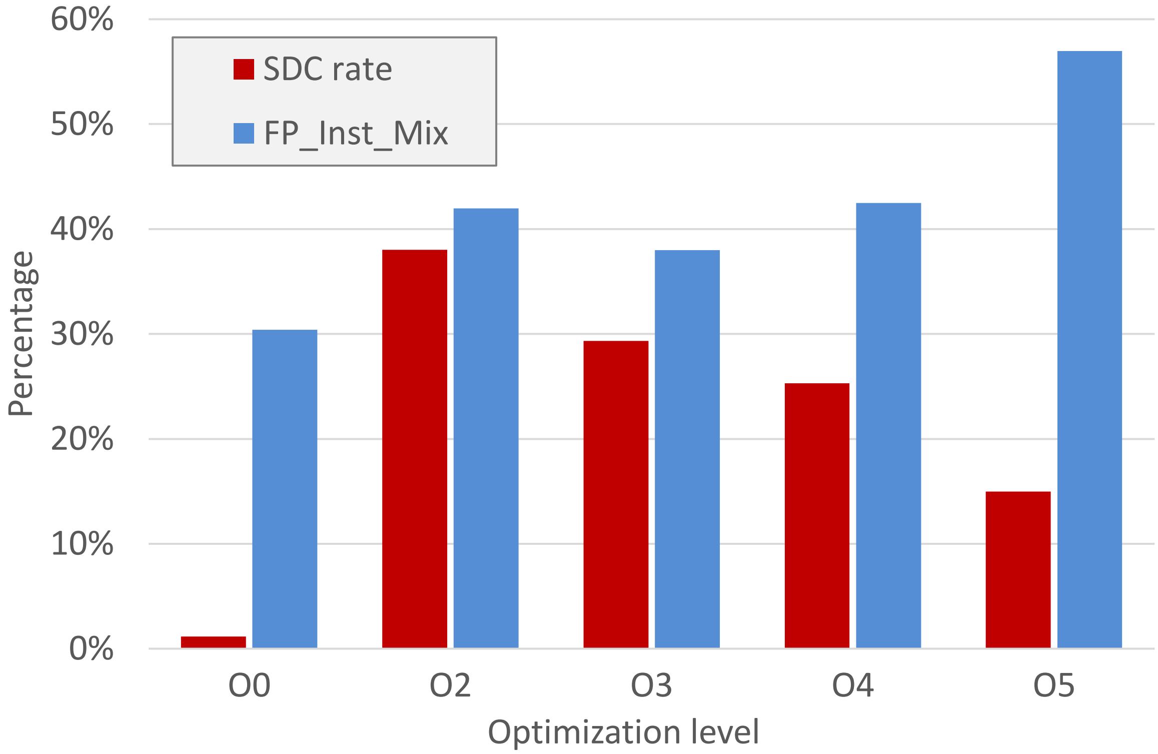
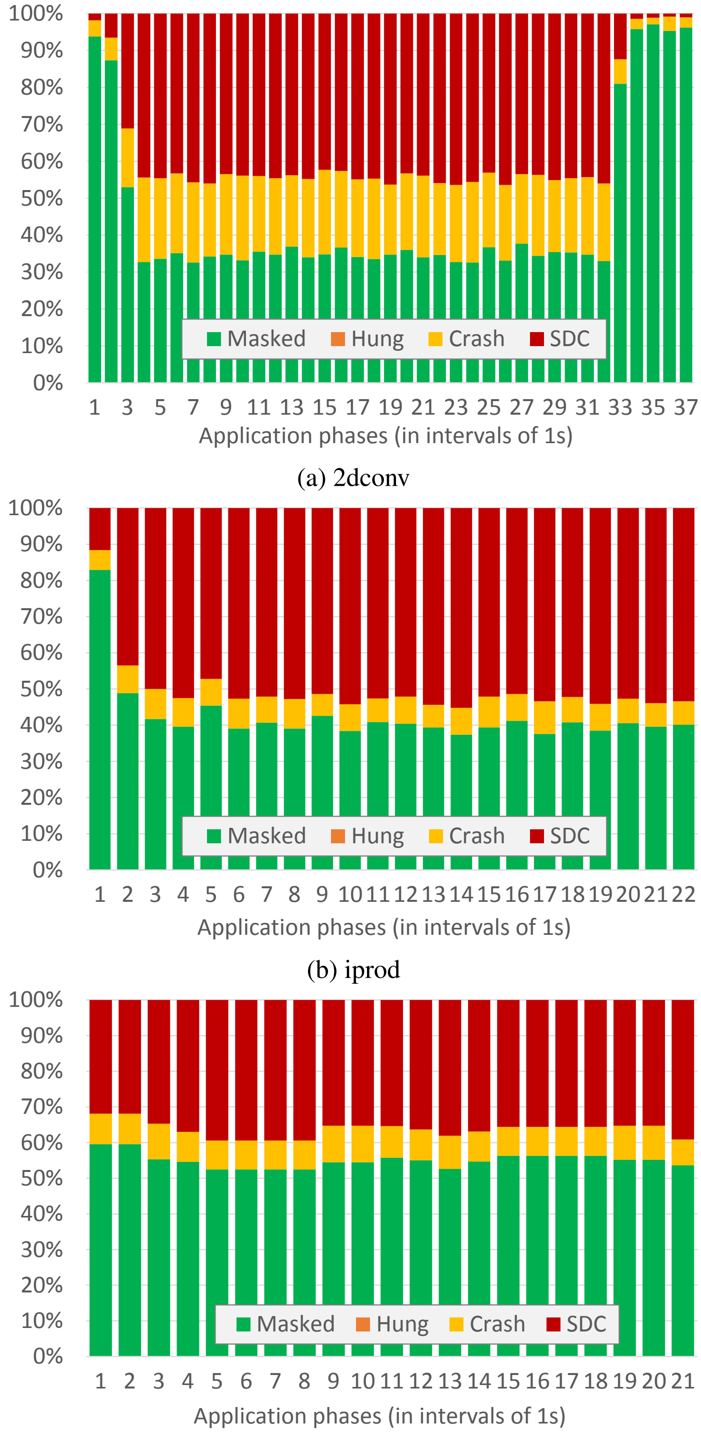


Figure 4.10: SDC rates of *sar.bp* with different compiler optimization levels

4.5.3 Case III: Compiler Optimization

AFI can also be used to study the resilience of the same application, but compiled with different optimization flags. In Figure 4.10, we show the SDC rates of *sar.bp* compiled with five different optimization levels (O0, O2, O3, O4, and O5) subject to a single bit-flip in FPRs. Along with bars of SDC rates, we have added bars for the dynamic or run-time floating point instruction percentage of the corresponding binaries. We observe that the SDC rate increases significantly from optimization level O0 to O2, then gradually decreases through O5. Meanwhile, the percentage of floating point instructions follows a similar trend from O0 through O3 (increase from O0 to O2, and decrease from O2 to O3). From O0 to O2, the increase of SDC rate is much larger than the increase of floating point instruction percentage. This is because, the compiler, at O2, does loop unrolling optimization which increases the usage of floating point registers a lot.

From O3 to O5, the SDC rate decreases while the floating point instruction percentage actually increases. This is because, starting with O3, the compiler starts optimization of vectorization and



(a) 2dconv

(b) iprod

(c) bp

Figure 4.11: Phased characterization on (a) pa1.2dconv, (b) stap.iprod, and (c) sar.bp

tries to use vector registers as much as possible. In this experiment, we inject bit-flips into FPRs only. Since FPR is part of the vector register file in POWER architecture, our injection only affects part of the vector register file. Therefore, O5 binary which has the most extensive utilization of vector instructions has the lowest SDC rates as shown in Figure 4.10.

4.5.4 Case IV: Temporal Vulnerability

We show phased characterization for three of the applications from PERFECT Suite in Figure 4.11. In Figure 4.11a, the SDC rates are rather low in both initialization phases and finalization phases. This is because *2dconv* is doing file reading and writing during these two phases respectively. In Figure 4.11b, only the initialization phases have low SDC rates. This is because *iprod* does calculation of checksum in its finalization phases, which are still vulnerable to SDC errors. Lastly, in Figure 4.11c, SDC rates tend to be roughly constant across all phases. This is because *sar.bp* reads a relatively small input file so that the initialization phase is far less than 1 second. During finalization phases, *sar.bp* does similar checksum calculation as *stap.iprod*. Therefore, these phases are vulnerable to SDC errors as well.

4.6 Discussion of Characterization Results

The application-level statistical fault injection studies for the PERFECT Suite reported in this chapter yield a couple of key conclusions:

- If single-event upset (SEU) events result in at most a single bit-flip in a single register file, then the results in Figure 4.5 and 4.6 accurately depict the *AD*-related masking characteristics of the general purpose and floating point register set respectively. Both figures show considerable application-sensitivity. For example, in the GPR case, SDC vulnerability can range from 2% (*histo* or *lucas*) to 50% (*iprod*).
- The temporal, phase-specific profiles of three selected applications (Figure 4.11), under the same single bit-flip error injection model yields interesting data that can be easily explained

by understanding the phase-specific functionality within each phase. For example, initialization, computational steady-state and final summarization steps often show up with distinct resilience characteristics.

- Alternate multi-bit error injections model were tried, with the goal of mimicking the effect of a single event multi-bit upset (SEMU) scenario. In one scenario, multiple (successive) bits within a single randomly selected register were flipped in a given experimental run. In another scenario, multiple (successive) registers were picked, and multiple bits across the selected registers were flipped in a given run. In the first scenario, the difference in masking and corruption characteristics across 1-, 2- and 4-bit injections was not much (Figure 4.8); whereas, in the second scenario, when multiple registers were picked to flip bits at the same time, a significant difference was found across 1-, 2- and 4-bit injections (Figure 4.9). This was easily explainable by considering the fact that a single corrupted register is expected to cause similar effects within the program, regardless of whether one or multiple bits are changed. Whereas, if multiple registers get corrupted, the chances of downstream inaccuracy of computed results become higher with the number of simultaneously affected registers.

- Compiler optimization level was found to be a strong influence in determining the resilience characteristics of a given application (Figure 4.10). In general, higher degrees of optimization tend to increase performance, but at the cost of increased vulnerability to transient errors. One key reason why we see such a behavior is that code optimization tends to use more registers than in un-optimized code - and this increases the probability of downstream failures in response to random register bit-flips.

- As discussed in section 4.5, many of the experimentally observed trend data can be explained quite easily by considering simple, machine-independent workload metrics like dynamic instruction frequency mix. This is a very useful result, since it implies that in a dynamic (adaptive) resilience framework, monitoring of instruction mix profiles (via performance counter readings) can prove to be valuable in effecting appropriate control changes (e.g. voltage-frequency settings) in order to optimize performance per watt, without giving up system resilience.

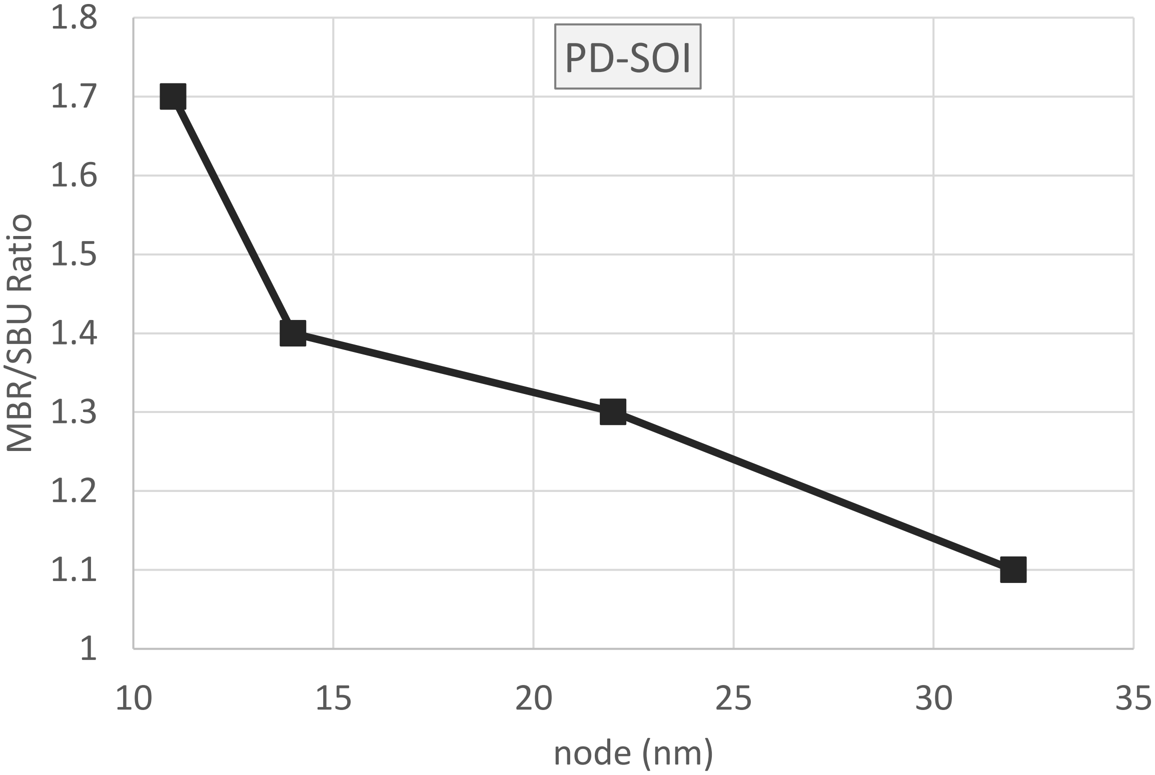


Figure 4.12: Single event multi-bit upset (SEMU) trends for CMOS PD-SOI [66].

The multi-bit error injection models were primarily inspired by SEMU physical trend data, as shown in Figure 4.12 (quoted from [66]). Also, recent statistical fault injection based analysis by Cho et al. [15] shows that random single bit-flips at the latch level (corresponding to an SEU) may not result in just a single bit-flip corruption at the architectural register state. Also, critical control registers outside the programmer-visible architectural state could get corrupted. Cho et al. have justifiably argued that conclusions about system-manifested net resilience or vulnerability levels cannot be drawn from naively directed application (or architectural state) fault injection experiments. Therefore, we experimented with a range of alternate single- and multi-bit error injection models in this research. Regardless of the particular choice of the error injection model, we could not observe any change in the relative ordering of the applications from ”highly resilient” to ”least resilient.” This is an important result - but we acknowledge, this is limited to considerations of the application derating (*AD*) component of the net SER equation only. If the application-sensitivity of the machine derating (*MD*) component were also considered, perhaps the resilience-directed relative application ordering would change, depending on the choice of error injection model. This, therefore, remains an important area of future research.

4.7 Conclusions

In this chapter, we describe a framework called AFI for application programmer and system designer to take advantage of application-level resilience. Using AFI, we have characterized 12 applications from PERFECT Suite, a benchmark suite for emerging embedded systems. The results shows a wide range of application-level resilience (SDC vulnerability rates from about 2% though more than 50%), which motivates further dynamic techniques to trade-off among resilience, power and performance. We also find that application-level resilience is very sensitive to the compiler optimization, where the SDC vulnerability rate is around 1% without optimization (or "-O0"), and increases to more than 35% with "-O2" optimization level.

Chapter 5

PEARL: Power-Efficient Embedded Processing with Resilience and Real-Time Constraints

A powerful control parameter for power management of embedded processor systems is dynamic voltage-frequency scaling (DVFS). However, soft error rates (SERs) are known to increase sharply as the supply voltage is scaled downward [70]. Hence, in order to preserve system resilience levels, it is important to apply voltage scaling carefully, keeping in mind the varying levels of vulnerability to SER within an application's execution profile. On the other hand, overclocking or turbo-boosting (with higher voltages applied if/as necessary) to meet real-time deadline, comes at the cost of higher power (or current) density and temperature (as well as higher gate-oxide field stress), which results in higher hard-failure rates.

In this chapter, we consider a class of embedded systems that require high levels of power-performance efficiency while meeting mission-critical reliability specifications and real-time performance targets. Such systems require an energy optimization protocol that is cognizant of the variable resiliency needs and properties of the executed application. A representative example of such an embedded system of interest is a single unmanned aerial vehicle (UAV) or a swarm of such UAVs. These are typically engaged in remote sensing of ground images, for the purposes of reconnaissance, object recognition/tracking and tactical response.

We first describe PEARL, a novel software modeling framework that enables users to: (a) *statically* prepare application workflows for energy-optimized resilience; and (b) experiment with *run-time* deployment options in targeted embedded systems. PEARL stands for power efficient and resilient embedded processing with real-time constraints, and is built upon the earlier toolset described by Wang [86]. We then describe the use of PEARL to pursue *static* optimization of voltage-frequency settings across segments of a workflow. We handle both linear and directed acyclic graph (DAG)-structured workflows. Subsequently, we describe how *dynamic* (run-time) uncertainties are factored into the user-driven workflow optimization process. Our analysis shows up to 15% and 35% energy efficiency improvement over a simple baseline, for linear and graph workflows respectively. Moreover, our proposed dynamic iterative approach achieves up to 25% energy efficiency improvement over a conservative static approach, while maintaining the same level of confidence in meeting deadline constraints.

This work has been published at ISLPED 2015 [88].

5.1 Related Work

Dynamic voltage and frequency scaling (DVFS) has been widely used for managing workload-driven power in a processor or system context. For example, Isci et al. [39] propose multi-core DVFS algorithms with the objective of maximizing chip throughput for a given power budget. In the real-time embedded systems domain, early work by Pillai et al. [61] and more recent work by Devadas et al. [25] and Qi et al. [63] are a few representative studies from a large body of work to meet real-time deadlines while minimizing power. Scheduling of tasks in the form of DAG has also been widely studied, for example, in Shang et al. [72] and Kanoun et al. [42]. However, none of them considered reliability as a system-level design constraint.

Combining reliability considerations with energy savings in a unified dynamic resource management framework is a topic area that has not been explored as widely as dynamic power management alone. Zhang et al. [94] address a reliability-aware power management problem, but in this case the reliability focus is only on checkpoint-restart based systems. Similar to our work, Zhao et al. [95] address the mutually opposing issues of energy efficiency and transient error probability (with DVS or DVFS control). However, Zhao et al. use a set voltage-dependence equation to model system resilience, without factoring in the application-level masking effects (as in our work). EPROF [93] exploits Integer Linear Programming (ILP) to optimize the trade-off among energy, performance, and reliability. However, the trade-off in EPROF targets heterogeneous multi-core systems composed of reliable cores consuming high power, and unreliable cores consuming low power. Performance, and reliability are modeled as intrinsic properties of hardware using proxies such as clock frequency and soft error rate. Our work relies on voltage-frequency scaling as the primary energy saving technique, and takes advantage of application-specific models for power and reliability. Finally, our work presents an approach to dynamic adaptation that accounts for run-time variations, which is not covered by EPROF. As in our static optimization work, the authors of [71] propose a similar linear-optimization based approach, while emphasizing the impact of application level correctness to the system resilience. However, in their work, the method of characterizing the application level resilience is SoC-specific, and does not apply to general-purpose processors, as studied in our work. In [86], Wang et al. present the idea to exploit ILP for linear workflow optimization. Our work extends this to more complex graph workflows, and provides a better dynamic approach that can achieve higher energy efficiency while still maintaining a high level of confidence in meeting deadline constraints.

5.2 PEARL Facility and Methodology

5.2.1 Workflow Models

We use the term *workflow* to denote a collection of inter-dependent tasks as executed by a single mobile embedded system (e.g. UAV) or a swarm thereof. The basic building block of a workflow is an application or a task. For simplicity, we stipulate that any workflow-specific voltage-frequency management decision is applied only at the beginning of an application segment. Within an application, the voltage-frequency setting remains fixed.

A linear workflow (Figure 5.1a) is used to model a single embedded system, where a sequence of applications is ordered by inter-application dependencies. A graph workflow (Figure 5.1b) is used to model multiple embedded systems, across which inter-communications are required in order to solve the mission-assigned task cooperatively. Each embedded system runs its own thread of linear workflow, while applications within that linear workflow are also subject to dependencies from other embedded systems, known as inter-dependencies. Adding inter-dependencies, threads of linear workflows form a directed acyclic graph (DAG). For simplicity, we only consider dependencies across applications, and leave the overhead associated with inter-application communication as future work.

5.2.2 Individual Application Programs

We use applications from the PERFECT benchmark suite provided for research in the DARPA PERFECT program [22]. The PERFECT suite includes 12 applications taken from signal and image processing tasks: outer product (*oprod*), system solve (*syssol*), inner product (*iprod*), discrete wavelet transform (*dwt53*), histogram equalization (*histo*), 2D convolution (*2dconv*), path finding algorithms (*pfa1* and *pfa2*), back projection (*bp*), change detection (*cd*), lucas kanade (*lucas*), and debayer (*debayer*). To add diversity to our set of benchmarks, we also select 8 applications from

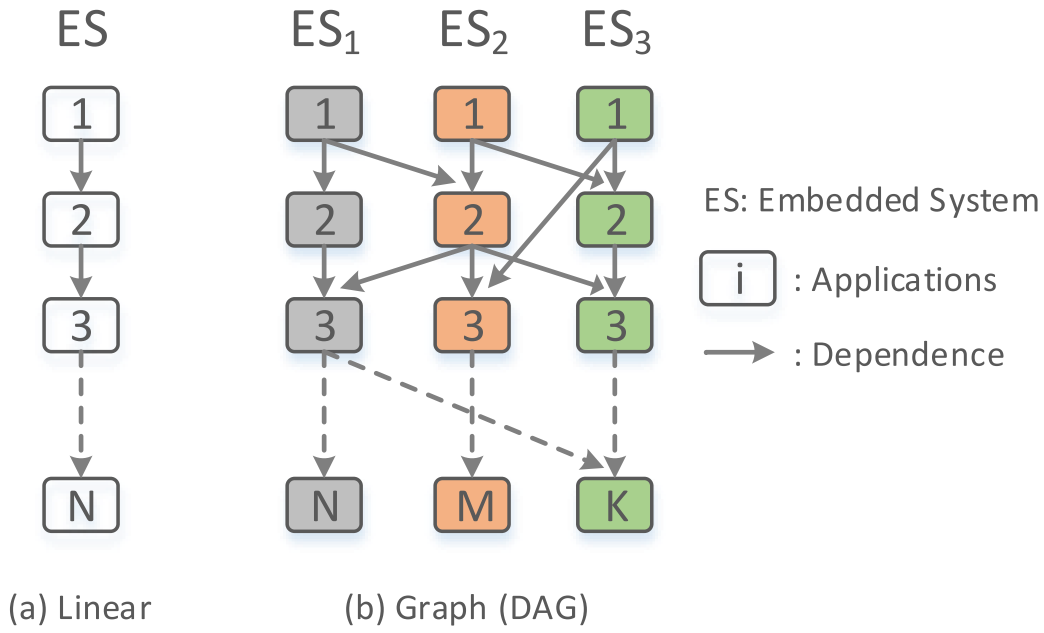


Figure 5.1: Workflow illustration. A linear workflow (a) is a stitched sequence of applications, modeling a single embedded system (ES). A graph workflow (b) is a set of linear workflows with inter-dependencies, modeling multiple embedded systems with inter-system communications.

SPEC2006 in the expectation that embedded systems will have an increasing burden of general purpose applications. Selected applications are *gcc*, *perlbench*, *bzip2*, *h264ref*, *mcf*, *sphinx*, *dealII*, and *lbm*, which constitute an application set with diverse computing domains and characteristics. We will use these 20 applications to synthesize workflows studied later in this chapter.

5.2.3 Power and Performance Modeling

We use machine measurement from real hardware for power and performance characterizations for all the applications mentioned in the last section. The experimental system is a state-of-the-art multi-core server processor system with POWER ISA. For each application, we measure the power and performance (execution time) under 20 frequency levels with a step size of 100MHz. The frequency range covers operating levels such as turbo boosting, nominal operation, and low-power operations with low supply voltage. Note that, in our experimental setup, the appropriate supply voltage is determined by firmware for each frequency level. Therefore, the frequency and supply voltage always come as a pair for voltage-frequency scaling. For simplicity, we will refer to voltage-frequency settings by their corresponding frequency levels. Unlike prior work using analytical models for power and performance characterization, we rely solely on machine measurements, which are more accurate with less modeling errors. The characterization cost may seem high due to the fine granularity of frequency levels for each application. However, the one-time cost can be amortized later by analysis on various workflows composed of the same set of applications.

The length of each application plays a critical role in the voltage-frequency allocation problem. For example, if a single application dominates the workflow in terms of the execution time, it will have an overwhelming effect in determining the energy efficiency of the workflow. The solution, in this case, would be straightforward: just choose the most efficient operation level for the *dominating* application. Therefore, in order to remove such application-specific bias in our illustrative analysis, we scale the execution time of each application so that all of them run for the same length of time (10 seconds) at the nominal supply voltage. This assumption allows us to explore the maximum achievable improvement via full-workflow scheduling. Furthermore, the DVFS transition time (which is in the milliseconds range) is a very small fraction of each application's run time. We ignore the transition overhead associated with DVFS by only allowing DVFS transitions at application boundaries.

Note that, all our presented approaches in this chapter do not rely on any specific value of applications' length of execution time, and can be broadly applied to any real applications of interest in the described scenarios.

We measure the power and performance using "cornflake" [82], which is an externally resident software that can attach to facilities inside the POWER7+ system. Using cornflake, we are able to remotely access both the power and performance metrics associated with a given application run. The performance metrics are accessed via the performance monitoring unit (PMU) facilities within the processor. The power metrics are measured via the AMESTER framework [45], which is a specially developed firmware that is capable of monitoring the power and temperature measurements tracked by the EnergyScale micro-controller [28].

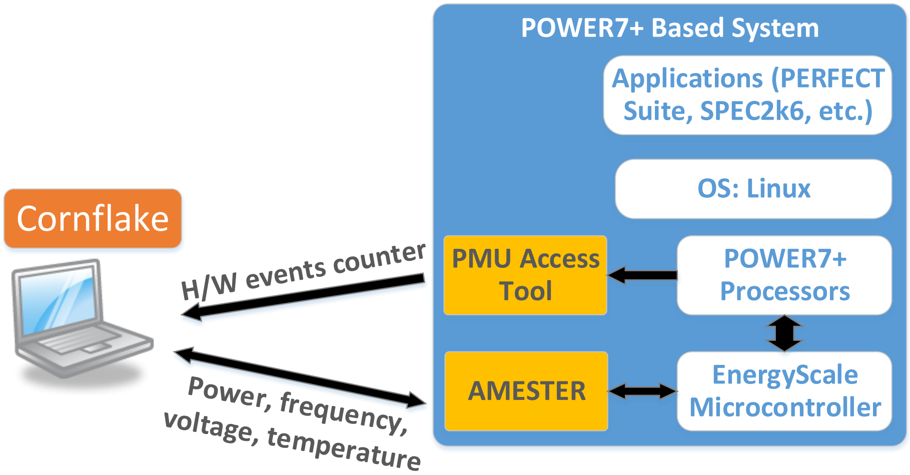
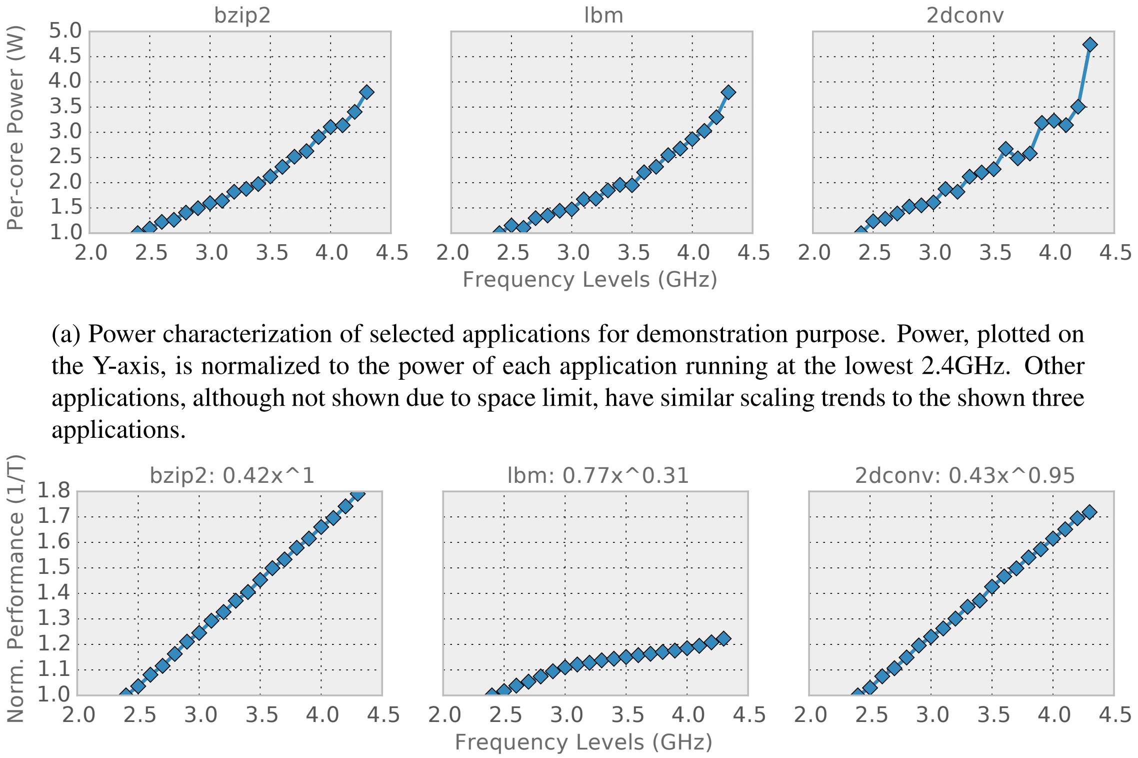

POWER7+ Based System
Applications (PERFECT Suite, SPEC2k6, etc.)
OS: Linux
PMU Access Tool
POWER7+ Processors
AMESTER
EnergyScale Microcontroller
Cornflake
H/W events counter
Power, frequency, voltage, temperature

Figure 5.2: Experimental system block diagram.

We have characterized the power and performance for all 20 applications mentioned in Section 5.2.2. Due to the space limit, we only show the results for 3 selected applications. As plotted in Figure 5.3a, applications' power consumption scales with a similar trend. We confirm that other applications that are not shown in the figure also share similar scaling profile. Execution time, on the other hand, is a little different. To demonstrate the characterization on the scaling of execution time, we fit the execution time to the operating frequency using equation $t = \beta \cdot f^{\alpha}$. As shown in



(a) Power characterization of selected applications for demonstration purpose. Power, plotted on the Y-axis, is normalized to the power of each application running at the lowest 2.4GHz. Other applications, although not shown due to space limit, have similar scaling trends to the shown three applications.

(b) Execution time characterization of selected applications for demonstration purpose. Performance, plotted on the Y-axis, is calculated as the reciprocal of applications' execution time, and normalized to the execution time of each application running at the lowest 2.4GHz. The formula in the title indicates coefficients when fitting performance to frequency levels.

Figure 5.3b, *2dconv* and *bzip2* have the α coefficient value as 0.95, and 1, indicating the two are more computation-intensive applications. While *lbm* has a α value of 0.31, indicating a memory-intensive application. For all target applications, 14 out of 20 have α values greater than 0.9. There are only a few exceptions whose α values are significantly less than 1. The application with the smallest α is *lbm*, where $\alpha = 0.31$.

5.2.4 Resilience Modeling

System reliability, in the context of transient (soft) and permanent (hard) errors, is of critical importance for embedded systems carrying out mission-critical tasks. For soft error rate (SER) modeling, we estimate the failure rate (measured in standard units of failures in time or FITs) using an approach adopted from industrial practice [44,69]. In such an evaluation methodology, machine-level derating (MD) and application-level derating (AD) are treated as decoupled factors. We model the system-level FIT as: $FIT_{SER} = SER_{latch} \times N_L \times (1-AD) \times (1-MD)$, where SER_{latch} refers to the raw per-latch SER; N_L is the number of latches; AD refers to the probability of application-level masking, given a corruption of the program-visible (register and memory) state; and MD is the probability of machine- (or micro-architecture) level masking in the context of a single latch bit upset.

To model SER_{latch}, we empirically fit it to the voltage sensitivity gradients published in [70] using the formula shown below:

$$\mathrm{SER}_{\mathrm{latch}} = e^{\alpha \cdot \mathrm{Vdd} + \beta}$$

where Vdd is the supply voltage, and α,β are fitting constants.

To derive the AD factor for a given application, we use an Application Fault Injection (AFI) tool with a built-in programmable statistical fault injection routine. AFI makes use of the ptrace debugging facility that is available in POSIX-compatible operating systems to put together a framework in which any target application can be compiled and run, under user-controllable fault injection directives. Each injected error in program-visible architectural state leads to one of the four outcomes: 1) *Masked*, where there is no effect on the correctness of program execution or final output; 2) *SDC* or silent data corruption where a deviation of the program output, relative to the golden value is observed; 3) *Crash*, where the program terminates prematurely; and 4) *Hang*, where a hung state is observed such that there is no forward progress in the program execution. The overall AD factor is empirically derived as the "masked" rate as observed from AFI experiments. However, if the concern is mainly on SDC (which is often the case in numerical, compute-intensive codes), the AD factor may be computed as: $AD = 1 - rate_{SDC}$.

MD factors can be estimated using the same approach as used by the authors of Phaser [65], which factors in true data residency statistics for each functional unit. These statistics can be derived from a properly validated, cycle-accurate architectural simulator as explained in [65]. As has been observed in prior work on POWER machines [69], the AD factor tends to dominate over MD by a large factor, especially if the focus is only on SDC. Therefore, for simplicity of analysis, in this chapter we only focus on AD, while effectively assuming that MD is invariant across the class of applications considered for a given (fixed) machine implementation.

In optimizing for power-performance efficiency (e.g. billions of instructions per second per watt or BIPS/W), the tendency is to push towards lower voltage-frequency operating points as much as possible, without violating latency (i.e. deadline) constraints. Therefore, our resilience modeling is mostly focused on soft errors, because, as discussed above, SER increases very sharply as the supply voltage is scaled downward, which constrains the useful range of DVFS. For operation at nominal or sub-nominal voltage-frequency regions, hard-fault incidence rates are well below design specification limits, so these effects need not be considered explicitly in resilience models within PEARL. Even when turbo-mode operation is required (very rarely) to meet deadline constraints in field operation, the hard-failure rate is actually within specification, since the processor is designed to tolerate the higher voltages and temperatures incurred in turbo-mode. Nonetheless, for completeness, we decided to consider power consumption as a proxy or gauge for hard-failure rates of the system. With increased voltage-frequency operation points, the current drawn is higher, so failures linked to current density (e.g. electro-migration) are higher. Also, higher voltages translate to failure rates associated with mechanisms such as oxide breakdown and bias-temperature instability (BTI).

5.2.5 Workflow Synthesis

PEARL is a framework for workflow optimization. However, the target benchmark suites are composed of individual applications. Therefore, a workflow synthesizer, which has been embedded in the PEARL framework, is required to assemble workflows from applications according to user-specified parameters, such as application dependencies, execution time (duration), etc. The synthesizer is able to generate various workflows for both design space exploration and algorithm evaluation. In order to characterize the potential benefits of full-workflow scheduling, this chapter synthesizes workflows using normalized applications. PEARL framework is capable of handling arbitrary workflows composed of applications with any variety of execution time.

5.3 Linear Workflow Optimization

Our primary goal is to maximize system energy-efficiency measured by GOPs/Watt, while still meet the deadline constraint for the workflow. For the requirement of resilient embedded processing, the maximization is also subject to soft-error induced resilience constraint as well as hard-error induced power constraint. In other words, the average power consumption of any applets can not exceeds certain value dictated by lifetime reliability requirement. We use the maximum power consumed by any applets within a workflow working at nominal 4.1GHz as the power maximum. While for soft-error induced resilience requirement, we define the workflow resilience metric as the time-weighted FIT rates of all applets within a workflow as:

$$\frac{\sum FIT_i * T_i}{\sum T_i} \leq C_r$$

where FIT_i and T_i is the FIT estimator and run time of the ith applet.

Before proceeding to the algorithm, we first formally define the optimization problem by given the follows:

- A linear workflow composed of N applications, $W = \{A_1, A_2, \ldots, A_N\}$;

- A list of M possible processor frequency levels (or modes), $F = \{f_1, f_2, \ldots, f_M\}$, each associated with M corresponding voltage levels.

- The power, execution time and resilience characteristics of each application under all available frequency levels, $P[i] = \{p_{i1}, p_{i2}, \ldots, p_{iM}\}$, $T[i] = \{t_{i1}, t_{i2}, \ldots, t_{iM}\}$, $R[i] = \{r_{i1}, r_{i2}, \ldots, r_{iM}\}$, $i = 1, 2, \ldots, N$;

- Constraints on maximum power (C_p), maximum execution time, or deadline (C_t), and maximum resilience metric(C_r).

Minimize: Total energy of execution by picking the “best” combination of frequency levels $(f_1, f_2, \ldots, f_N)$ across the N applets while adhering to workflow deadline, power, and resilience metric constraints.

5.3.1 LinOpt

We now describe a heuristic called LinOpt that is an available option within PEARL. LinOpt is formulated as a static real-time-constrained DVFS mode allocation problem, and solved by an linear programming algorithm. To formulate it as an integer linear programming problem, for each application A_i, we define a frequency mask bit-vector (FM) of length M, FM_i is a member of the set $[0,1]$, where only bit j within FM_i is set to 1, indicating a particular choice of DVFS mode, as illustrated in Figure 5.4.

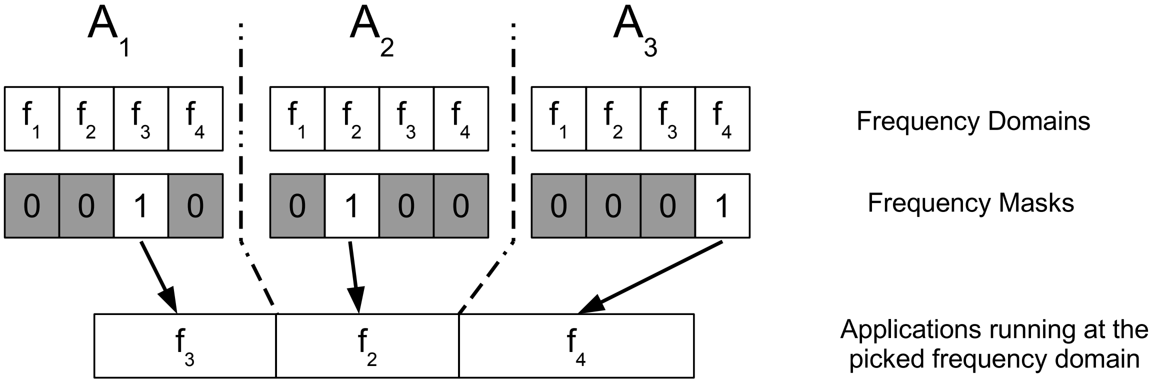


Figure 5.4: Using frequency masks to denote the running frequency domains of each application.

A_i's power, performance and resilience can be expressed as the dot product of the characteristics and the associated frequency mask FM_i:

- Power = $P[i] \cdot FM_i$
- (Inverse) Resilience = $R[i] \cdot FM_i$
- Execution time = $T[i] \cdot FM_i$

Constraints can be expressed as:

- Power: $P[i] \cdot FM_i \leq C_p$ for each application A_i
- Resilience metric: $(\sum R[i] \cdot FM_i)/C_t \leq C_r$ for each application A_i
- Execution time: $\sum T[i] \cdot FM_i \leq C_t$

The goal of the original problem is equivalent to minimizing the total energy of execution by picking the “best” combination of FM bit-vectors ($FM_1, FM_2, \ldots, FM_N$) across the N applications while adhering to workflow execution time, power and resilience constraints. It is an integer linear programming problem. We use a standard branch-and-cut algorithm [52] to solve the problem.

5.3.2 Energy Efficiency Improvement

We evaluate the proposed algorithm against the baseline using various constraints. The baseline picks a single frequency level that is the lowest that meets all the constraints across all the applications. For a given workflow, the maximum and minimum execution time is determined by running all applets at the lowest and highest frequency levels, respectively. We evenly sample 32 data points within the range of possible workflow execution time. For resilience metric constraint, we similarly sample 32 data points within the range of possible resilience metrics, where the max metric is adopted by running all applets at the lowest frequency level while the minimum metric is adopted by running all applets at the highest frequency level. We first show the results when applying LinOpt on the workflow that is composed of all 20 applets we have characterized. For each set of constraints including deadline, power, and resilience, we compare the resulting energy efficiency derived from running applets at frequency levels chosen by LinOpt and the baseline. Finally, we plot the energy efficiency improvement of LinOpt over the baseline for the whole space of sweeping constraints in Figure 5.5.

The energy efficiency improvement of Linopt is plotted as a heat-map with the darker colors indicate higher improvements, while the lighter colors indicating lower improvements. LinOpt achieves the highest improvement of 15% when the resilience metric constraint is 50.8 and the deadline constraint lays in the upper region of the range, shown as the darkest red stripe in Figure 5.5. The top left portion of the heat-map is mostly red, and lower right portion is mostly blue. This suggests that energy efficiency improvement over the baseline decreases when the resilience metric constraint is less stringent. This is because when the resilience metric constraint is less tight, a single frequency chosen by SimpleOpt is more likely to be close to optimal, whereas when the constraints are tighter, LinOpt is required.

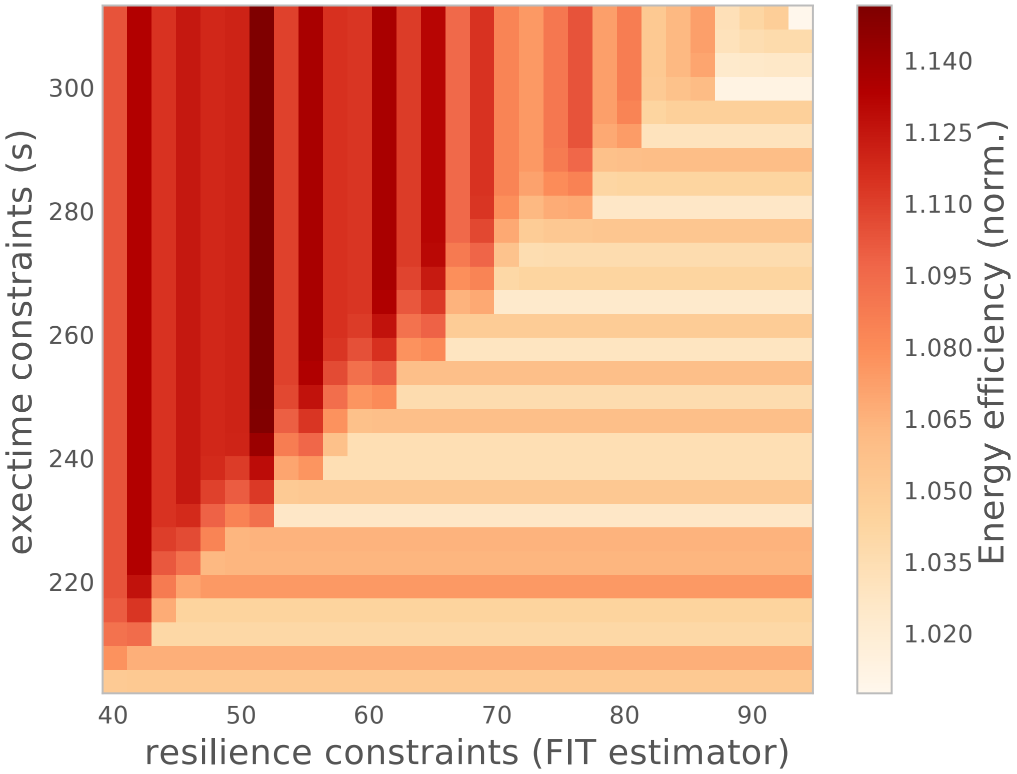


Figure 5.5: LinOpt improvement in performance/Watt (GOPS/W) over SimpleOpt across a range of constraints. The workflow is a stitched sequence of all 20 characterized applets. Constraints of resilience and execution time are plotted on X-axis and Y-axis respectively. Energy efficiency obtained by LinOpt under various constraints are normalized to the baseline SimpleOpt under the same constraint, and plotted in a heat-map with dark color indicating the highest, while light color indicating the lowest.

Then, we use the embedded workflow synthesizer to generate workflows by selecting 10, 15, or 20 applications from the full suite of 20. The selection is done by using random sampling with replacement. For each generated workflow, we set the power and resilience constraints to be the maximum values of these metrics as observed across all the applications within the workflow at nominal frequency level. After that, we exhaustively search the space of feasible deadline constraints. We then report the best energy efficiency improvement over the baseline to demonstrate the maximum potential of the algorithm. As shown in Figure 5.6, LinOpt outperforms the baseline by around 15% for the majority of the 20-application workflows. In some less common cases, the improvement may go up to 25%. Comparing the three plots, the mass of the histogram is seen to

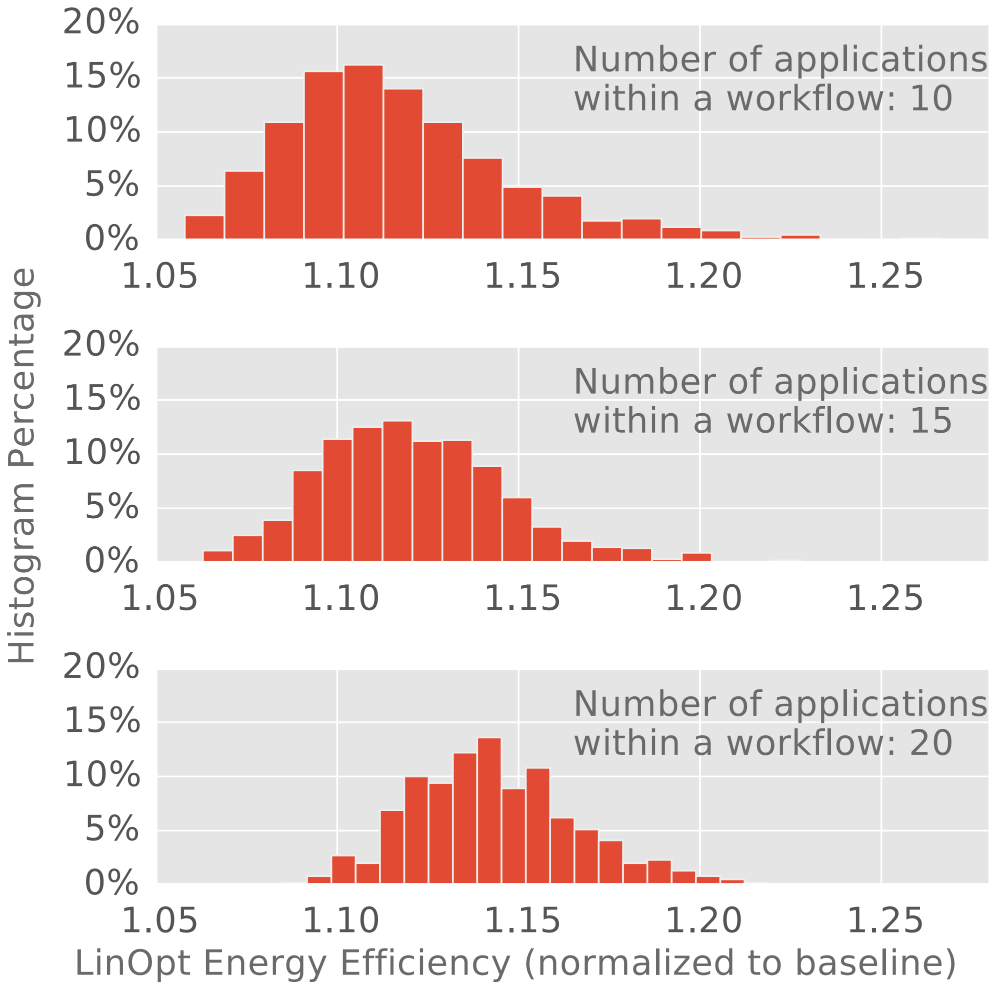


Figure 5.6: Histogram of the maximum improvement in BIPS/W achieved by LinOpt over the baseline across 1,000 generated workflows with a given number of applications. Power and resilience constraints are set as the maximum of applications within a workflow running at the nominal frequency level. X-axis is the relative energy efficiency of LinOpt normalized to the baseline. Y-axis is histogram in percentage.

shift to the right, suggesting longer workflows with more diverse behaviors will generally lead to better energy efficiency improvements from LinOpt.

5.3.3 Impact of Power Variation

We notice that power characterization under various voltage/frequency settings are quite similar for most of applications we include in this study (Figure 5.3a). Therefore, the energy saving from

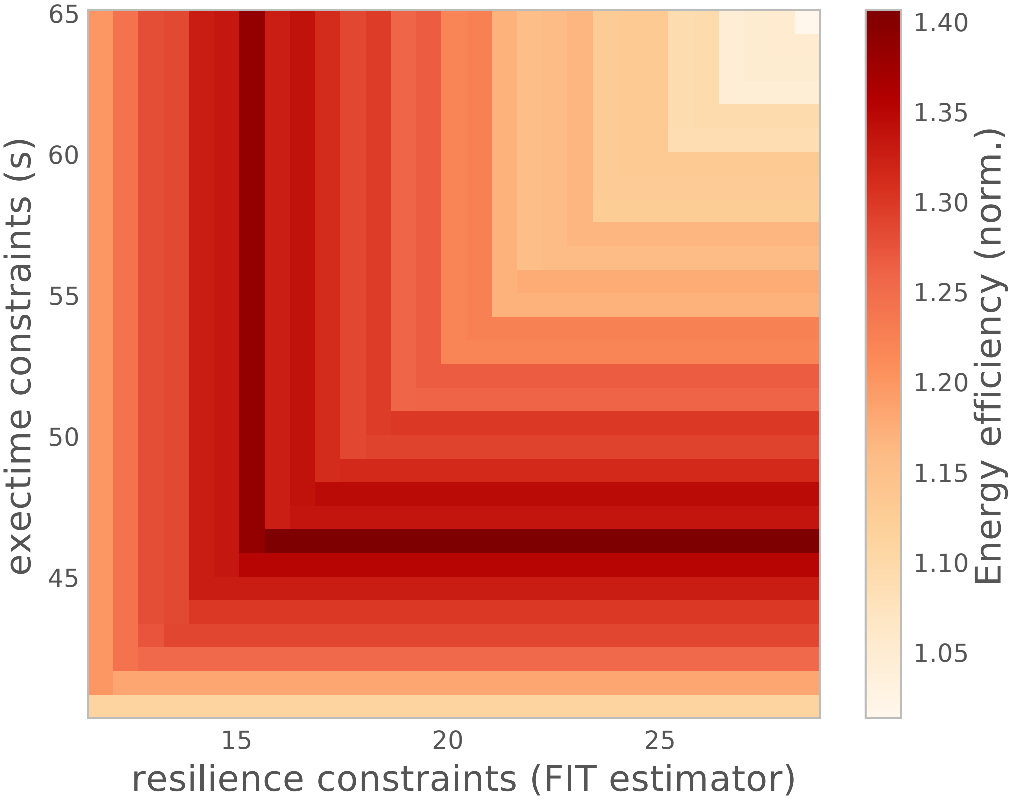


Figure 5.7: *LinOpt* benefit over *SimpleOpt* for a workflow composed of two synthetic applets based on *2dconv*.The *syn-hp* doubles the power characterization of *2dconv* to serve as a high-power variant, and the *syn-lp* reduce the power characterization by 90% to serve as a low-power variant.

applets operating at lower voltage/frequency settings is offset considerably by applets operating at higher voltage/frequency settings.

To validate our argument, we create a workflow composed of two synthetic application segments with high power (*syn-hp*) and lower power (*syn-lp* consumption). The two synthetic applications are derived from characterizations of *2dconv*, where *syn-hp* doubles the power characterization of *2dconv* at each frequency level to mimic a hypothetical power-hungry application and *syn-lp* adopts 1/10 of the power characterization of *2dconv* to mimic a hypothetical low-power application. Noted that these two synthetic applications are for illustration purpose, the ratios are picked in order to create a large power variations to explore potential of LinOpt under the extreme case. The generated workflow has one segment of *syn-hp* followed by three segments of *syn-lp*. As shown in Figure 5.7, LinOpt exploits the power variations between the two synthetic applications and delivers up to 40% of energy efficiency improvement over SimpleOpt.

5.3.4 Impact of Execution Time Scaling

Another observation is that most of the characterized applications are computation-intensive, of which the execution time scales linearly to the operating frequency. As a result, the energy saving from slowing an applet is largely offset by another applet which has to be boosted in order to meet the deadline. This ends up with a minimal overall energy efficiency improvement.

To justify the above argument, we create a workflow composed of both a computation-intensive applet (*2dconv*) and a memory-intensive applet (*lbm*). With this workflow, LinOpt achieves much higher energy efficiency improvement over baseline. In Figure 5.8, we use the workflow has 7 ap-

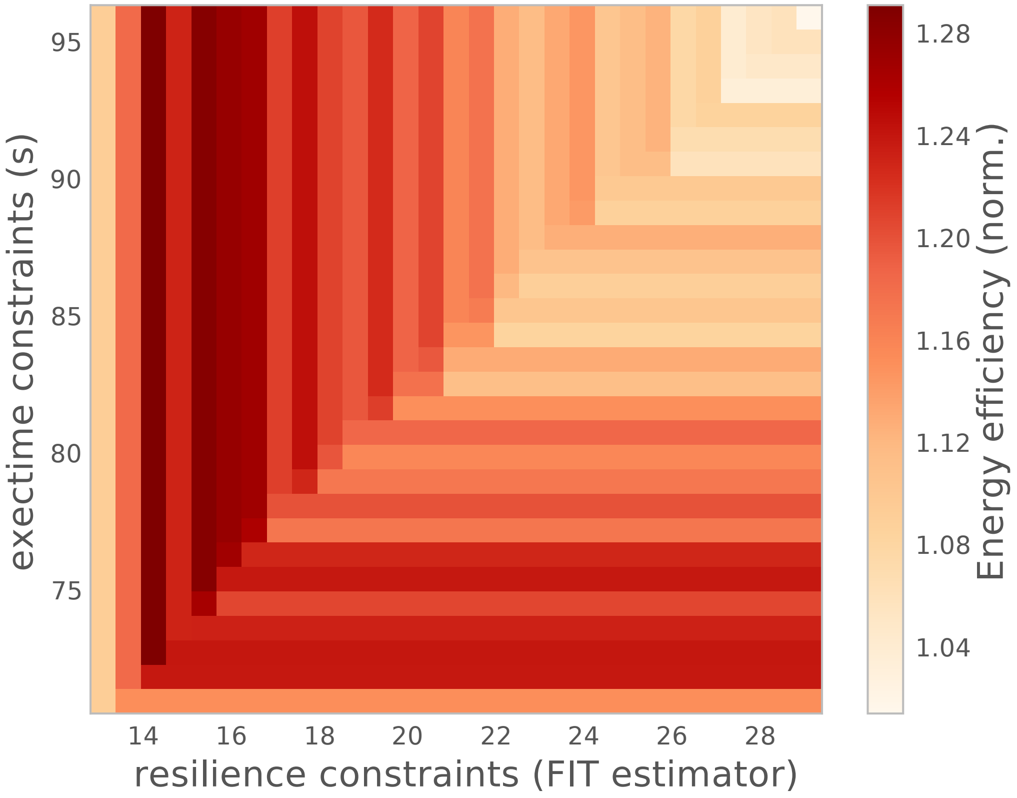


Figure 5.8: LinOpt improvement in performance/Watt (GOPS/W) over SimpleOpt when the targeting workflow compose of *2dconv* and *lbm*, of which have quite different execution time characterization with regard to voltage-frequency scaling. The workflow includes 7 applets as 3 *2dconv* followed by 4 *lbm*.

plication segments with 3 copies of *2dconv*, and 4 copies of *lbm*. LinOpt achieves up to 29% energy efficiency improvement over the baseline. This is because with $\alpha = 0.31$, *lbm* shows very limited sensitivity in run time with regard to voltage-frequency scaling. As a result, the operating frequency for *lbm* can usually be lowered significantly without violating the overall deadline constraint, and eventually leads to higher energy efficiency by operating at lower voltage-frequency levels.

5.4 Graph Workflow Optimization

5.4.1 DAGopt

We propose an iterative optimization algorithm called DAGopt to statically optimize voltage-frequency settings of applications within a graph workflow. The basic idea of DAGopt is to apply LinOpt iteratively on paths of a DAG until all applications within a graph workflow are optimized. For a given graph workflow, as shown in Algorithm 1, DAGopt initializes all applications with the lowest frequency level, and identifies the critical path as C: one that has the longest execution time. Then it applies LinOpt to C to choose the optimal frequency levels for each application within C. After a successful optimization of C, DAGopt updates the DAG with the execution time of all applications within C running at the optimized frequency level, and marks these applications as processed. In the next iteration, DAGopt identifies the *next_critical_path* as C^*, which is defined as the longest path that traverses at least one un-processed application. Then it applies LinOpt on C^* with the same deadline constraint of T, where frequency levels can only increase on any already-processed applications. As a result, any paths that share applications with C^* can only run faster after the current iteration. This restriction ensures that no paths processed in prior iterations would violate deadline constraints, while sacrificing energy efficiency due to the potential increase of frequency levels on already-processed applications. DAGopt keeps working on the *next_critical_path* until all applications have been marked as processed (therefore, no new *next_critical_path* can be found). The algorithm processes at least one application in each iteration so that it finishes in at most N-iterations, where N is the number of applications within a workflow. Figure 5.9 shows how the algorithm executes on an illustrative DAG.

Algorithm 1 Graph workflow optimization algorithm

```
function DAGopt(workflow, constraints)
    initialize $freq\_list$ by the lowest frequency level ($freq_{min}$)
    for all $app$ in $workflow$ do
        update the execution time of $app$ set by $freq_{min}$
    while not all applications processed do
        $C \leftarrow$ NextCriticalPath(workflow)
        $freq\_opt \leftarrow$ LinOpt($C$, constraints, $freq\_list$)
        if LinOpt fails on path $C$ then
            return Failure due to in-feasible constraints
        update $freq\_list$ with $freq\_opt$
        for all $app$ in $C$ do
            update the execution time of $app$ according to $freq\_opt$
    return $freq\_list$
function NextCriticalPath(workflow)
    for all $app$ that is not processed do
        $C_{to} \leftarrow$ longest path from source to $app$
        $C_{from} \leftarrow$ longest path from $app$ to target
        $C \leftarrow C_{to} + C_{from}$
    return max $C$
```

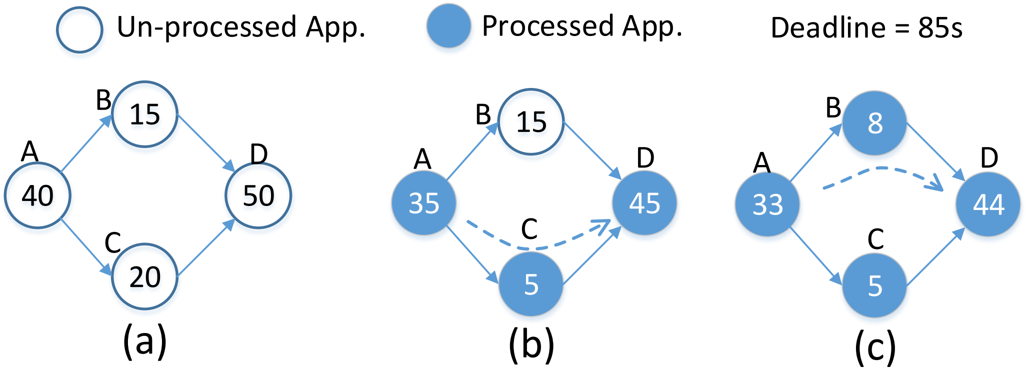


Figure 5.9: DAGopt illustration on a synthetic workflow. Numbers in each node indicate the execution time of the corresponding application in seconds. Suppose that the deadline constraint is 85s. In (a), all applications are initialized with the lowest frequency levels. In (b), DAGopt identifies the critical path as $A \rightarrow C \rightarrow D$, then invokes LinOpt on the path. Upon a successful optimization, DAGopt sets optimized frequency levels as indicated by the new execution time, and marks all nodes within the path as *processed*. In (c), DAGopt identifies the *next_critical_path* as $A \rightarrow B \rightarrow D$ and invokes LinOpt on the path while only selecting higher frequency levels for A and D than their already chosen levels. DAGopt terminates after this step since all the applications within the workflow have been marked as processed.

5.4.2 DAGopt Evaluation

We first show a workflow illustrated in Figure 5.10, for which the DAGopt has a promising energy efficiency boost over the baseline. The workflow has two threads of linear workflows. Each linear workflow is composed of applications chosen from *2dconv*, *lbm*, and *bzip2*. We add inter-dependencies between the two threads, shown by the dashed arrows. Power and resilience constraints are set as the maximum power and maximum FIT estimates of all applications within a workflow running at nominal frequency. For demonstration purposes, we select 10 evenly-spaced sample points for deadline constraints from the range of critical path execution times, determined by running all applications within a workflow at the lowest and highest frequency levels. Results shown in Figure 5.11 suggest up to 35% energy efficiency improvement is achievable over the simple single-frequency selection heuristic. This is because, with the added inter-dependencies, application segments such as *bzip2* in the first thread can be slowed down considerably thanks to the dependence-imposed timing slack.

Finally, we carry out the energy efficiency distribution study that is similar to what we have shown in the previous section. We first randomly choose 14 applications to form two linear workflows, with seven applications per workflow. Then we randomly add a given number of inter-dependencies between the two linear workflows to finish a graph workflow. Using this approach, we generate 1,000 graph workflows, and compare DAGopt against the baseline which picks a single frequency for applications within the workflow. We use the same constraints that were just described. We choose the number of inter-dependencies to be 5, 10, and 15, and plot the histogram of maximum relative energy efficiency obtained by DAGopt over the baseline in Figure 5.12. It shows that with fewer inter-dependencies (e.g. 5), DAGopt is able to achieve an energy efficiency boost over the baseline by 10% to 20%. As the number of inter-dependencies increases to 15, the improvement in energy efficiency reduces to the range of 5% to 10%. This is because DAGopt

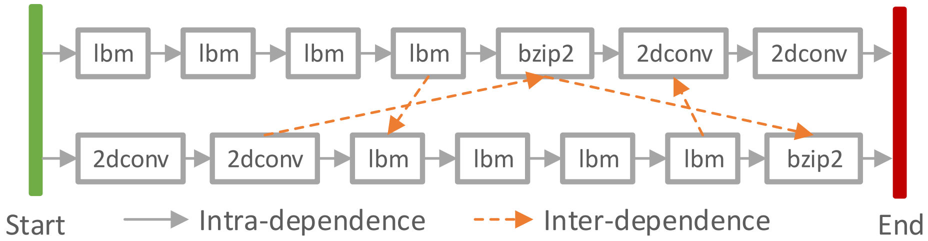


Figure 5.10: An illustrative graph workflow, consisting of 2 threads with inter-dependencies indicated by dashed arrows.

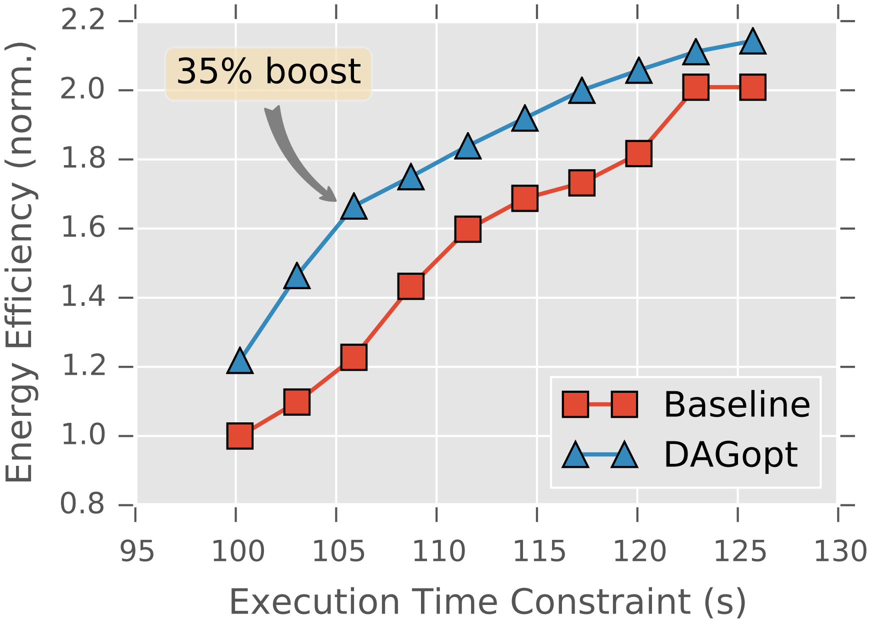


Figure 5.11: DAGopt energy efficiency improvement over simple single-frequency selection heuristics, evaluated on an illustrative workflow (Figure 5.10). Power and resilience constraints are set as the maximum of applications within a workflow running at the nominal frequency level. Energy efficiency values, plotted on Y-axis, are normalized to the baseline under the execution time constraint of 100s (the leftmost data point). DAGopt outperforms the baseline in all cases, with up to 35% boost when the execution time constraint is 106s.

becomes more limited in trade-off options among applications as the number of inter-dependencies increases.

5.5 Dynamic, Run-time Efficiency Optimization

5.5.1 Dynamic Iterative Approach

PEARL is also capable of emulating real systems by considering dynamic execution time variations. For each application within a workflow, we define its run-time variation factor (rvf) for a specific execution instance as the ratio of actual execution time to the pre-characterized value at the same frequency level. An rvf value greater than one indicates that the actual execution time of the corresponding application exceeds the expected pre-characterization value. In this case, there is a

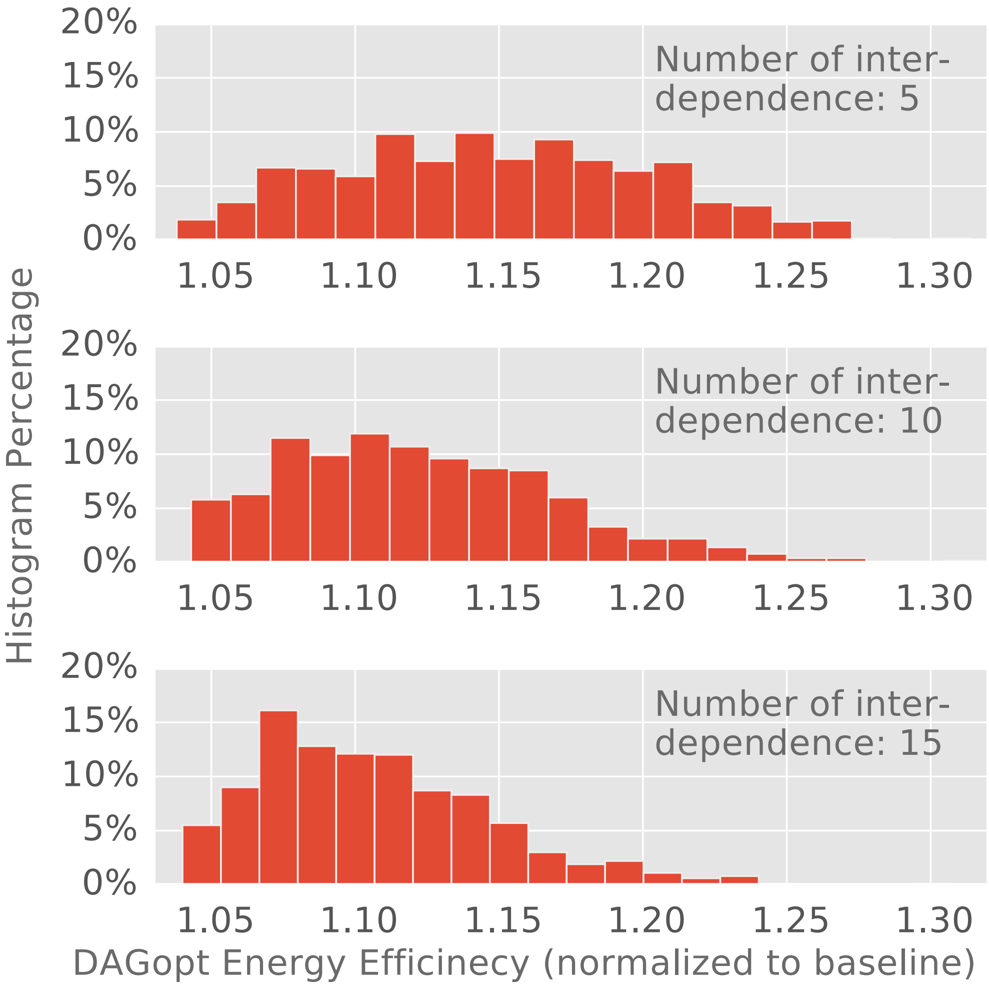


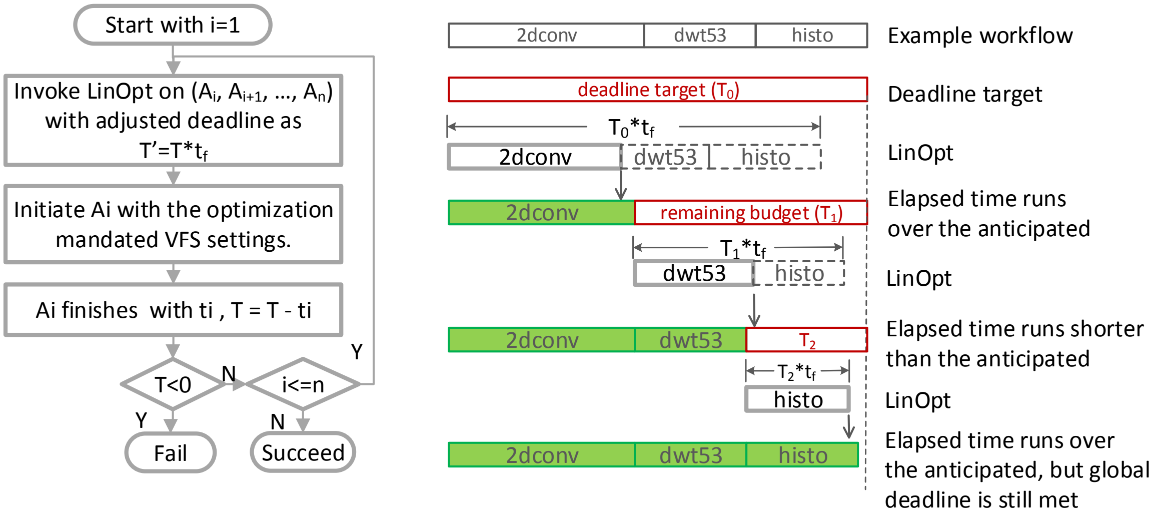
Figure 5.12: Histogram of the maximum relative BIPS/W obtained by DAGopt over the baseline across 1,000 generated workflows. The number of inter-dependencies is varied from 5 to 15. Power and resilience constraints are set as the maximum of these metric values across applications within a workflow running at the nominal frequency level. The X-axis is the relative energy efficiency normalized to the baseline, Y-axis is the histogram distribution percentage. It shows that with fewer inter-dependencies, DAGopt is good to achieve an energy efficiency boost over the baseline by 10% to 20%, while as the number of inter-dependencies increases to 15, the improvement in energy efficiency reduces to the range of 5% to 10%.

potential to miss the overall deadline for a workflow. On the other hand, an rvf value less than one indicates that the actual execution time is less than the pre-characterized value. The resulting slack could be potentially reclaimed to improve overall energy efficiency.

The static optimization algorithm on a linear workflow tends to choose lower frequency settings to maximize energy efficiency. As a result, the total execution time of a given solution is usually close or equal to the deadline constraint. Therefore, such a static algorithm's success becomes vulnerable to dynamic execution time variations, where the rvf value of at least one application in the workflow is greater than 1. With rvf following the Gaussian distribution of $\mathcal{N}(1,0.1)$, frequency settings given by LinOpt miss deadline constraints in more than 50% of trials. Although it is possible to invoke LinOpt with a stricter deadline constraint than the original to tolerate execution time variations, it reduces the achieved energy efficiency significantly. For example, with the same rvf distribution, it reduces the average energy efficiency by 20% when statically invoking LinOpt with only 90% of the original deadline constraint.

In this section, we present an iterative dynamic algorithm (Figure 5.13a) to achieve a high level of confidence in meeting deadline constraints without significantly compromising the energy efficiency. The algorithm works as follows:

1. Before starting application A_i, we invoke LinOpt with an adjusted deadline by applying a scaling factor (t_f) to the original deadline constraint. Then, we launch A_i with the frequency


Start with i=1
Invoke LinOpt on (A_i, A_i+1, …, A_n) with adjusted deadline as T'=T*t_f
Initiate Ai with the optimization mandated VFS settings.
Ai finishes with ti , T = T - ti
T<0
N
i<=n
Y
Y
N
Fail
Succeed
2dconv
dwt53
histo
Example workflow
deadline target (T0)
Deadline target
T0*tf
LinOpt
remaining budget (T1)
Elapsed time runs over the anticipated
T1*tf
Elapsed time runs shorter than the anticipated
T2
T2*tf
Elapsed time runs over the anticipated, but global deadline is still met

a) Algorithm flowchart

b) Illustrative example

Figure 5.13: An iterative algorithm adapting to dynamic execution time variations. In (a), the algorithm is presented as a flowchart, and in (b), a step-by-step example illustrates the dynamic optimization algorithm. The illustrative linear workflow is a stitched sequence of three applications: *2dconv*, *dwt53*, and *histo*.

level picked by LinOpt.

2. At the end of A_i, we observe that the actual elapsed execution time for A_i is t_i. We incorporate this slack (positive or negative) and update the deadline constraint for the residual workflow by deducting t_i from the residual deadline. If it is less than zero, the algorithm terminates and reports "in-feasible" as the deadline constraint is not met.

3. We go back to step 1) and iterate until the workflow is fully processed.

The algorithm is visualized step by step in Figure 5.13(b) using an illustrative workflow composed of *2dconv*, *dwt53*, and *histo*.

It is important to choose appropriate values for the deadline scaling factor t_f. When t_f equals 1, it invokes LinOpt with the original deadline constraint. In this case, the algorithm achieves maximized energy efficiency, while it is vulnerable to any applications that run slower than their pre-characterized latency. On the other hand, when t_f is less than 1, the algorithm invokes LinOpt with a more stringent deadline constraint. In this case, the algorithm creates slack that tolerates runtime delays while sacrificing energy efficiency. Note that the value of t_f can be larger than 1 in speculating that most applications finish earlier than their pre-characterized latency. However, we will not discuss this scenario in this chapter.

We choose t_f based on the observation that delays in early phases of a workflow are more easily compensated for, while delays in later phases of a workflow are more dangerous because there is less opportunity to compensate with few remaining residual applications. Based on this observation, our algorithm exploits the most aggressive $\max(t_f)$ at the beginning for the first application to maximize potential energy efficiency gains, then gradually lowers t_f to the most conservative $\min(t_f)$ at the end for the final applications to stay within the deadline constraint. The pair of parameters ($\max(t_f)$,$\min(t_f)$) enables the trade-off between energy efficiency and the level of confidence in meeting the deadline constraints. We choose a value of 1 for $\max(t_f)$, speculating that few applications finish earlier. We evaluate alternate $\min(t_f)$ values in the next subsection with regard to distributions of rvfs.

5.5.2 Evaluation

To study the impact of run-time variations, we use an illustrative workflow by stitching together four copies of the *lbm* application, two copies of *2dconv*, and one copy of *bzip2*, and replicate the 7-application sequence three times. The resulting workflow has 21 applications. For illustration purposes, we use Monte-Carlo simulation method to generate rvf sequences by following a Gaussian distribution for the workflow. For each rvf distribution, we sample 1,000 rvf sequence samples. We study three alternative distributions to cover various dynamic execution scenarios: a)*VarL* with $N(1,0.1)$, which reflects the scenario that the system experiences significant, intermittent changes in operating environment; b)*VarS* with $N(1,0.03)$, which reflects a scenario similar to *VarL* except that only moderate changes in operating environment are observed; c)*Delay* with $N(1.05,0.02)$, which reflects the scenario in which steady deviations from the nominal operating environment lead to persistent delays for all applications (as indicated by the mean rvf that is larger than 1). We study the dynamic iterative approach with three values for parameter $\min(t_f)$: 0.9, 0.85, and 0.8. The three cases are denoted as DYN_0.9, DYN_0.85, and DYN_0.8, respectively. We compare the three dynamic approaches against the static LinOpt (denoted as ST_1). We include an alternate static case which targets 90% of the original deadline constraint (denoted as ST_0.9). Note that rvf distributions are chosen to illustrate the capabilities of PEARL. PEARL accepts distributions chosen by users.

As shown in Figure 5.14, static approaches are either too conservative, yielding lower energy efficiency (in the case of ST_0.9), or too vulnerable to dynamic execution time variations, yielding higher deadline miss ratio (in the case of ST_1). Dynamic approaches, on the other hand, achieve energy efficiency that is close to static LinOpt, which targets the original deadline constraint. Within dynamic approaches, the energy efficiency increases in a limited manner as $\min(t_f)$ increases. In the cases of *VarS* and *Delay* where distribution variations of rvf are small, dynamic approaches do not miss any deadlines, in any experimental trials, which is as good as the conservative static approach (ST_0.9). In the case of *VarL*, dynamic approaches with higher $\min(t_f)$ (e.g. DYN_0.9) suffer from a higher deadline miss ratio. The latter is more than double the miss ratio observed in dynamic approaches with lower $\min(t_f)$ (e.g. DYN_0.8), despite a modest energy efficiency improvement. Overall, the dynamic approach prefers a high $\min(t_f)$ when the variation of rvf is small, and a low $\min(t_f)$ is better when the variation of rvf is large.

5.6 Conclusions

Future ultra-efficient embedded systems with mission-critical resilience requirements in the deep-submicron design era will require a careful balance between static preparation and run-time adaptation of applications. In this chapter, we first describe a software framework called PEARL for application preparation and runtime steering in the context of diverse performance, power, and resilience constraints. Then we present experimental analysis for a diverse set of application sequences to demonstrate the functionality and capabilities of the PEARL framework. Our analysis shows up to 15% and 35% energy efficiency improvement over a simple baseline, for linear and graph workflows respectively. Moreover, our proposed dynamic iterative approach achieves up to

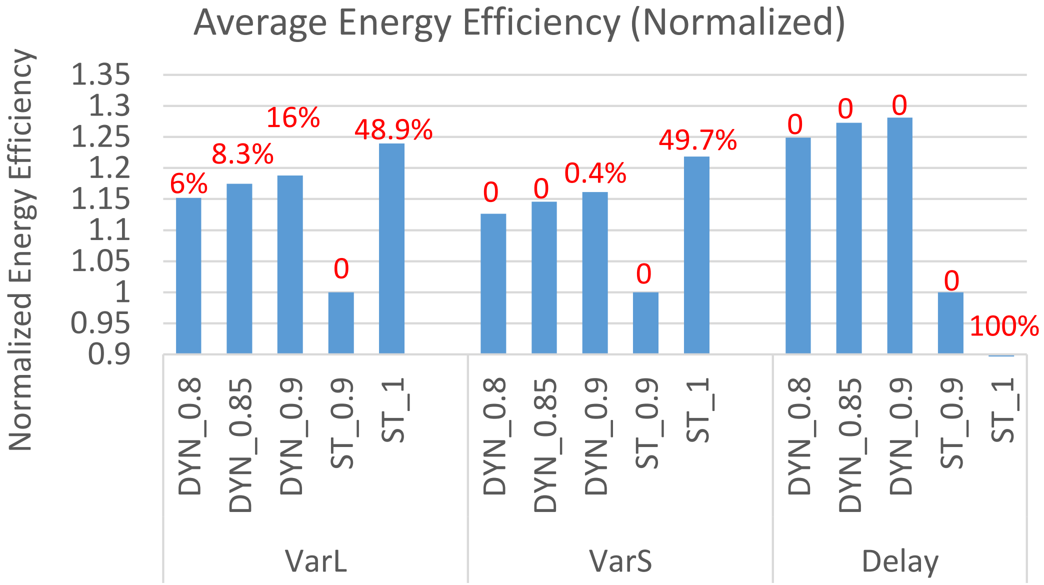


Figure 5.14: Dynamic simulation results. For the described scenarios (*VarL*, *VarS*, and *Delay*), we compare the dynamic iterative approach with 3 values of $\min(t_f)$ (DYN_0.9, DYN_0.85, and DYN_0.8), and two two cases of static LinOpt with (ST_0.9) and without (ST_1) conservative provision on deadline constraints. The Y-axis is the average energy efficiency computed in BIPS/W normalized to ST_0.9. The percentage on top of a bar is its deadline miss ratio. Dynamic approaches generally achieve energy efficiency close to the aggressive static approach (ST_1), while maintaining miss ratios close to the conservative static approach (ST_0.9).

25% improvement in energy efficiency over a conservative static approach, while maintaining the same level of confidence in meeting deadline constraints.

Chapter 6

Conclusions and Future Work

6.1 Dissertation Summary

Continuous scaling in manufacturing process has introduced new challenges to improve performance without violating design constraints of power and resilience. In this dissertation, we have proposed first-order modeling frameworks for power, performance, and resilience, to help system architects to understand various implications for future computer systems.

In Chapter 2, we describe *Lumos*, an analytical modeling framework for power and performance of both homogeneous and heterogeneous architectures. By exploring the performance potential of homogeneous systems operating at lower-than-nominal supply voltage (e.g. with dim cores), we show that up to 2x speedup can be expected over cores running at the nominal supply. However, the diminishing performance returns from dim cores open a window for more efficient specialized hardware. Further analysis indicates that efficient accelerators that can be re-purposed after fabrication are the key for future heterogeneous architectures.

In Chapter 3, we continue our exploration from Chapter 2, by using an augmented analytical framework called Lumos+ to explore the design space of heterogeneous architectures composed of hardware accelerators. With the help of a genetic algorithm based search heuristic, we show that dedicated accelerators are only beneficial when their performance premiums are large enough (e.g. 40x) to compensate for limited programmability, or when only a few kernels need acceleration. However, systems equipped with reconfigurable logic are promising to achieve close-to-optimal performance consistently across applications with a wide range of kernel characteristics, an important implication for rapidly evolving, general-purpose workloads.

In Chapter 4, we describe a framework called AFI for application programmer and system designer to take advantage of application-level resilience. Using AFI, we have characterized 12 applications from PERFECT Suite, a benchmark suite for emerging embedded systems. The results show a wide range of application-level resilience (SDC vulnerability rates from about 2% though more than 50%), which motivates further dynamic techniques to trade-off among resilience, power and performance. We also find that application-level resilience is very sensitive to the compiler optimization, where the SDC vulnerability rate is around 1% without optimization and increases to more than 35% with the "O2" optimization level.

Finally, in Chapter 5, we designed a framework, called PEARL, for resilience-aware energy-efficiency optimization with the deadline constraint for future ultra-efficient embedded systems with mission-critical resilience requirements. PEARL is capable of application preparation and runtime steering in the context of diverse performance, power, and resilience constraints. Then we present experimental analysis for a set of various application sequences to demonstrate the functionality and capabilities of the PEARL framework. Our analysis shows up to 15% and 35% energy efficiency improvement over a simple baseline, for linear and graph workflows respectively. Moreover, our proposed dynamic iterative approach achieves up to 25% improvement in energy efficiency over a conservative static approach, while maintaining the same level of confidence in meeting deadline constraints.

6.2 Future Directions

6.2.1 Design Space Exploration

As shown in Chapter 2 and 3, reconfigurable accelerators are more promising due to its capability to adapt to various kernels. However, overhead, if accumulated by excessive reconfiguration operations, could eliminate all their gains in performance and power efficiency. This actually inspires an alternative reconfigurable accelerator design: instead of programming and serving a single kernel at a time, designers could implement and instantiate multiple accelerators simultaneously and serve more than one kernels without repeatedly reprogramming the logic. Such a design paradigm raises interesting research questions such as:

- Design trade-off between implementation of multiple kernels and a single kernel. The multiple-kernel implementation benefits from fewer reconfiguration and smaller overhead thanks to the partial reconfiguration techniques. However, it suffers from worse per-kernel performance compared to a single-kernel implementation as each kernel is only allocated by a portion of all the logic resources.

- Resource allocation optimization among kernels in a multiple-kernel implementation. In the case of practical applications, the performance of reconfigurable accelerator scales to the amount of logic resources by a discrete step-wise function, instead of a continuous linear function. And scaling functions vary across accelerators. Allocation algorithms for logic resources are needed to optimize the performance of each instantiated accelerator.

- Real-world applications have far more kernels than what can be instantiated in the reconfigurable fabric at a single time. Therefore, dynamic scheduling algorithms are required to implement accelerator replacement when a kernel needs to be accelerated buy yet implemented in the logic. A good scheduling algorithm is capable of minimize replacement to reduce the reconfiguration overhead.

6.2.2 Application-Level Resilience

The results presented in Chapter 4 target for processors from POWER family. It would be interesting to compare results from other architectures by porting the AFI framework to platforms such as X86, ARM, etc. The comparison helps to understand the resilience impact of architectural variations across platforms. Another promising research direction on application-level resilience is to characterize resilience for multi-threaded applications. Most of the prior resilience characterization work, including the one presented in this dissertation, focused on single-threaded applications. However, multi-threaded applications become more and more popular in most computing domains.

The resilience characterization on a threaded application may be quite different from its serial implementation, due to complexities in synchronization and inter-thread communications.

6.2.3 Resilience-Aware Energy-Efficiency Optimization

In Chapter 5, we have shown energy-efficiency optimization for single-entity linear workflows and multi-entity graph workflows. In either cases, applications are executed in serial on a single entity, which is a reasonable assumption for most of the embedded systems today. However, the hardware in the embedded system is evolving rapidly to be capable of handling multiple tasks. Recent development of autonomous techniques for flying and grounded vehicles corroborate the need of multi-tasking in embedded systems. Multi-tasking introduces new challenges for resilience-aware energy-efficiency optimization due to the task scheduling. The optimization algorithm depends on the output of scheduling to determine the critical path and overall power consumption of all tasks, while the scheduling algorithm is sensitive to the finish time of each task which depends on the voltage levels selected by the energy-efficiency optimization. The two algorithms are required to collaborate with each other to form a feedback loop, which is a new challenge for the design of both algorithms.

Additionally, the adaptation approach proposed in Chapter 5 needs re-optimization before launching every single task. The total number of tasks should be maintained at a low level to avoid exaggerate overhead of re-optimization, which limits the granularity of sub-task partition for long-running tasks. On the other hand, the proposed approach requires global knowledge of all tasks to be executed. This is only acceptable for routine tasks that can be pre-characterized. To support fine-grained truly dynamic adaptation, two challenges need to be addressed. First, light-weight online estimates of power, performance and reliability are required. It is relatively easier to find estimates of power and performance via monitoring performance counters which are widely available on modern processors. On the other hand, the online estimate for reliability is more complicated to adopt. SoftArch [49,50] presented a promising SER estimate based on statistical sampling. Second, a runtime adaptation algorithm is needed to speculatively apply DVFS on each application segment, in order to save energy while still meet power, performance (deadline), and reliability constraints.

Appendix A

Glossary

Definitions are collected here for easy reference. In general, the accepted definitions for terms are used, although some terms are used in a more restricted sense than their usual interpretation.

SER Soft error rate.

SEU Single-event upset.

DMR Dual-Modular Redundancy.

TMR Triple-Modular Redundancy.

SDC Silent data corruption.

MPSoC Multiprocessor system-on-chip.

FPGA Field Programming Gate Array, a device that can be reprogrammed to change its functionality after fabrication.

Reconfiguration An operation to change the function of reconfigurable fabric.

RL Reconfigurable Logic, a device whose functionality can be changed after fabrication. FPGA is one of the most popular RL devices.

ASIC Application Specific Integrated Circuit, a device that has a fixed functionality (application-specific), and can not be changed after fabrication.

Workflow A collection of applications with dependencies, that accomplishes certain tasks.

BIPS/W Billion Instructions Per Second Per Watt, a metric to evaluate energy efficiency of a computer system, the higher the better efficiency.

Bibliography

[1] Gene M Amdahl. Validity of the Single Processor Approach to Achieving Large Scale Computing Capabilities. In *Proceedings of the Spring Joint Computer Conference*, 1967.

[2] Todd Austin, Eric Larson, and Dan Ernst. SimpleScalar: An Infrastructure for Computer System Modeling. *Computer*, 35(2):59–67, February 2002.

[3] Simon Baker and Iain Matthews. Lucas-Kanade 20 Years On: A Unifying Framework. *International Journal of Computer Vision*, 56(3):221–255, February 2004.

[4] Robert C. Baumann. Radiation-Induced Soft Errors in Advanced Semiconductor Technologies. *IEEE Transactions on Device and Materials Reliability*, 5(3):305–316, September 2005.

[5] Carl Bender, Pia N. Sanda, Prabhakar Kudvaand Ricardo Mata, Vikas Pokala, Ryan Haraden, and Matthew Schallhorn. Soft-Error Resilience of the IBM POWER6 Processor Input/Output Subsystem. *IBM Journal of Research and Development*, 52(3):285–292, May 2008.

[6] Christian Bienia, Sanjeev Kumar, Jaswinder P. Singh, and Kai Li. The PARSEC Benchmark Suite: Characterization and Architectural Implications. In *International Conference on Parallel Architectures and Compilation Techniques*, 2008.

[7] Nathan L. Binkert, Ronald G. Dreslinski, Lisa R. Hsu, Kevin T. Lim, Ali G. Saidi, and Steven K. Reinhardt. The M5 Simulator: Modeling Networked Systems. *IEEE Micro*, 26(4):52–60, July 2006.

[8] Arijit Biswas, Paul Racunas, Razvan Cheveresan, Joel Emer, Shubhendu S. Mukherjee, and Ram Rangan. Computing Architectural Vulnerability Factors for Address-Based Structures. In *International Symposium on Computer Architecture*, 2005.

[9] Shekhar Borkar and Andrew A Chien. The Future of Microprocessors. *Communication of the ACM*, 54(5):67–77, May 2011.

[10] David Brooks, Vivek Tiwari, and Margaret Martonosi. Wattch: A Framework for Architectural-level Power Analysis and Optimizations. In *International Symposium on Computer Architecture*, 2010.

[11] J. Adam Butts and Gurindar S. Sohi. A Static Power Model for Architects. In *International Symposium on Microarchitecture*, 2000.

[12] Benton H. Calhoun, Sudhanshu Khanna, Randy Mann, and Jiajing Wang. Sub-threshold Circuit Design with Shrinking CMOS Devices. In *International Symposium on Circuits and Systems*, March 2009.

[13] João Carreira, Henrique Madeira, and João Gabriel Silva. Xception: A Technique for the Experimental Evaluation of Dependability in Modern Computers. *IEEE Transactions on Software Engineering*, 24(2):125–136, February 1998.

[14] Tianshi Chen, Zidong Du, Ninghui Sun, Jia Wang, Chengyong Wu, Yunji Chen, and Olivier Temam. DianNao: A Small-footprint High-throughput Accelerator for Ubiquitous Machine-learning. In *International Conference on Architectural Support for Programming Languages and Operating Systems*, 2014.

[15] Hyungmin Cho, Shahrzad Mirkhani, Chen-Yong Cher, Jacob A. Abraham, and Subhasish Mitra. Quantitative Evaluation of Soft Error Injection Techniques for Robust System Design. In *Design Automation Conference*, 2013.

[16] Eric S. Chung, Peter A. Milder, James C. Hoe, and Ken Mai. Single-Chip Heterogeneous Computing: Does the Future Include Custom Logic, FPGAs, and GPGPUs? In *International Symposium on Microarchitecture*, 2010.

[17] Jason Cong, Zhenman Fang, Michael Gill, and Glenn Reinman. PARADE: A Cycle-Accurate Full-System Simulation Platform for Accelerator-Rich Architectural Design and Exploration. In *International Conference on Computer-Aided Design*, 2015.

[18] Intel Corporation. Intel® Core™2 Quad Processor Q9550S. `http://ark.intel.com/products/40815`.

[19] Intel Corporation. Intel® Xeon® Processor W5590. `http://ark.intel.com/products/41643`.

[20] Oracle Corporation. Oracle SPARC T4 Processor. `http://www.oracle.com/us/products/servers-storage/servers/sparc-enterprise/t-series/sparc-t4-processor-ds-497205.pdf`.

[21] Edward W. Czeck and Daniel P. Siewiorek. Effects of Transient Gate-Level Faults on Program Behavior. In *International Symposium on Fault-Tolerant Computing*, 1990.

[22] DARPA. DARPA PERFECT Program. `http://www.darpa.mil/Our_Work/MTO/Programs/Power_Efficiency_Revolution_for_Embedded_Computing_Technologies_(PERFECT).aspx`.

[23] Jr. David C. Munson, James D. O'Brien, and W.Kenneth Jenkins. A Tomographic Formulation of Spotlight-Mode Synthetic Aperture Radar. *Proceedings of the IEEE*, 71(8):917–925, August 1983.

[24] Mita D. Desai and W. Kenneth Jenkins. Convolution Backprojection Image Reconstruction for Spotlight Mode Synthetic Aperture Radar. *IEEE Transactions on Image Processing*, 1(4):505–517, October 1992.

[25] Vinay Devadas and Hakan Aydin. Real-Time Dynamic Power Management Through Device Forbidden Regions. In *IEEE Real-Time and Embedded Technology and Applictions Symposium*, 2008.

[26] Hadi Esmaeilzadeh, Emily Blem, Renee St. Amant, Karthikeyan Sankaralingam, and Doug Burger. Dark Silicon and the End of Multicore Scaling. In *International Symposium on Computer Architecture*, 2011.

[27] David Fick, Ronald G. Dreslinski, Bharan Giridhar, Gyouho Kim, Sangwon Seo, Matthew Fojtik, Sudhir Satpathy, Yoonmyung Lee, Daeyeon Kim, Nurrachman Liu, Michael Wieckowski, Gregory K. Chen, Trevor N. Mudge, Dennis Sylvester, and David Blaauw. Centip3De: A 3930DMIPS/W Configurable Near-threshold 3D Stacked System with 64 ARM Cortex-M3 Cores. In *International Solid-State Circuits Conference*, 2012.

[28] Michael Floyd, Malcolm Allen-Ware, Karthick Rajamani, Bishop Brock, Charles Lefurgy, Alan J. Drake, Lorena Pesantez, Tilman Gloekler, Jose A. Tierno, Pradip Bose, and Alper Buyuktosunoglu. Introducing the Adaptive Energy Management Features of the Power7 Chip. *IEEE Micro*, 31(2):60–75, March 2011.

[29] Félix-Antoine Fortin, François-Michel De Rainville, Marc-André Gardner, Marc Parizeau, and Christian Gagné. DEAP: Evolutionary Algorithms Made Easy. *The Journal of Machine Learning Research*, 13(1):2171–2175, July 2012.

[30] LeRoy A. Gorham and Linda J. Moore. SAR Image Formation Toolbox for MATLAB. In *Proceedings of Algorithms for Synthetic Aperture Radar Imagery*, 2010.

[31] Venkatraman Govindaraju, Chen-Han Ho, and Karthikeyan Sankaralingam. Dynamically Specialized Datapaths for Energy Efficient Computing. In *International Symposium on High-Performance Computer Architecture*, 2011.

[32] Rehan Hameed, Wajahat Qadeer, Megan Wachs, Omid Azizi, Alex Solomatnikov, Benjamin C. Lee, Stephen Richardson, Christos Kozyrakis, and Mark Horowitz. Understanding

Sources of Inefficiency in General-purpose Chips. In *International Symposium on Computer Architecture*, 2010.

[33] Siva Kumar Sastry Hari, Man-Lap Li, Pradeep Ramachandran, Byn Choi, and Sarita V. Adve. mSWAT: Low-Cost Hardware Fault Detection and Diagnosis for Multicore Systems. In *International Symposium on Microarchitecture*, 2009.

[34] Johann Hauswald, Michael A. Laurenzano, Yunqi Zhang, Cheng Li, Austin Rovinski, Arjun Khurana, Ron Dreslinski, Trevor Mudge, Vinicius Petrucci, Lingjia Tang, and Jason Mars. Sirius: An Open End-to-End Voice and Vision Personal Assistant and Its Implications for Future Warehouse Scale Computers. In *International Conference on Architectural Support for Programming Languages and Operating Systems*, 2015.

[35] Mark D Hill and Michael R Marty. Amdahl's Law in the Multicore Era. *Computer*, 41(7):33–38, July 2008.

[36] Wei Huang, Karthick Rajamani, Mircea R. Stan, and Kevin Skadron. Scaling with Design Constraints: Predicting the Future of Big Chips. *IEEE Micro*, 31(4):16–29, July 2011.

[37] Texas Instruments Inc. OMAP4470 Mobile Application Processor. `http://focus.ti.com/pdfs/wtbu/OMAP4470_07-05-v2.pdf`.

[38] Nangate Incorparation. Nangate FreePDK45 Generic Open Cell Library. `http://www.si2.org/openeda.si2.org/projects/nangatelib`.

[39] Canturk Isci, Alper Buyuktosunoglu, Chen-Yong Cher, Pradip Bose, and Margaret Martonosi. An Analysis of Efficient Multi-Core Global Power Management Policies: Maximizing Performance for a Given Power Budget. In *International Symposium on Microarchitecture*, 2006.

[40] Andrew B. Kahng, Bill Lin, and Siddhartha Nath. Explicit Modeling of Control and Data for Improved NoC Router Estimation. In *Design Automation Conference*, 2012.

[41] Ghani A. Kanawati, Nasser A. Kanawati, and Jacob A. Abraham. FERRARI: A Flexible Software-Based Fault and Error Injection System. *IEEE Transactions on Computers*, 44(2):248–260, February 1995.

[42] Karim Kanoun, Nicholas Mastronarde, David Atienza, and Mihaela van der Schaar. Online Energy-Efficient Task-Graph Scheduling for Multicore Platforms. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2014.

[43] Seongwoo Kim and Arun K. Somani. Soft Error Sensitivity Characterization for Microprocessor Dependability Enhancement Strategy. In *International Conference on Dependable Systems and Networks*, 2002.

[44] Prabhakar Kudva, Jeffrey W. Kellington, Pia N. Sanda, Ryan McBeth, John Schumann, and Ron Kalla. Fault Injection Verification of IBM POWER6 Soft Error Resilience. In *Proceedings of the Workshop on Architectural Support for Gigascale Integration*, 2007.

[45] Charles. Lefurgy, Xiaorui Wang, and Malcolm Ware. Server-Level Power Control. In *Proceedings of Autonomic Computing*, 2007.

[46] Man-Lap Li, Ramachandran Pradeep, Swarup Kumar Sahoo, Sarita V. Adve, Vikram S. Adve, and Yuanyuan Zhou. Understanding the Propagation of Hard Errors to Software and Implications for Resilient System Design. In *International Conference on Architectural Support for Programming Languages and Operating Systems*, 2008.

[47] Man-Lap Li, Pradeep Ramachandran, Swamp K. Sahoo, Sarita V. Adve, Vikram S. Adve, and Yuanyuan Zhou. Trace-Based Microarchitecture-Level Diagnosis of Permanent Hardware Faults. In *International Conference on Dependable Systems and Networks*, 2008.

[48] 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 *International Symposium on Microarchitecture*, 2009.

[49] Xiaodong Li, Sarita V. Adve, Pradip Bose, and Jude A. Rivers. SoftArch: An Architecture-Level Tool for Modeling and Analyzing Soft Errors. In *International Conference on Dependable Systems and Networks*, 2005.

[50] Xiaodong Li, Sarita V. Adve, Pradip Bose, and Jude A. Rivers. Architecture-Level Soft Error Analysis: Examining the Limits of Common Assumptions. In *International Conference on Dependable Systems and Networks*, 2007.

[51] Henrique S. Malvar, Li-Wei He, and Ross Cutler. High-Quality Linear Interpolation for Demosaicing of Bayer-Patterned Color Images. In *International Conference of Acoustic, Speech and Signal Processing*, 2004.

[52] John E Mitchell. Branch-and-Cut Algorithms for Combinatonial Optimization Problems. *Handbook of Applied Optimization*, pages 65–77, January 2002.

[53] Gordon E. Moore. Cramming More Components onto Integrated Circuits. *Electronics*, 38(8):114–117, April 1965.

[54] Gordon E. Moore. Progress in Digital Integrated Electronics. In *International Electron Devices Meeting*, volume 21, pages 11–13, 1975.

[55] Shubhendu S. Mukherjee, Christopher Weaver, Joel Emer, Steven K. Reinhardt, and Todd Austin. A Systematic Methodology to Compute the Architectural Vulnerability Factors for a High-Performance Microprocessor. In *International Symposium on Microarchitecture*, 2003.

[56] Hashem H. Najaf-abadi, Niket K. Choudhary, and Eric Rotenberg. Core-Selectability in Chip Multiprocessors. In *International Conference on Parallel Architectures and Compilation Techniques*, 2009.

[57] Ramanathan Narayanan, Berkin Özısıyılmaz, Joseph Zambreno, Gokhan Memik, and Alok Choudhary. MineBench: A Benchmark Suite for Data Mining Workloads. In *IEEE International Symposium on Workload Characterization*, 2006.

[58] Suriyaprakash Natarajan, Melvin A. Breuer, and Sandeep K. Gupta. Process Variations and their Impact on Circuit Operation. In *IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems*, 1998.

[59] Nicholas Nethercote and Julian Seward. Valgrind: A Framework for Heavyweight Dynamic Binary Instrumentation. In *Proceedings of ACM SIGPLAN Conference on Programming Language Design and Implementation*, 2007.

[60] Kyprianos Papadimitriou, Apostolos Dollas, and Scott Hauck. Performance of Partial Reconfiguration in FPGA Systems: A Survey and a Cost Model. *ACM Transactions on Reconfigurable Technology and Systems*, 2011.

[61] Padmanabhan Pillai and Kang G. Shin. Real-Time Dynamic Voltage Scaling for Low-Power Embedded Operating Systems. In *ACM Symposium on Operating Systems Principles*, 2001.

[62] Andrew Putnam, Adrian M. Caulfield, Eric S. Chung, Derek Chiou, Kypros Constantinides, John Demme, Hadi Esmaeilzadeh, Jeremy Fowers, Gopi Prashanth Gopal, Jan Gray, Michael Haselman, Scott Hauck, Stephen Heil, Amir Hormati, Joo-Young Kim, Sitaram Lanka, James Larus, Eric Peterson, Simon Pope, Aaron Smith, Jason Thong, Phillip Yi Xiao, and Doug Burger. A Reconfigurable Fabric for Accelerating Large-scale Datacenter Services. In *International Symposium on Computer Architecture*, 2014.

[63] Xuan Qi and Dakai Zhu. Power Management for Real-Time Embedded Systems on Block-Partitioned Multicore Platforms. In *IEEE International Conference on Embedded Software and Systems*, 2008.

[64] Jan M. Rabaey, Anantha Chandrakasan, and Borivoje Nikolić. *Digital Integrated Circuits: A Design Perspective*. Prentice Hall, second edition, 2003.

[65] Jude A. Rivers, Pradip Bose, Prabhakar Kudva, John-David Wellman, Pia N. Sanda, Ethan H. Cannon, and Luiz C. Alves. Phaser: Phased Methodology for Modeling the System-Level Effects of Soft Errors. *IBM Journal of Research and Development*, 52(3):293–306, 2008.

[66] K. P. Rodbell. Trends in the SEE and TID Radiation Response of 65-, 45- and 32-nm SOI CMOS Transistors, Memories, and Latches. In *Proceedings of the GOMACTech Conference*, 2012.

[67] Paul Rosenfeld, Elliott Cooper-Balis, and Bruce Jacob. DRAMSim2: A Cycle Accurate Memory System Simulator. *IEEE Computer Architecture Letters*, 2011.

[68] Swamp Kumar Sahoo, Man-Lap Li, Pradeep Ramachandran, Sarita V. Adve, Vikram S. Adve, and Yuanyuan Zhou. Using Likely Program Invariants to Detect Hardware Errors. In *International Conference on Dependable Systems and Networks*, 2008.

[69] Pia N. Sanda, Jeffrey W. Kellington, Prabhakar Kudva, Ronald N. Kalla, Ryan B. McBeth, Jerry Ackaret, Ryan Lockwood, John Schumann, and Christopher R. Jones. Soft-Error Resilience of the IBM POWER6 Processor. *IBM Journal of Research and Development*, 52(3):275–284, May 2008.

[70] Norbert Seifert, Balkaran Gill, Shah Jahinuzzaman, Joseph Basile, Vinod Ambrose, Quan Shi, Randy Allmon, and Arkady Bramnik. Soft Error Susceptibilities of 22nm Tri-Gate Devices. *IEEE Transactions on Nuclear Science*, 59(6), December 2012.

[71] Rishad Ahmed Shafik, Bashir M. Al-Hashimi, Sandip Kundu, and Alireza Ejlali. Soft Error-Aware Voltage Scaling Technique for Power Minimization in Application-Specific Multiprocessor System-on-Chip. *Journal of Low Power Electronics*, 5(2):145–156, August 2009.

[72] Mingsheng Shang, Shixin Sun, and Qingxian Wang. An Efficient Parallel Scheduling Algorithm of Dependent Task Graphs. In *International Conference on Parallel and Distributed Computing, Applications and Technologies*, 2003.

[73] Yakun Sophia Shao, Brandon Reagen, Gu-Yeon Wei, and David Brooks. Aladdin: A Pre-RTL, Power-Performance Accelerator Simulator Enabling Large Design Space Exploration of Customized Architectures. In *International Symposium on Computer Architecture*, 2014.

[74] Premkishore Shivakumar, Michael Kistler, Stephen W. Keckler, Doug Burger, and Lorenzo Alvisi. Modeling the Effect of Technology Trends on the Soft Error Rate of Combinational Logic. In *International Conference on Dependable Systems and Networks*, 2002.

[75] Saurabh Sinha, Greg Yeric, Vikas Chandra, Brian Cline, and Yu Cao. Exploring Sub-20Nm FinFET Design with Predictive Technology Models. In *Design Automation Conference*, 2012.

[76] Balaram Sinharoy, Ron Kalla, William J. Starke, Hung Q. Le, Robert Cargnoni, James A. Van Norstrand, Bruce J. Ronchetti, Jeffrey Stuecheli, Jens Leenstra, Guy L. Guthrie, Dung Q. Nguyen, Bart Blaner, Charles F. Marino, Eric Retter, and Phil Williams. IBM POWER7 Multicore Server Processor. *IBM Journal of Research and Development*, 55(3):191–219, May 2011.

[77] Daniel Skarin, Raul Barbosa, and Johan Karlsson. GOOFI-2: A Tool for Experimental Dependability Assessment. In *International Conference on Dependable Systems and Networks*, 2010.

[78] Chris Stauffer and W. E L Grimson. Adaptive Background Mixture Models for Real-Time Tracking. In *IEEE Conference on Computer Vision and Pattern Recognition*, 1999.

[79] Michael B. Taylor. Is Dark Silicon Useful?: Harnessing The Four Horsemen of the Coming Dark Silicon Apocalypse. In *Design Automation Conference*, 2012.

[80] Anna Thomas and Karthik Pattabiraman. Error Detector Placement for Soft Computation. In *International Conference on Dependable Systems and Networks*, 2013.

[81] Amanda Chih-Ning Tseng and David Brooks. Analytical Latency-Throughput Model of Future Power Constrained Multicore Processors. In *Workshop on Energy-Efficient Design*, 2012.

[82] Augusto Vega, Alper Buyuktosunoglu, Heather Hanson, Pradip Bose, and Srinivasan Ramani. Crank It Up or Dial It Down: Coordinated Multiprocessor Frequency and Folding Control. In *International Symposium on Microarchitecture*, 2013.

[83] Sravanthi Kota Venkata, Ikkjin Ahn, Donghwan Jeon, Anshuman Gupta, Christopher Louie, Saturnino Garcia, Serge Belongie, and Michael Bedford Taylor. SD-VBS: The San Diego Vision Benchmark Suite. In *IEEE International Symposium on Workload Characterization*, 2009.

[84] 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 *International Conference on Architectural Support for Programming Languages and Operating Systems*, 2010.

[85] Liang Wang, Ramon Bertran, Alper Buyuktosunoglu, Pradip Bose, and Kevin Skadron. Characterization of Transient Error Tolerance for a Class of Mobile Embedded Applications. In *IEEE International Symposium on Workload Characterization*, 2014.

[86] Liang Wang, Jude A. Rivers, Meeta S. Gupta, Augusto J. Vega, Alper Buyuktosunoglu, Pradip Bose, and Kevin Skadron. Resilience and Real-Time Constrained Energy Optimization in Embedded Processor Systems,. In *IEEE Workshop on Silicon Errors in Logic - System Effects*, 2014.

[87] Liang Wang and Kevin Skadron. Implications of the Power Wall: Dim Cores and Reconfigurable Logic. *IEEE Micro*, 33(5):40–48, September 2013.

[88] Liang Wang, Augusto J. Vega, Alper Buyuktosunoglu, Pradip Bose, and Kevin Skadron. Power-efficient embedded processing with resilience and real-time constraints. In *IEEE/ACM International Symposium on Low Power Electronics and Design*, 2015.

[89] Nicholas J. Wang, Justin Quek, Todd M. Rafacz, and Sanjay J. Patel. Characterizing the Effects of Transient Faults on a High-Performance Processor Pipeline. In *International Conference on Dependable Systems and Networks*, 2004.

[90] Jiesheng Wei, Anna Thomas, Guanpeng Li, and Karthik Pattabiraman. Quantifying the Accuracy of High-Level Fault Injection Techniques for Hardware Faults. In *International Conference on Dependable Systems and Networks*, 2014.

[91] Lisa Wu and Martha A. Kim. Acceleration Targets: A Study of Popular Benchmark Suites. In *Dark Silicon Workshop*, 2012.

[92] Lisa Wu, Andrea Lottarini, Timothy K. Paine, Martha A. Kim, and Kenneth A. Ross. Q100: The Architecture and Design of a Database Processing Unit. In *International Conference on Architectural Support for Programming Languages and Operating Systems*, 2014.

[93] Yavuz Yetim, Sharad Malik, and Margaret Martonosi. EPROF: An Energy/Performance/Reliability Optimization Framework for Streaming Applications. In *Asia and South Pacific Design Automation Conference*, 2012.

[94] Ying Zhang and Krishnendu Chakrabarty. A Unified Approach for Fault Tolerance and Dynamic Power Management in Fixed-Priority Real-Time Embedded Systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 25(1):111–125, January 2006.

[95] Baoxian Zhao, H. Aydin, and Dakai Zhu. Reliability-Aware Dynamic Voltage Scaling for Energy-Constrained Real-Time Embedded Systems. In *International Conference on Computer Design*, 2008.

[96] Tsahee Zidenberg, Isaac Keslassy, and Uri Weiser. MultiAmdahl: How Should I Divide My Heterogeneous Chip? *IEEE Computer Architecture Letters*, 11(2):65–68, July 2012.

[97] J.F. Ziegler, H. W. Curtis, H.P. Muhlfeld, C.J. Montrose, B. Chin, M. Nicewicz, C. A. Russell, W. Y. Wang, L. B. Freeman, P. Hosier, L. E. LaFave, J.L. Walsh, J. M. Orro, G. J. Unger, J. M. Ross, T.J. O'Gorman, B. Messina, T.D. Sullivan, A. J. Sykes, H. Yourke, T. A. Enger, V. Tolat, T. S. Scott, A. H. Taber, R. J. Sussman, W. A. Klein, and C. W. Wahaus. IBM Experiments in Soft Fails in Computer Electronics (1978-1994). *IBM Journal of Research and Development*, 40(1):3–18, January 1996.