SOFTWARE, ARCHITECTURE, AND VLSI CO-DESIGN FOR FINE-GRAIN VOLTAGE AND FREQUENCY SCALING

A Dissertation

Presented to the Faculty of the Graduate School

of Cornell University

in Partial Fulfillment of the Requirements for the Degree of

Doctor of Philosophy

by
Christopher Lin Torng
December 2019

SOFTWARE, ARCHITECTURE, AND VLSI CO-DESIGN
FOR FINE-GRAIN VOLTAGE AND FREQUENCY SCALING

Christopher Lin Torng, Ph.D.
Cornell University 2019

As emerging domains demand higher performance under stringent constraints on power and energy, computer architects are increasingly relying on a combination of parallelization and specialization to improve both performance and energy efficiency. However, the combination of parallelism and specialization is also steadily increasing on-chip asymmetry in the form of spatial heterogeneity and temporal variation, which poses key challenges in the form of widely varying utilization in space (i.e., across different components) and in time (i.e., used at different times across varying performance levels). Fine-grain on-chip asymmetry requires analogously fine-grain power-control techniques in order to power (or not power) different components to different levels at just the right times to significantly reduce waste. At the same time, traditional walls of abstraction have broken down, allowing a cross-stack co-design approach across software, architecture, and VLSI to provide new, previously inaccessible information to precisely control new hardware mechanisms.

This thesis explores novel fine-grain voltage and frequency scaling techniques to improve both performance and energy efficiency with software, architecture, and VLSI co-design. First, I explore architecture-circuit co-design and leverage recent work on fully integrated voltage regulation to enable realistic fine-grain voltage and frequency scaling for homogeneous systems of little cores at microsecond timescales. Second, I broaden the scope to heterogeneous systems of big and little cores and specialize for productive software task-based parallel runtimes. Third, I investigate much finer-grain asymmetry that can be exploited within coarse-grain reconfigurable arrays, which have recently attracted significant interest due to their flexibility and potential for reducing data movement energy. Finally, I describe my work on four silicon prototypes including a mixed-signal test chip and three digital ASIC test chips that support different aspects of my thesis.

Throughout my thesis, I take a software, architecture, and VLSI co-design approach and focus on exploiting information newly exposed across layers of abstraction. I leverage a vertically

integrated research methodology spanning across applications, runtimes, architecture, cycle-level modeling, RTL, VLSI CAD tools, SPICE-level modeling, and silicon prototyping to evaluate the potential benefit of fine-grain voltage and frequency scaling techniques.

BIOGRAPHICAL SKETCH

Christopher Torng was born on July 25, 1990 to Ming Hwa Torng and Liyin Lin Torng. He is a middle child with a sister three years older and a brother three years younger. As the son of a piano teacher and a computer science engineer working in finance, he grew up with music and also learned basic programming from a young age. In his early education, Chris naturally developed just those skills, playing the trumpet and the piano for about a decade, writing fun programs, pole vaulting with the track team, and eventually graduating from his high school.

Chris was accepted to Cornell University as an undergraduate student and began as a materials scientist. However, he quickly switched to electrical and computer engineering, where he bonded with the simple abstractions of switches with which so much could be done. After passing through an internship at Intel, he picked up an industry perspective and also some much-needed vim text-editing skills. The corresponding academic perspective came after he joined Professor Christopher Batten's group at Cornell, first as a teaching assistant for computer architecture and then as an undergraduate researcher just as the group was being formed. After seeing first-hand how rigorous research is done, he was inspired to begin his PhD career with Professor Christopher Batten.

Throughout the next seven years, he learned a tremendous amount about software, about computer architecture, about VLSI, and about how to do "good work". After getting involved in silicon prototyping team projects, he discovered how engaging and rewarding it can be to work with younger students on project goals as well as mentorship goals. He was inspired to continue on the academic track after graduation.

In his free time during the PhD, Chris was an avid figure skater and singer. Chris visited the ice rink to figure skate every semester, and he continues to work on his spins, jumps, and moves in the field. He also spent time directing an a cappella group on campus for a few years. As an unusually technology-oriented music director, he arranged new music on the command line in text-based music notation, compiling the music notation into sheet music PDF with open-source rendering software and a makefile.

Chris is very thankful for his PhD experience. Despite the ups and downs, the period was a time for invaluable personal growth under a great advisor and together with good friends. Lastly, it was also a time to meet his beloved fiancée Yingqiu Cao.

This document is dedicated to my parents and my beloved fiancée Yingqiu Cao

ACKNOWLEDGEMENTS

My graduate career has been a roller-coaster ride. So much has happened that I am thankful for, and I have many friends to thank for making the ride enjoyable and worthwhile.

My advisor Christopher Batten is the best advisor I could imagine having. There are many ways in which he helped me grow that simply did not have to be done for me. When I expressed interest in teaching, he invited me to get involved. When I wanted to try my hand at reviewing papers, he somehow found a great journal in my rough area that would allow a young graduate student to review. During my time working with Chris, I learned that there are no hyphens after -ly for compound modifiers, that we should use the Oxford comma, that vertically integrated research is the best, and that whiteboard walls are great. Chris was also patient enough to let me go through all of my odd phases experimenting with task management and information management during which I was probably not a very useful graduate student. I think that without Chris, I would not have anywhere near the level of attention to detail (e.g., with structured writing) that I have now.

I am also deeply thankful to my PhD friends in the Batten Research Group for all of their support through the years. I am thankful in particular for Shreesha Srinath, who always had interesting topics to discuss, new papers to look at, and was always so supportive even during the harder phases of PhD life. I am also very grateful for Derek Lockhart and Ji Yun Kim as senior role models, telling me it is fine to have missed a group meeting, and inspiring all of CSL to be better researchers within a tight-knit community. Berkin Ilbeyi entered in the same year I did and was always ingeniously and magically speeding things up with JIT magic. As I built my weird chips way down there, I am glad he was there to put the upper end of the computing stack into perspective for me. The remaining BRG members are all younger, but while I have become the mentor due to seniority, they are no less impactful to me. Moyang Wang, I will always remember your cat picture on Github. Shunning Jiang, you have really put family matters in perspective for me, and I am really really looking forward to seeing you learn and grow. Khalid Al-Hawaj, you should pull fewer late nights. Tuan Ta and Lin Cheng, you were both magical hackers in BRGTC2 and you know way more about gem5, pypy, and coherence policies than I will ever know. Yanghui Ou and Peitian Pan, it is amazing how much you can do and how quickly you pick things up. Shady Agwa, it was great having you in the group to chat about VLSI. Cheng Tan, you will do great bro. Finally, I want to thank some younger students for being great fun to work with: Bharath Sudheendra, Nagaraj Murali, Rohan Agarwal, Eric Tang, Cameron Haire.

Within the CSL community, I want to give a special callout to Saugata Ghose, who has really been an amazing role model for me even though he left early on in my PhD. Saugata, you are such an engaging researcher and friend, and I know that wherever you go, you will find a way not only to be satisfied with your work but also to be an expert tour guide for the exact history of your area. Ritchie Zhao and Steve Dai, I am glad you were both around to chat about anything and everything in the office. Suren Jayasuriya, you are not actually CSL but I am putting you here, and that is enough said.

To my friends in the circuits side of ECE: Professor Alyssa Apsel, thanks for always being very accepting of me even when I was a young architect somewhat clueless about the circuits side of things. I am very thankful to you for that attitude. I spent many hours with Waclaw Godycki and Ivan Bukreyev talking about everything from basic circuits to advanced topics to broader life. Talking with both of you really opened my world to the analog side.

I would also like to acknowledge Celerity friends. Professor Ronald G. Dreslinski was an amazing mentor to me during the two months I spent as a visiting research affiliate at the University of Michigan. I was struck by how much you invested in your students from your heart and I hope to do the same with others someday. Austin Rovinski, Tutu Ajayi, and Aporva Amarnath, we have way too many memories in a short period of time. Thanks Tutu, for taking that video of us all sleeping under desks the night before the tapeout. From UCSD/UWashington, I am really happy I could meet Professor Michael B. Taylor and his own really intense research group. There are more people here I would like to thank: Julian Puscar, Scott Davidson, Paul Gao, Shaolin Xie, and others I worked with directly.

From the broader architecture community, I am very indebted to some key friends and mentors. Hung-wei Tseng, I am so glad that I met you randomly at that lunch table, and I am glad that we got to be good friends as well as colleagues. I would like to thank Brucek Khailany, Michael Pellauer, and Sophia Shao for their advice and support throughout my academic job search. Similar thanks go out to Chris Fletcher, Matt Sinclair, Tony Nowatzki, Brandon Lucia, and David Brooks.

Finally, I would like to thank my parents for supporting me in both my career and my personal life throughout the PhD. I always look forward to coming home to chat with you, and I hope to still do so in the future despite being on the other side of the country. Rebecca Torng and Thomas Torng, I am looking forward to sending more dumb stuff to you two. Yingqiu Cao, you are the

most meaningful thing that happened in my entire PhD career. I hope to make you as happy in the future as you make me.

In terms of funding, this thesis was supported in part by NSF CAREER Award #1149464, a Spork Fellowship, AFOSR YIP Award #FA9550-15-1-0194, the MOSIS Educational Program, NSF XPS Award #1337240, NSF SHF Award #1527065, DARPA POSH Award #FA8650-18-2-7852, DARPA CRAFT Award HR0011-16-C-0037, NSF CRI Award #1059333, NSF CRI Award #1512937, NSF SaTC Award #1563767, NSF SaTC #1565446, EDA tool donations from Synopsys, Cadence, and Mentor, physical IP donations from ARM, EDA tool and FPGA development board donations from Xilinx, and donations from Intel Corporation and Synopsys, Inc. NSF CAREER Award #1149464 NSF XPS Award #1337240, NSF SHF Award #1527065, NSF CRI Award #1059333, NSF CRI Award #1512937, AFOSR YIP Award #FA9550-15-1-0194, DARPA YFA Award #N66001-12-1-4239, DARPA CRAFT Award #HR0011-16-C-0037, DARPA SDH Award #FA8650-18-2-7863, a Spork Fellowship, the MOSIS Educational Program, and equipment, tool, and/or physical IP donations from Intel, NVIDIA, Xilinx, Synopsys, and ARM.

TABLE OF CONTENTS

Biographical Sketch . . . iii
Dedication . . . iv
Acknowledgements . . . v
Table of Contents . . . viii
List of Figures . . . xi
List of Tables . . . xiii
List of Abbreviations . . . xiv

1 Introduction **1**
1.1 A Growing Trend Towards On-Chip Asymmetry . . . 2
1.2 The Need for Fine-Grain Power Control . . . 4
1.3 Breaking Through Walls of Abstraction for Better Control . . . 6
1.4 Thesis Overview . . . 7
1.5 Collaboration and Funding . . . 10

2 Reconfigurable Power Distribution Networks **14**
2.1 Introduction . . . 14
2.2 Target System . . . 18
2.3 FGVS Architecture Design: FG-SYNC+ . . . 18
2.3.1 Basic FG-SYNC+ Controller . . . 19
2.3.2 FG-SYNC+ with Fine-Grain Scaling in Level . . . 20
2.3.3 FG-SYNC+ with Fine-Grain Scaling in Space . . . 21
2.3.4 FG-SYNC+ with Fine-Grain Scaling in Time . . . 23
2.3.5 FG-SYNC+ Summary . . . 24
2.4 FGVS Circuit Design: RPDNs . . . 25
2.4.1 SFVR: Single Fixed-Voltage Regulator . . . 26
2.4.2 MAVR: Multiple Adjustable-Voltage Regulators . . . 27
2.4.3 RPDN: Reconfigurable Power Distribution Networks . . . 29
2.4.4 Summary of Power Distribution Networks . . . 32
2.5 Evaluation Methodology . . . 32
2.5.1 Circuit-Level Modeling . . . 32
2.5.2 Gate- and RTL Modeling . . . 33
2.5.3 Cycle-Level Modeling . . . 34
2.5.4 Application Kernels and Benchmarks . . . 35
2.6 Evaluation Results . . . 36
2.7 Discussion . . . 39
2.8 Related Work . . . 41
2.9 Conclusion . . . 42

3 Asymmetry-Aware Work-Stealing Runtimes **43**
3.1 Introduction . . . 43
3.2 A Marginal-Utility-Based Approach . . . 47
3.2.1 First-Order Model . . . 47

3.2.2 Marginal Utility Optimization Problem . . . 49
3.2.3 Marginal Utility in the High-Parallel Region . . . 51
3.2.4 Marginal Utility in the Low-Parallel Region . . . 52
3.2.5 Leakage Sensitivity Study . . . 53
3.3 AAWS Runtimes . . . 57
3.3.1 Work-Pacing and Work-Sprinting . . . 57
3.3.2 Work-Mugging . . . 59
3.3.3 Serial-Sprinting and Work-Biasing . . . 60
3.4 Evaluation Methodology . . . 61
3.4.1 Target System . . . 61
3.4.2 Benchmark Suite . . . 62
3.4.3 Work-Stealing Runtime . . . 62
3.4.4 Cycle-Level Performance Modeling . . . 64
3.4.5 Energy Modeling . . . 65
3.5 Evaluation Results . . . 66
3.5.1 Performance of Baseline Work-Stealing Scheduler . . . 67
3.5.2 Performance Analysis of Work-Pacing, Work-Sprinting, and Work-Mugging 67
3.5.3 Performance Versus Energy Analysis . . . 71
3.6 Related Work . . . 71
3.7 Conclusion . . . 74

4 Ultra-Elastic Coarse-Grain Reconfigurable Arrays 75
4.1 Introduction . . . 75
4.2 UE-CGRA Analytical Modeling . . . 78
4.2.1 Discrete-Event Performance Model . . . 78
4.2.2 First-Order Energy Model . . . 79
4.2.3 Analytical Case Study . . . 81
4.3 UE-CGRA Compiler . . . 81
4.4 UE-CGRA Architecture . . . 83
4.5 UE-CGRA VLSI . . . 85
4.6 Methodology . . . 87
4.6.1 Benchmarks . . . 87
4.6.2 Compiler . . . 87
4.6.3 Architecture and VLSI Modeling . . . 88
4.6.4 Energy Modeling . . . 88
4.7 Results . . . 89
4.7.1 Tile Area and Energy . . . 89
4.7.2 CGRA Area and Cycle Time . . . 91
4.7.3 Mapping Kernels . . . 91
4.7.4 Performance and Energy Efficiency . . . 92
4.8 Related Work . . . 93
4.9 Conclusion . . . 96

5 Silicon Prototyping with Fine-Grain Voltage and Frequency Scaling **97**
5.1 Dynamic Capacitance Sharing (DCS) . . . 97
5.1.1 Research Purpose . . . 97
5.1.2 Chip Details . . . 99
5.2 Batten Research Group Test Chip 1 (BRGTC1) . . . 99
5.2.1 Research Purpose . . . 100
5.2.2 Chip Details . . . 101
5.3 Batten Research Group Test Chip 2 (BRGTC2) . . . 105
5.3.1 Research Purpose . . . 105
5.3.2 Chip Details . . . 105
5.4 The Celerity Open-Source 511-Core RISC-V Tiered Accelerator Fabric . . . 110
5.4.1 Research Purpose . . . 110

6 Conclusion **113**
6.1 Thesis Summary and Contributions . . . 113
6.2 Potential Long-Term Impacts: Reconfigurable Power-Distribution Networks . . . 115
6.3 Potential Long-Term Impacts: Asymmetry-Aware Work-Stealing Runtimes . . . 117
6.4 Potential Long-Term Impacts: Ultra-Elastic Coarse-Grain Reconfigurable Arrays . 120
6.5 Future Work . . . 122

Bibliography **125**

LIST OF FIGURES

1.1 Specialization Spectrum . . . 2
1.2 On-Chip Asymmetry . . . 3
1.3 Space-Time Granularities for Power-Control Mechanisms . . . 5
1.4 Thesis Overview . . . 8

2.1 Activity Profile for Select Applications on Eight Cores . . . 17
2.2 Lookup Table Mapping Activity Patterns to DVFS Modes . . . 20
2.3 FGVS Exploration in Level, Space, and Time . . . 20
2.4 Lookup Table Mapping Activity Patterns to DVFS Modes (Four Domains) . . . 22
2.5 Application Activity Plots for FG-SYNC+ . . . 23
2.6 SFVR . . . 27
2.7 MAVR . . . 28
2.8 MAVR Transient Response . . . 29
2.9 RPDN With Adjustable Unit Cells . . . 29
2.10 RPDN Power Efficiency vs. Output Power For Single Core . . . 31
2.11 RPDN Transient Response . . . 31
2.12 Tapeout-Ready Layout of a sub-RPDN Test Chip . . . 33
2.13 System-Level Evaluation for SFVR, MAVR, and RPDN . . . 37
2.14 RPDN Power Efficiency vs. Output Power For Single Core in a Leakier 65 nm Process . . . 40

3.1 Activity Profile for Convex Hull Application on Statically Asymmetric System . . . 47
3.2 Pareto-Optimal Frontier for 4B4L System . . . 50
3.3 4B4L System w/ All Cores Active . . . 51
3.4 Theoretical Speedup for 4B4L System vs. α and β . . . 52
3.5 4B4L System w/ 2B2L Active . . . 52
3.6 Leakage Comparison for Pareto-Optimal Frontier of 4B4L System . . . 54
3.7 Leakage Comparison for 4B4L System w/ All Cores Active . . . 54
3.8 Leakage Comparison for Theoretical Speedup for 4B4L System vs. α and β . . . 55
3.9 Leakage Comparison for 4B4L System w/ 2B2L Active . . . 55
3.10 Leakage Parameter Sweeps . . . 56
3.11 4B4L System w/ Hardware Support for AAWS Runtimes . . . 58
3.12 Activity Profiles for *radix-2* on 4B4L . . . 67
3.13 Activity Profiles for *hull* on 4B4L . . . 68
3.14 Normalized Execution Time Breakdown . . . 69
3.15 Energy Efficiency vs. Performance . . . 72

4.1 UE-CGRA Vision . . . 76
4.2 UE-CGRA Intuition with Analytical Modeling . . . 79
4.3 UE-CGRA Compiler Flow . . . 82
4.4 UE-CGRA Compiler Power Mapping Algorithm . . . 82
4.5 UE-CGRA Architecture and VLSI . . . 84
4.6 Tile Area vs. Cycle Latency . . . 89
4.7 Tile Energy and Area Breakdowns . . . 90

4.8 CGRA Layouts 91
4.9 Kernel DFGs 92
4.10 Kernel DFGs Mapped to UE-CGRA 92

5.1 Timeline of Silicon Prototypes 98
5.2 DCS Test Chip 98
5.3 BRGTC1 Chip Plot and Die Photo 100
5.4 BRGTC1 Detailed Pre-Silicon Methodology 101
5.5 Experimental Setup 102
5.6 Case Study for Hardware-Accelerated Sorting 103
5.7 Messages Captured by Agilent 16822A Logic Analyzer 103
5.8 BRGTC2 Chip Plot and Die Photo 106
5.9 Block Diagram of BRGTC2 107
5.10 Celerity SoC Chip Plot and Die Photo 111

LIST OF TABLES

2.1 Comparison of SFVR, MAVR, and RPDN 32
2.2 Application Performance and Energy 37

3.1 Cycle-Level System Configuration 61
3.2 Performance of Baseline Runtime vs. Intel Cilk++ and Intel TBB on Real System 63
3.3 Application Kernels 63

4.1 Benchmark Kernels 87
4.2 Energy-Optimized UE-CGRA Results 93
4.3 Performance-Optimized UE-CGRA Results 93

5.1 BRGTC1 System Parameters and Post-Silicon Resting Measurements 102

LIST OF ABBREVIATIONS

CAD	computer aided design
RTL	register-transfer level
VLSI	very-large-scale integration
ASIC	application-specific integrated circuit
DVFS	dynamic voltage and frequency scaling
FGVS	fine-grain voltage scaling
IVR	integrated voltage regulation
SCVR	switched-capacitor voltage regulator
HP	high parallelism
LP	low parallelism
DAG	directed acyclic graph
CGRA	coarse-grain reconfigurable array
FPGA	field-programmable gate array
CMP	chip multiprocessor
MIC	many integrated core
GPGPU	general-purpose graphics processing unit
SIMD	single-instruction multiple-data
SIMT	single-instruction multiple-thread
RISC	reduced instruction set computer
GPP	general-purpose processor
SoC	system-on-chip
TBB	(Intel) threading building blocks
API	application programming interface
DSL	domain-specific language
IR	intermediate representation
ISA	instruction set architecture
PC	program counter
RF	register file
LLFU	long-latency functional unit
DRC	design-rule check
LVS	layout versus schematic
HLS	high-level synthesis
SRAM	static random access memory
DRAM	dynamic random access memory
I$	instruction cache
D$	data cache

CHAPTER 1
INTRODUCTION

Emerging application domains are increasing the demand for higher compute performance. Domains including machine learning, self-driving vehicles, augmented and virtual reality, and intelligence on the edge have either promised or have already delivered real-world impact. Despite (or perhaps due to) these successes, there is significant interest in further increasing performance across the computing spectrum from servers to mobile platforms and even further to the edge. Unfortunately, achieving higher performance is not a straightforward endeavour and is in fact bottlenecked by one of the most critical grand challenges in the computing industry.

High-performance design techniques are bottlenecked by the *power wall* as well as the *energy wall*. Today, both power and energy have become first-class design constraints across all computing platforms. Specifically, high-performance design techniques require more power and generate more heat within the device. Each computing platform today operates under an effective power cap (i.e., the power wall) that is largely pre-determined by form factor, packaging, and cooling technology. Similarly, for battery-powered devices, which range from mobile phones to energy-harvesting systems operating in harsh environments, applying high-performance design techniques that are very energy-inefficient drain the energy storage too quickly for the device to be useful. Dennard constant-field scaling [DGY+74] briefly mitigated these concerns through the early 2000s, but challenges in supply voltage scaling continue to prevent further scaling. Today, these constraints threaten to worsen [Hor14] to the point where only a limited fraction of the chip can be active at once, with the remaining unpowered portion known as dark silicon [Tay13].

Architects have responded to the power and energy walls primarily with one of two high-level approaches: *parallelism* and *specialization*. With parallelism, architects step back from complex power-hungry circuits and towards smaller and more energy-efficient circuits before integrating as many as possible on a chip. Highly energy-efficient execution is possible as long as there is sufficient parallelism. With specialization, architects improve efficiency by trading off flexibility, for example by reducing programmability or by narrowing the circuit capability to a specific domain. Figure 1.1 draws a spectrum of various specialization approaches that range from accelerating parallel patterns [PZK+17, Bat10, KJT+17], to exploiting properties of spatial architec-

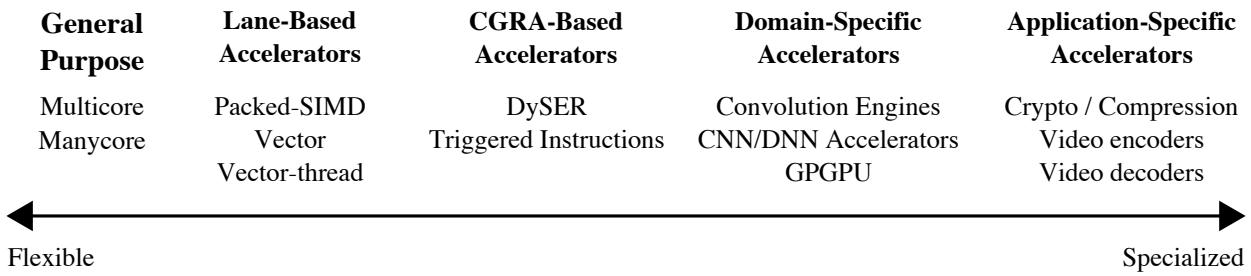


Figure 1.1: Specialization Spectrum – A spectrum of approaches for trading off flexibility for efficiency. General-purpose processors are the most flexible but also the least efficient. A full spectrum of accelerators work to reduce control, data, and memory access latency and energy by giving up the flexibility to execute arbitrary workloads.

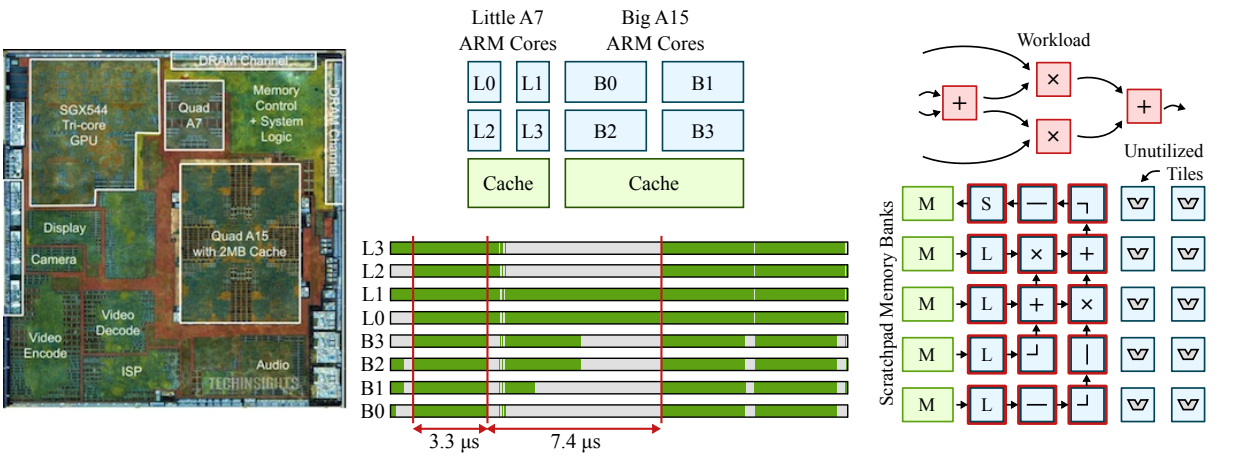
tures [CKES17, KKV+18, GHS11, PPA+13], to accelerating only specific algorithms like convolution [CKES17, QHS+13], and finally to small highly efficient but fixed-function ASICs.

This thesis is motivated by three high-level observations. The first observation is that the combination of parallelism and specialization is steadily increasing on-chip asymmetry in the form of spatial heterogeneity and temporal variation (Section 1.1). The second observation is that on-chip asymmetry results in widely varying utilization in space (i.e., across different components) and in time (i.e., used at different times across varying performance levels), which potentially demands very fine-grain power-control techniques in order to power (or not power) the different components to different levels at the right times without significant waste (Section 1.2). The final observation is that traditional walls of abstraction have broken down, allowing a cross-stack co-design approach across software, architecture, and VLSI to provide new, previously inaccessible information to control novel fine-grain power-control techniques (Section 1.3).

1.1 A Growing Trend Towards On-Chip Asymmetry

The combination of parallelism and specialization is steadily increasing on-chip asymmetry in the form of spatial heterogeneity and temporal variation. Figure 1.2 illustrates both cases and also calls attention to reconfigurable architectures that create asymmetry at configuration time.

Spatial heterogeneity can be observed in Figure 1.2(a), which shows an annotated die photo for the Samsung Exynos-5422 Octa Mobile Processor [sam19], an eight-core mobile SoC which, while somewhat dated (released in 2014), still demonstrates how SoCs today integrate a large number of heterogeneous IP on the same die. The Exynos integrates four "big" A15 cores and four "little" A7 cores together with various specialized IP blocks for display, camera, audio, video encoding and decoding, and graphics acceleration. Focusing on more recent announcements, the



(a) Spatial Heterogeneity (b) Temporal Variation (c) Configured Asymmetry

Figure 1.2: On-Chip Asymmetry – The growing popularity of parallelism and specialization is also increasing on-chip asymmetry in the form of spatial heterogeneity and temporal variation. (a) Spatial heterogeneity demonstrated by the Samsung Exynos Octa Mobile Processor with four big cores, four little cores, and various IP blocks for display, camera, audio, and video processing; (b) Temporal variation illustrated with an activity profile of a convex hull application kernel on a system with four big cores and four little cores (green = executing task; light-gray = waiting for work; see methodology in Chapter 3); (c) Configured asymmetry visualized in a coarse-grain reconfigurable array with unutilized tiles (M = memory bank, L = load, S = store).

Apple A12 Bionic chip integrates six cores with two "performance" cores and four "efficiency" cores while also integrating a second-generation "Bionic" neural engine with eight processing elements [Gwe19]. Turning towards the edge, the GreenWaves Technologies GAP8 is an ultra-low-power processor that integrates eight tiny cores with a convolutional hardware accelerator and is designed for intelligence on the edge [Whe18]. Finally, the Microsoft HoloLens is a mixed-reality head-mounted display with a custom-made Holographic Processing Unit that integrates 28 custom DSPs that process and compute on sensor data. These examples indicate that SoCs across a range of computing platforms are increasingly incorporating spatial heterogeneity.

Temporal variation can be observed in Figure 1.2(b), which shows a simplified block diagram for an eight-core processor similar to the Samsung Exynos-5422 with four big cores and four little cores. The figure also shows an activity profile of the system running a convex hull application kernel (collected in cycle-level simulation, see methodology in Chapter 3). Each row corresponds to either a little core or a big core. Green corresponds to a task executing on that core, and light gray indicates that core is waiting for work. The activity varies across all eight cores at the microsecond timescale, indicating that there is significant activity imbalance at runtime for this specific workload and dataset. Although this is just one example, similar occurrences of temporal variation also

likely manifest in the Samsung Exynos, the Apple A12 Bionic chip, the Microsoft HoloLens, and in any other device that integrates and manages many components together.

Reconfigurable architectures can potentially create either form of on-chip asymmetry at configuration time. Figure 1.2(c) shows the block diagram of a coarse-grain reconfigurable array (CGRA), a class of spatial architectures that has received significant attention in recent years due to its properties in reducing data movement to and from main memory (also refer back to Figure 1.1). CGRAs are programmed using dedicated spatial compilers that convert application workloads into dataflow graphs (DFGs) which are then mapped to the CGRA fabric. The fabric is configured to interconnect different tiles according to the DFG mapping before executing the kernel with input data. Figure 1.2(c) shows the DFG for a small workload with only four compute operators. When mapped to the 25-tile CGRA, on-chip asymmetry arises because not every tile and memory bank will be utilized on every cycle, and different tiles execute different functions (e.g., multiplication, addition, memory access, routing).

1.2 The Need for Fine-Grain Power Control

Fundamentally, both forms of on-chip asymmetry reflect a potential imbalance in utilization across the chip or over time. Two well-known power-control mechanisms can actively exploit this utilization imbalance with different tradeoffs, assuming that other power-management techniques including clock gating and data gating have already been applied. *Power-gating* provides a PMOS header switch that connects or disconnects the logic from the power grid. This approach eliminates both static power (i.e., leakage) and dynamic power (i.e., switching) but also invalidates volatile state (e.g., SRAM banks) which may require significant energy to restore. *Dynamic voltage and frequency scaling* (DVFS) lowers the voltage of the power grid as well as the frequency of the circuit, quadratically reducing dynamic energy while retaining volatile state. We can visualize the *control granularity* for either of these power-control mechanisms along a spectrum in both the space and time dimensions. Figure 1.3 illustrates a rough sketch of a two-dimensional space with spatial control granularity on the y-axis and temporal control granularity on the x-axis.

Finer spatial control granularities imply that power-control regions are smaller (i.e., big cores are larger than little cores, which are larger than functional units). Referring back to the Samsung Exynos-5422 Octa Mobile Processor in Figure 1.2(a), it is clear that the annotated blocks should

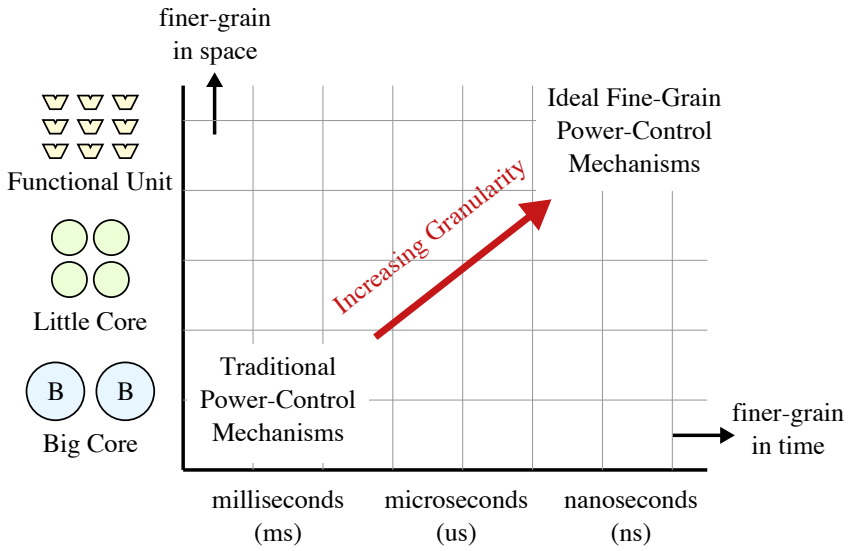


Figure 1.3: Space-Time Granularities for Power-Control Mechanisms – As on-chip asymmetry becomes finer grain in both space and time, a corresponding exploration of fine-grain power-control mechanisms is necessary to balance utilization in both dimensions.

not all be fully powered on all the time (e.g., camera). This observation is true within blocks as well. For example, a sequential application would not make use of more than one core inside the A15 cluster. We could potentially subdivide the A15 cluster into four finer-grain regions with one per core. However, doing the same for the A7 cluster might require a more flexible power-control mechanism designed for regions many times smaller than before. In Figure 1.3, this trend would be represented by moving from the lower row (i.e., "big core" spatial control granularity) up toward the middle row (i.e., "little core" spatial control granularity).

Finer temporal control granularities indicate the ability to quickly adapt to changes (e.g., at the millisecond, microsecond, or nanosecond timescales). Referring back to the octacore activity profile for the convex hull application kernel in Figure 1.2(b), the activity varies at the microsecond timescale. Although we could power each core throughout the entire kernel, it would be more efficient to manipulate each core's power-control region to take advantage of microsecond-scale periods of low activity. In Figure 1.3, this capability corresponds to the center region with microsecond-scale, per-core (little and big) power control.

Placing the state of the art into perspective, the majority of past literature and industry chips have assumed DVFS-based, coarse-grain power-control in both space and time dimensions (see "Traditional Power-Control Mechanisms" in Figure 1.3), with the oldest works assuming even simpler systems with a single power-control region for the entire chip. These works primarily assume

off-chip switching regulators that operate at low switching frequencies with large high-Q passives to reduce parasitic switching losses, and they have longer control latencies due to slow switching speeds and parasitics between the on-chip load and the off-chip regulator, resulting in voltage scaling response times on the order of tens to hundreds of microseconds [PSCP10, Mif01, BPSB00]. Academia has recently displayed a clear trend toward finer-grain power-control techniques in both architecture and circuit research communities [BM09, KGWB08, TCM+09, LCVR03, DWB+10, CC06, RWB09, DM06, LK09, AGS05, RES+13, LK14, MPT+12, SAD+02, JTH+13, MYN+11, RL14, WM15, LPD+14, KCZ+17]. And industry has followed suit with a limited form of per-core voltage scaling available in Intel Haswell and Broadwell [Kan13, MBH+14] as well as in AMD Ryzen Mobile [Kan17]. This trend corresponds to an initial push towards the center region in Figure 1.3. As on-chip asymmetry continues to increase, exploration of finer-grain power-control techniques will continue to grow in importance.

1.3 Breaking Through Walls of Abstraction for Better Control

Fine-grain power-control mechanisms are important. However, of perhaps even greater importance is the application-level information that determines *when* and *where* these mechanisms should be applied. Today, there is an exciting undercurrent throughout the entire field of computer engineering that is transforming how we address these challenges.

The majority of past work prior to this new era has assumed that available information is limited by traditional walls of abstraction in the computing stack. For example, many prior architecture works that explore fine-grain DVFS try to predict when to actuate voltage and frequency changes based on hardware performance counters [WJMC04, MCGG06, HM07, SAD+02, WJMC05]. The authors in [MCGG06] use performance counters and per-access energy registers to estimate energy within each domain (at a sampling interval) before using a state machine to profile several DVFS modes and decide on the optimal voltages and frequencies. These profiling schemes are delicate, take time to execute, and do not scale well, eating into the available benefit afforded by finer-grain power-control techniques.

Today, the traditional walls of abstractions in the computing stack have broken down, enabling far more information to pass between software, architecture, and VLSI layers than previously possible. For example, software-architecture co-design has become commonplace to explore the

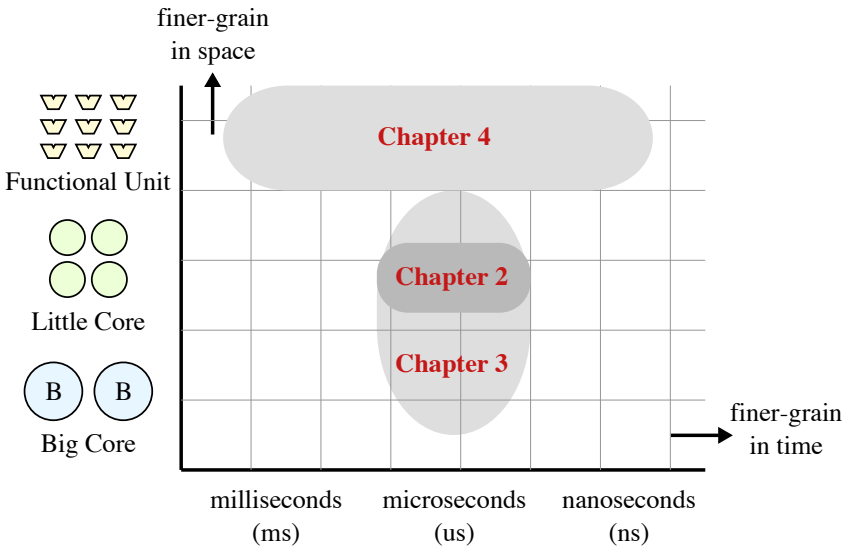
design space of machine learning accelerators and spatial architectures [CKES17, KKV+18, ea19, GHS11, PPA+13]. Meanwhile, an architecture-VLSI co-design approach has enabled research on reliable computing for deep space and high-radiation environments using architectural modular redundancy [KMRM19]. As the walls of abstractions break down further, it is becoming more feasible to explore research not only pair-wise but also across software, architecture, and VLSI all at once, exposing useful information across the stack. In this new era of computer architecture, there is great potential to improve on the state of the art by carefully orchestrating this previously inaccessible information.

1.4 Thesis Overview

This thesis explores novel fine-grain voltage and frequency scaling techniques with the goal of exploiting on-chip asymmetry in both the space and time dimensions to improve both performance and energy efficiency. Differentiating from previous work, I focus on a software, architecture, and VLSI co-design approach to provide control for these techniques using previously inaccessible information newly exposed across layers of abstraction. I further differentiate by exploring the potential for specializing these techniques for productive task-based parallel runtimes as well as for coarse-grain reconfigurable arrays. Figure 1.4 summarizes the focus of each chapter.

Chapter 2 explores realistically enabling fine-grain voltage and frequency scaling for homogeneous systems of little cores at microsecond timescales by leveraging recent work on fully integrated voltage regulation. On-chip voltage regulators have notable key challenges, including lower on-chip conversion efficiencies and on-die area overheads. In this chapter, I demonstrate how careful software, architecture, and circuit co-design can not only mitigate circuit-level challenges for integrated voltage regulation but also resolve architecture-level bottlenecks in homogeneous multicores, resulting in more energy-efficient and performant systems. This work was published at the Int'l Symp. on Microarchitecture (MICRO) in 2014 [GTB+14]. I was an equally contributing first author on this work as the core architect, and I worked with Waclaw Godycki who had expertise on mixed-signal design.

Chapter 3 broadens the scope to heterogeneous multicore systems while also specializing the techniques for the domain of productive task-based parallel runtimes. This work focuses on fine-grain voltage and frequency scaling for both big and little cores at microsecond timescales. In



	Software	Architecture	VLSI
Chapter 2	SPMD MIMD	Little Multicore	Dynamic Capitance Sharing
Chapter 3	Task-Based Parallel Runtime	Big / Little Multicore	
Chapter 4	Spatial Compiler Dataflow Graph	CGRA	Rational Clocking with Elasticity

Figure 1.4: Thesis Overview – This thesis explores fine-grain voltage and frequency scaling in both space and time for three different contexts involving software, architecture, and VLSI co-design. Chapter 2: Reconfigurable Power Distribution Networks; Chapter 3: Asymmetry-Aware Work-Stealing Runtimes; Chapter 4: Ultra-Elastic Coarse-Grain Reconfigurable Arrays.

addition, while the previous chapter relied on simple static work distribution and sped up lagging cores, in this chapter we focus on productive task-based parallel runtimes that use sophisticated work-stealing algorithms for better load balancing. I argue that work-stealing algorithms are a natural fit for managing on-chip asymmetry at the software level. This chapter explores how these software runtimes can be made aware of underyling asymmetry in the architecture and VLSI layers to create more efficient schedules and to dynamically tune processing elements. I propose an asymmetry-aware work-stealing runtime based on three key software/hardware techniques: work-pacing, work-sprinting, and work-mugging. Work-pacing and work-sprinting are novel techniques that greatly improve both performance and energy efficiency of the runtime by *sprinting* (i.e., increasing voltage and frequency) threads that are executing tasks and *resting* (i.e., decreasing volt-

age and frequency) threads that are waiting for work in the steal loop. I also propose infrastructure to enable work-mugging, which moves tasks from slower little cores to faster big cores in the low-parallel region. This work was published at the Int'l Symp. on Computer Architecture (ISCA) in 2016 [TWB16]. I was the lead author for this work.

Chapter 4 narrows the focus to coarse-grain reconfigurable arrays (CGRAs). Here, I explore fine-grain voltage and frequency scaling for each tile and memory subbank at reconfiguration timescales, which may vary from hundreds of nanoseconds to milliseconds. CGRAs have become increasingly popular as specialized compute fabrics due to their potential for high performance while reducing control and data-movement energy. However, widespread industry adoption has been limited due to the complexity of compiler scheduling algorithms that must optimize across many constraints. Recent work on *elastic CGRAs* promises to significantly mitigate compiler-level challenges with hardware-managed flow control. In this chapter, I propose *ultra-elastic CGRAs* which capitalize on new opportunities in elastic CGRAs, enabling support for configurable per-tile fine-grain power control and significantly improved dataflow efficiency. I am the lead author of this work, and it is currently unpublished.

Chapter 5 describes my work on four silicon prototypes to support various aspects of my thesis. I tested two of these prototypes in our digital ASIC/FPGA prototyping lab, with the other two chips tested by collaborators. Together with my collaborators, the results from my silicon prototyping experience were published at top-tier chip and design-automation conferences including Hot Chips, VLSI, IEEE TCAS I and IEEE MICRO. The chips include a mixed-signal test chip and three digital ASIC test chips. Of these chips, I was the project lead for two chips (BRGTC1 in IBM 130 nm [TWS+16] and BRGTC2 in TSMC 28 nm [TJAH+18]) and Cornell University student lead for the DARPA-funded, multi-university project on developing the Celerity SoC in TSMC 16 nm [AAHA+17, DXT+18, RZAH+19]. For the DCS test chip, I helped with full-custom design and also worked on the post-silicon testing process [BTG+17].

The primary contributions of this thesis are:

- A novel approach for fine-grain voltage and frequency scaling for homogeneous systems of little cores at microsecond timescales based on switched-capacitor-based integrated voltage regulators using a novel **dynamic capacitance sharing** technique.

- A novel approach for fine-grain power control for heterogeneous multicore systems at microsecond timescales specialized for **task-based parallel runtimes** using a set of three techniques based on balancing marginal utility.
- A novel proposal for ultra-elastic CGRAs which capitalize on new opportunities in elastic CGRAs, enabling support for configurable per-tile fine-grain power control and significantly improved dataflow efficiency.
- A deep design-space exploration of these ideas using a vertically integrated research methodology that in many cases extends from cycle-level modeling down to silicon prototyping.

1.5 Collaboration and Funding

This thesis would not have been possible without support from all of the members of the Batten Research Group and others. My advisor Christopher Batten was a key source of inspiration and guidance, helping to transform ideas and guide them in interesting directions. The number of times I have had ideas that became many times more interesting through his intervention cannot be understated.

The work on reconfigurable power distribution networks presented in Chapter 2 was an interdisciplinary project with an architecture half and a circuits half. I was the architecture lead, and the circuits portion was led by Waclaw Godycki and Professor Alyssa Apsel. Waclaw designed the SPICE-level DC-DC converters used in the project and worked with me at the architecture-circuit interface to accurately model voltage transients in my architectural cycle-level models based on gem5. Waclaw also later led the tapeout for the DCS test chip that is briefly introduced in Section 5.1. When the chip came back, Ivan Bukreyev led the post-silicon testing as well as the chip characterization paper [BTG+17]. Finally, I would also like to thank Derek Lockhart and Yunsup Lee who established the initial ASIC CAD toolflow that I later leveraged and adapted to build the 65 nm energy model used in this work.

I led the asymmetry-aware work-stealing runtime work presented in Chapter 3, but Moyang Wang was integral to the success of the project. He designed the work-stealing runtime from scratch (inspired by Intel TBB). We then worked together on instrumenting the runtime to enable the new techniques. Moyang added the exception handler for the work-mugging thread swap to the

work-stealing runtime. Moyang also helped port a wide variety of benchmarks to our architecture including PBBS, Cilk, and PARSEC application kernels.

I led the ultra-elastic CGRA work presented in Chapter 4, but a strong team of students worked with me throughout the project. Peitian Pan and Yanghui Ou led the RTL for the CGRA, helped implement special circuitry for ratiochronous clock-domain crossings, characterized energy for the tiles and CGRAs, and measured the throughput. Cheng Tan implemented the LLVM-based compiler, transformed C benchmarks into DFGs, and mapped them onto the RTL and my analytical model. This project only took shape due to the tremendous contributions from these three.

I was the lead for the BRGTC1 project, but the chip would have been impossible without *immense* support from Moyang Wang, Bharath Sudheendra, Nagaraj Murali, Suren Jayasuriya, Shreesha Srinath, Taylor Pritchard, Robin Ying, Eric Tang, Rohan Agarwal, and Cameron Haire. Moyang was the verification lead for our PyMTL core. Bharath and Nagaraj were integral in bringing up much of the physical backend flow in Synopsys ICC, since we started the project with little real-world physical backend expertise ourselves. On that note, Suren provided invaluable advice as I worked through Calibre DRC and LVS and many other physical design topics. Shreesha wrote the bubble-sorting accelerator attached to our core using commercial high-level synthesis tools. Taylor designed the PyMTL RTL for the host interface surrounding the eight-bit asynchronous channel. Robin Ying led the design of the full-custom LVDS receiver before handing it off to me to integrate with our digital flow. Eric and Rohan designed the breakout board that pulled the pins of our (packaged) die out to headers for preliminary post-silicon validation. Finally, Cameron brought up the post-silicon validation flow at our bench using the logic analyzer, pattern generator, and DC power analyzer. I also thank Ivan Bukreyev, who helped me connect the core power trunks to the IO ring in Cadence Virtuoso just a few hours before tapeout. I would never have been able to do this in time without his willingness to help friends in a moment's notice.

I was the lead for the BRGTC2 project, but the chip would not have been possible without these six people pitching in over the final month: Shunning Jiang, Khalid Al-Hawaj, Ivan Bukreyev, Berkin Ilbeyi, Tuan Ta, Lin Cheng. Shunning Jiang was a key contributor to the RTL design, singlehandedly bringing up many components including the L0 buffer, the core, the multiply/divide unit, and a great deal of PyMTL verification infrastructure. Khalid ported the shared cache. Ivan ported the synthesizable PLL from 16 nm to 28 nm (this was initially designed by Julian Puscar and Ian Galton for the Celerity project). Berkin added RTL for a custom Bloom filter accelerator.

Tuan led the cycle-level gem5 modeling to inform how the smart-sharing architecture should be assembled. Lin made sure that the work-stealing runtime ran on our chip in RTL simulation. I would also like to thank Shreesha Srinath for his work on smart-sharing architectures in general, and I would like to thank Moyang Wang for designing the work-stealing runtime. Finally, everyone deserves thanks for their creativity in naming our incremental design milestones after Pokémon.

The Celerity project was a big multi-university effort across four universities spread across different geographical locations. I was the Cornell student lead, but in the whole project there were twenty students and faculty involved. From the Michigan team, Tutu Ajayi, Austin Rovinski, and Aporva Amarnath worked together with me on top-level SoC integration. We also worked together to bring up the 16 nm physical backend for the entire SoC (I visited Michigan for two months during the project to help out with this). Ron G. Dreslinski was an inspirational advisor. From the UCSD team, Julian Puscar and Ian Galton designed the synthesizable PLL, and Loai Salem and Patrick P. Mercier designed the digital LDOs powering the chip. Both teams handed off GDS to us along with other files for integration with the digital flow. Rajesh K. Gupta was instrumental in bringing the whole multi-university team together. From the Bespoke Silicon Group (BSG), a number of people developed the RTL for the manycore, the chip interconnection, and the Rocket infrastructure: Scott Davidson, Shaolin Xie, Anuj Rao, Ningxiao Sun, Luis Vega, Bandhav Veluri, Chun Zhao, and Michael B. Taylor. BSG also provided many physical backend scripts from older chips developed by previous generations of students for our reference. Finally, Dustin Richmond and Paul Gao led the post-silicon bringup with BSG infrastructure. At Cornell, Ritchie Zhao, Steve Dai, Gai Liu, and Zhiru Zhang led the development of the binarized neural network. Khalid Al-Hawaj and Steve Dai ported the BNN from Vivado HLS with an FPGA backend to Cadence Stratus HLS targeting an ASIC backend.

In terms of funding, this thesis was supported in part by NSF CAREER Award #1149464, a Spork Fellowship, AFOSR YIP Award #FA9550-15-1-0194, the MOSIS Educational Program, NSF XPS Award #1337240, NSF SHF Award #1527065, DARPA POSH Award #FA8650-18-2-7852, DARPA CRAFT Award HR0011-16-C-0037, NSF CRI Award #1059333, NSF CRI Award #1512937, NSF SaTC Award #1563767, NSF SaTC #1565446, EDA tool donations from Synopsys, Cadence, and Mentor, physical IP donations from ARM, EDA tool and FPGA development board donations from Xilinx, and donations from Intel Corporation and Synopsys, Inc. NSF CAREER Award #1149464 NSF XPS Award #1337240, NSF SHF Award #1527065, NSF CRI Award

#1059333, NSF CRI Award #1512937, AFOSR YIP Award #FA9550-15-1-0194, DARPA YFA Award #N66001-12-1-4239, DARPA CRAFT Award #HR0011-16-C-0037, DARPA SDH Award #FA8650-18-2-7863, a Spork Fellowship, the MOSIS Educational Program, and equipment, tool, and/or physical IP donations from Intel, NVIDIA, Xilinx, Synopsys, and ARM.

CHAPTER 2
RECONFIGURABLE POWER DISTRIBUTION NETWORKS

This chapter explores realistically enabling fine-grain voltage and frequency scaling for homogeneous systems of little cores at microsecond timescales by leveraging recent work on fully integrated voltage regulation. On-chip voltage regulators have notable key challenges, including lower on-chip conversion efficiencies and on-die area overheads. In this chapter, I demonstrate how careful software, architecture, and circuit co-design can not only mitigate circuit-level challenges for integrated voltage regulation but also resolve architecture-level bottlenecks in homogeneous multicores, resulting in more energy-efficient and performant systems.

2.1 Introduction

Monolithic integration using a standard CMOS process provides a tremendous cost incentive for including more and more functionality on a single die. This system-on-chip (SoC) integration enables both low-power embedded platforms and high-performance processors to include a diverse array of components such as processing engines, accelerators, embedded flash memories, and external peripheral interfaces. Almost every computing system requires closed-loop voltage regulators that, at first glance, seem like another likely target for monolithic integration. These regulators convert the noisy voltage levels available from the system's environment into the multiple fixed or adjustable voltage levels required by the system, and they are usually based on efficient switch-mode circuits. These regulators have traditionally been implemented off-chip for two key reasons: (1) limited availability of high-speed switching with suitable parasitic losses; and (2) limited availability of integrated energy-storage elements with suitable energy densities. The economic pressure towards monolithic integration has simply not outweighed the potential reduction in efficiency.

Recent technology trends suggest that we are entering a new era where it is now becoming feasible to reduce system cost by integrating switching regulators on-chip. High-speed switching efficiencies have increased with technology scaling, reducing the need for very high-density inductors and capacitors. This trend is evident in industry, especially in Intel's recent Haswell microprocessors which use in-package inductors with on-chip regulators to provide fast-changing

supply voltages for different chip modules [Kan13, MBH+14]. At the same time, materials improvements such as integrated in-package magnetic materials (e.g., Ni-Fe [SPW+11]) and new integrated on-chip capacitor organizations (e.g., deep-trench capacitors [AKK+13, CMJ+10]) have improved the density of the energy storage elements that are available. The future of on-chip voltage regulation offers interesting opportunities and significant challenges, and this has sparked interest from the circuit research community [HSH+05, LSA11, LCSA13, WS11, SPW+11, HKB+05, KH11, ASL+09, KBW12, GSA14] and to a lesser degree in the architecture research community [YLH+12, KGWB08, ASSK11, CCK07, ZJKS11, ZYFL10].

In addition to reduced system cost, one of the key benefits of on-chip regulation is the potential for fine-grain voltage scaling (FGVS) in level (i.e., many different voltage levels), space (i.e., per-core regulation), and time (i.e., fast transition times between levels). Dynamic voltage and frequency scaling (DVFS) is perhaps one of the most well-studied techniques for adaptively balancing performance and energy efficiency. DVFS has been leveraged to improve energy efficiency at similar performance [BM09, GC97, KGWB08, TCM+09, LCVR03], operate at an energy-minimal or energy-optimized point [DWB+10, CC06], improve performance at similar peak power [RWB09, DM06, LK09, AGS05, PDS+13, RES+13, LK14], and mitigate process variation [MPT+12]. Most of these studies have assumed off-chip voltage regulation best used for coarse-grain voltage scaling. Traditional off-chip switching regulators operate at low switching frequencies due to the availability of large high-Q passives and the desire to reduce parasitic switching losses. They also have longer control latencies due to slow switching speeds and parasitics between the on-chip load and the off-chip regulator, resulting in voltage scaling response times on the order of tens to hundreds of microseconds [PSCP10, Mif01, BPSB00]. On-chip switching regulators can leverage faster control loops and are tightly integrated with the on-chip load enabling voltage scaling response times on the order of hundreds of nanoseconds. Traditional off-chip switching regulators are expensive, bulky, and obviously require dedicated power pins and on-chip power distribution networks limiting the number of independent on-chip power domains; on-chip switching regulators can be located close to each core enabling per-core voltage scaling.

In this chapter, we use an architecture and circuit co-design approach to explore the potential system-level benefit of FGVS enabled by integrated voltage regulation and techniques to mitigate the overhead of this regulation. Section 2.2 describes our target system: a sub-Watt eight-core

embedded processor design implemented in a TSMC 65 nm process using a commercial standard-cell-based ASIC toolflow.

Section 2.3 uses architectural-level modeling to explore a new FGVS controller called the *fine-grain synchronization controller* (FG-SYNC+) that exploits the specific opportunities of fine-grain scaling in level, space, and time. Inspired by Miller et al.'s recent work on Booster SYNC [MPT+12], FG-SYNC+ uses a thread library instrumented with hint instructions to inform the hardware about which cores are doing useful work vs. useless work (e.g., waiting for a task or waiting at a barrier). FG-SYNC+ improves upon this prior work in several ways by leveraging the ability of on-chip voltage regulation to provide multiple voltage levels and using additional hints to inform the hardware of how each core is progressing through its assigned work. Booster SYNC improves performance at the expense of increased average power (i.e., the "boost budget"). Other DVFS controllers usually improve energy efficiency at similar performance or improve performance under a conservative peak power limit fixed at design time. FG-SYNC+ has a more ambitious goal of improving performance and energy efficiency while maintaining similar average power. To do this, FG-SYNC+ exploits the fine-grain activity imbalance often found in multithreaded applications. For example, Figure 2.1 illustrates the activity of an eight-core system on three multithreaded applications and highlights potential opportunities for increasing the voltage of active cores and decreasing the voltage of waiting cores. Note that exploiting this fine-grain imbalance is simply out-of-reach for traditional off-chip regulators.

Section 2.4 uses circuit-level modeling to explore the practical design of an integrated voltage regulator suitable for use by FG-SYNC+. Much of the prior work in this area explores inductor-based regulators, but we argue that carefully designed on-chip switched-capacitor (SC) regulators can potentially mitigate many of the challenges involved in on-chip regulation. We explore designs with a single-fixed voltage regulator (SFVR) and multiple adjustable voltage regulators (MAVR). Unfortunately, per-core voltage regulation can incur significant area overhead and longer responses times than one might expect. This is mostly because each MAVR regulator must be designed to efficiently support the peak power that can be consumed by the fastest operating mode. Our study of FG-SYNC+ enables us to make a key observation: MAVR is significantly over-designed, since all cores can never be in the fastest operating mode. Based on this observation, we propose a new approach called *reconfigurable power distribution networks* (RPDNs). RPDNs include many small "unit cells" shared among a subset of the cores in the design. Each unit cell contains the fly-

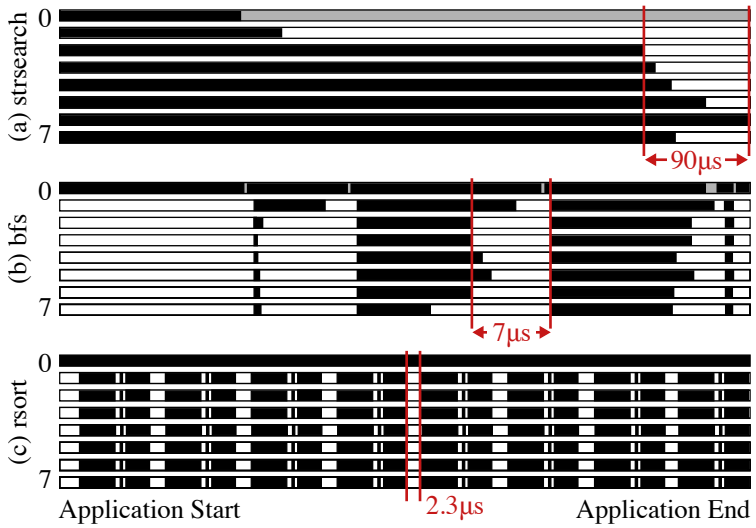


Figure 2.1: Activity Profile for Select Applications on Eight Cores – Variation in activity across cores produces opportunities for FGVS. Black = active; gray = waiting for join; white = waiting for work.

back capacitance and regulator switches required for a SC regulator; the unit cells can be flexibly reconfigured through a switch fabric and combined with per-core control circuitry to effectively create multiple SC regulators "on-demand". This reconfiguration reduces area overhead by avoiding the over-provisioning inherent in MAVR, improves response time when changing the target output voltage by leveraging the adjustable flyback capacitance in addition to the adjustable regulation frequency, and can potentially improve efficiency in leaky processes by reducing flyback capacitance at low current.

In Section 2.5, we describe our detailed evaluation methodology based on a combination of circuit-, gate-, register-transfer-, and architectural-level modeling; in Section 2.6, we use this methodology to explore the system-level implication of combining FG-SYNC+ with RPDN. These results suggest a promising new approach that can facilitate fine-grain voltage scaling with low-overhead in future multicore processors. In Section 2.7, we discuss the impact of di/dt noise on RPDN and the implications of scaling RPDN to larger networks, higher power densities, and different technologies.

The contributions of this work are: (1) we propose a new controller called FG-SYNC+ that improves performance and energy efficiency at similar average power; (2) we propose a novel approach to on-chip regulator design based on the idea of reconfigurable power distribution networks; and (3) we use a vertically integrated research methodology to explore the FGVS design space.

2.2 Target System

Although much of our analysis is applicable to larger high-performance systems, we choose to focus on the smaller low-power systems that will likely be the first to integrate significant on-chip voltage regulation. Our target system is an embedded processor composed of: eight in-order, single-issue, five-stage, RISC cores; private, coherent 16 KB instruction and data L1 caches; and a shared 512 KB unified L2 cache.

We implemented the core and L1 memory system for this design in RTL and used a commercial standard-cell-based CAD toolflow targeting a TSMC 65 nm process to generate layout for one core. Section 2.5 describes our research methodology in greater detail. We assume the external supply voltage is 2.2 V and that FGVS should provide up to four voltage levels: 1.0 V for the nominal supply; 0.7 V for a slow, low-power execution mode (*resting mode*); 1.15 V for a fast, high-power execution mode (*sprinting mode*); and 1.33 V for an even faster execution mode (*super-sprinting mode*), all within the acceptable process operating range. Analysis of the placed-and-routed design indicates each core is approximately 0.75 mm^2 and can run at 333 MHz at 1 V. We predict that more aggressive RTL and circuit design could increase this clock frequency by 2× or more.

First-order estimates suggest the full eight-core system would be approximately 6 mm^2. When running a reasonable workload, each core/L1 consumes approximately 20 mW, and when waiting for work or a synchronization primitive, each core/L1 consumes approximately 3 mW. This implies that the power for all eight cores and L1 memory system (excluding the L2 cache) can range from 100–200 mW when doing useful work and that the peak power density of the cores and L1 memory system is approximately 25 mW/mm^2. For all designs we assume (potentially multiple) on-chip phase-locked-loops (PLLs) to enable fast frequency adjustment based on recent low power designs [DMS+13, FDD+06]. We will use this target system to help drive the design of FG-SYNC+ and RPDN.

2.3 FGVS Architecture Design: FG-SYNC+

In this section, we explore a new *fine-grain synchronization controller* (FG-SYNC+) using architecture-level modeling. Section 2.5 includes more details about our evaluation methodology. After introducing the basic FG-SYNC+ controller, we use three sensitivity studies to understand

the implication of varying: (1) the number of voltage levels, (2) the number of voltage domains, (3) and voltage-settling response times. Insights from this section will help motivate our design-space exploration of on-chip voltage regulation in Section 2.4.

2.3.1 Basic FG-SYNC+ Controller

The goal of FG-SYNC+ is to improve performance at the same average power. FG-SYNC+ rests cores that are not doing useful work, creating power slack to sprint cores that are doing useful work. Inspired by previous work on Booster SYNC [MPT+12], we instrument synchronization primitives in the threading library with hint instructions to inform the hardware which threads are doing useful work. This elegant approach avoids the need for complex prediction heuristics by exploiting application-level information to efficiently sprint the most critical cores. FG-SYNC+ extends Booster SYNC in two ways: (1) by carefully using the multiple voltage levels available with on-chip regulation and (2) by including hints indicating the progress of each thread.

The hint instructions toggle activity bits in each core. FG-SYNC+ reads these bits every sampling period and uses a lookup table to map activity patterns to DVFS modes. In the example table in Figure 2.2, if all cores are doing useful work, then FG-SYNC+ runs the entire system at nominal voltage and frequency (first row). As more cores are waiting, FG-SYNC+ rests the waiting cores and uses the resulting power slack to sprint or super-sprint active cores. We design lookup tables offline using our RTL-based energy model to ensure that the power of each configuration will remain below the average power of all cores running at nominal voltage (i.e., with no DVFS). Booster SYNC only provides two voltage levels since it relies on fast switching between two off-chip voltage regulators, thus Booster SYNC improves performance by increasing power consumption. FG-SYNC+'s use of multiple levels enables balancing sprinting and resting cores to improve performance at the same average power.

FG-SYNC+ includes additional "work left" hint instructions embedded in the thread library's `parallel_for` function to inform the hardware how many iterations the core has left to process. This gives FG-SYNC+ insight into the relative progress of each core in a multithreaded application. Without these additional hints, FG-SYNC+ can determine which cores are active but not which of these cores are most critical. The "work left" hint instructions enable FG-SYNC+ to sprint those cores that have the most work to do, potentially reducing the overall execution time.

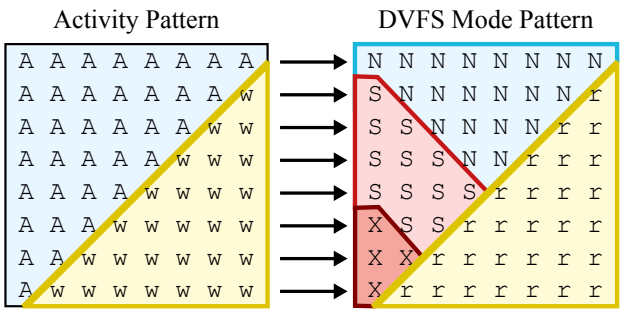


Figure 2.2: Lookup Table Mapping Activity Patterns to DVFS Modes – FG-SYNC+ uses activity information to rest cores that are waiting, creating power slack to sprint cores that are doing useful work. A = core doing useful work; w = core waiting; r = core resting at 0.7 V; N = core in nominal mode at 1.0 V; S = core sprinting at 1.15 V; X = core super-sprinting at 1.33 V.

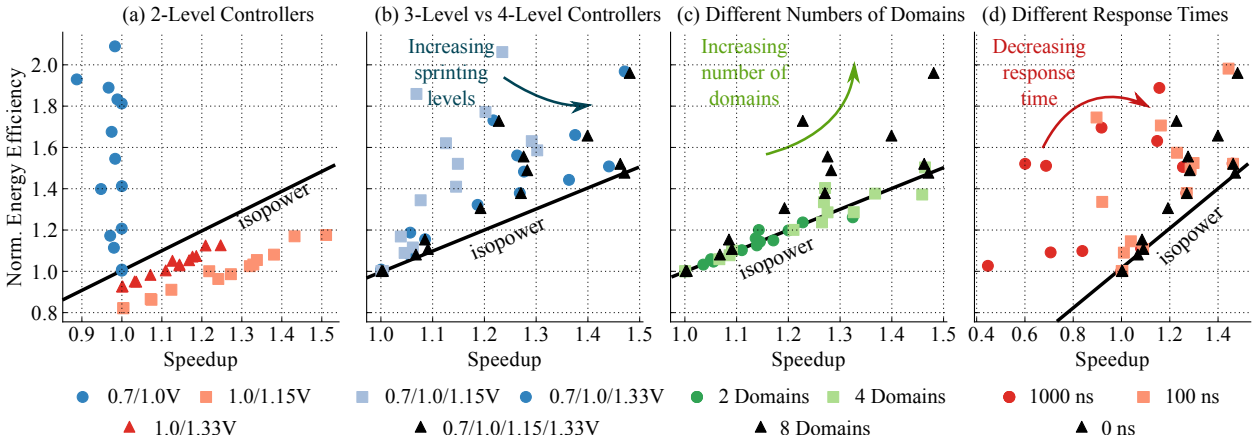


Figure 2.3: FGVS Exploration in Level, Space, and Time – Normalized energy efficiency and speedup over a baseline system with no DVFS. Points are applications simulated with the given controller. (a) Comparison between 2-level controllers; nominal paired with different resting and sprinting levels. (b) Comparison between different 3-level and 4-level controllers. (c) Using a 4-level controller, comparison between different numbers of voltage domains. (d) Using a 4-level controller with 8 domains, sweep response time per 0.15 V step. The black triangles in plots (b), (c), and (d) represent the same controller with very fine-grain voltage scaling in all three dimensions (i.e., 4-level, 8-domain, 0 ns response time).

2.3.2 FG-SYNC+ with Fine-Grain Scaling in Level

We begin our study assuming a system with very fine-grain voltage scaling in space and time: eight voltage domains (i.e., per-core voltage regulation) and instantaneous voltage-settling response time. Then we scale the number of available voltage levels and study the impact on performance and energy efficiency. Note that with one voltage level (1.0 V), FG-SYNC+ is identical to the baseline system with no DVFS since it can neither rest nor sprint.

Supporting two voltage levels enables adding either a rest or a sprint level. Figure 2.3(a) compares different 2-level FG-SYNC+ controllers running a diverse set of multithreaded applications on our target system. Each controller pairs the nominal level with either the resting, sprinting, or

super-sprinting level. The upper-right quadrant in these normalized energy efficiency vs. performance plots has improved performance and energy efficiency compared to the baseline. Points above the isopower line use less power than the baseline, and points below it use more power than the baseline. Figure 2.3(a) shows that choosing a rest level (0.7 V) improves energy efficiency with no speedup. Some applications even slow down because cores that are waiting for work in a spin-loop respond more slowly to newly-available work. Choosing a sprinting level (i.e., 1.15 V or 1.33 V, similar in spirit to Booster SYNC) increases performance but also significantly increases average power. With only two levels, we are forced to choose between performance or energy efficiency.

Supporting three voltage levels enables adding both a rest and a sprint level. Figure 2.3(b) compares 3-level and 4-level FG-SYNC+ controllers. FG-SYNC+ can now improve both performance and energy efficiency by resting waiting cores and sprinting active cores. Choosing either sprint (1.15 V) or super-sprint (1.33 V) as our third level improves both performance and energy efficiency, but choosing super-sprint offers greater performance while still staying under the baseline power. Supporting four voltage levels enables adding rest, sprint, and super-sprint levels. FG-SYNC+ gains the ability to super-sprint 1–2 cores or to more evenly sprint 3–7 cores (see Figure 2.2). In short, FG-SYNC+ can use the fourth level to better utilize power slack, further increasing performance and more closely tracking the isopower line.

These results motivate supporting at least three levels to benefit from FGVS. For the remainder of this work, we will assume supporting four levels. Note that systems like Booster that use off-chip regulators will find it costly to support more than two levels, since this would require either more resources for additional power pins and on-chip power networks or poorer quality regulation.

2.3.3 FG-SYNC+ with Fine-Grain Scaling in Space

We now explore how FG-SYNC+ performs with fewer than eight voltage domains. With two domains, cores 0–3 and 4–7 are grouped into quads; with four domains, neighboring cores are grouped into pairs. All cores in a group must scale their voltages together. Therefore, a core waiting for work cannot rest unless all other cores in the same group are also waiting. If a core sprints, the whole group must sprint as well. In contrast, cores in the 8-domain system can independently scale voltage and frequency. Note that with one voltage domain, FG-SYNC+ cannot sprint without

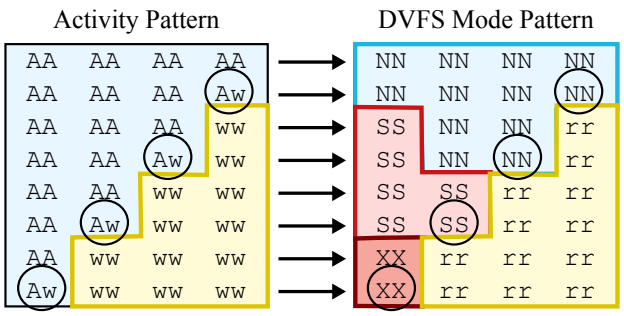


Figure 2.4: Lookup Table Mapping Activity Patterns to DVFS Modes (Four Domains) – Compare and contrast with eight domains in Figure 2.2. In the circled pairs, FG-SYNC+ with four domains must choose between either running the waiting core at a non-resting level (energy-inefficient) or running the busy core at rest (lower performance).

significantly increasing the average power over the baseline; therefore it cannot offer a performance benefit at the same average power.

Figure 2.3(c) compares FG-SYNC+ with 2–8 voltage domains. Each controller has four voltage levels and instantaneous voltage-settling response time. With two domains, FG-SYNC+ can improve energy efficiency by resting one quad given that all cores in the quad are waiting for work, and active cores in the other quad can be sprinted for modest performance gains. We cannot super-sprint an active quad without exceeding the average power of the baseline. FG-SYNC+ with four domains significantly improves: (1) energy efficiency by enabling more cores to rest in pairs and (2) performance by enabling a single pair to super-sprint when all other pairs are resting.

Having eight domains (i.e., per-core voltage regulation) enables FG-SYNC+ to independently optimize each core's voltage and frequency for its activity. Figure 2.3(c) shows that having eight domains significantly improves energy efficiency over four domains. Compare the lookup table for four domains in Figure 2.4 with the lookup table for eight domains in Figure 2.2. Notice that for the circled pairs in the 4-domain table, FG-SYNC+ must choose between running a waiting core at a non-resting level or running an active core at the resting level, sacrificing either performance or energy-efficiency. In this study, if a domain has at least one active core we choose to run the entire quad at the best voltage level for that active core. This prioritizes performance over energy efficiency for the 2- and 4-domain configuration. The 8-domain configuration does not need to make this trade-off and is able to improve both performance and energy efficiency.

Figure 2.5(a,b) illustrates the impact of coarser voltage domains on application performance for the *SPLASH-2 LU factorization* benchmark. Rows represent cores, black strips represent core activity, and colors represent DVFS modes. In Figure 2.5(a), FG-SYNC+ is heavily constrained

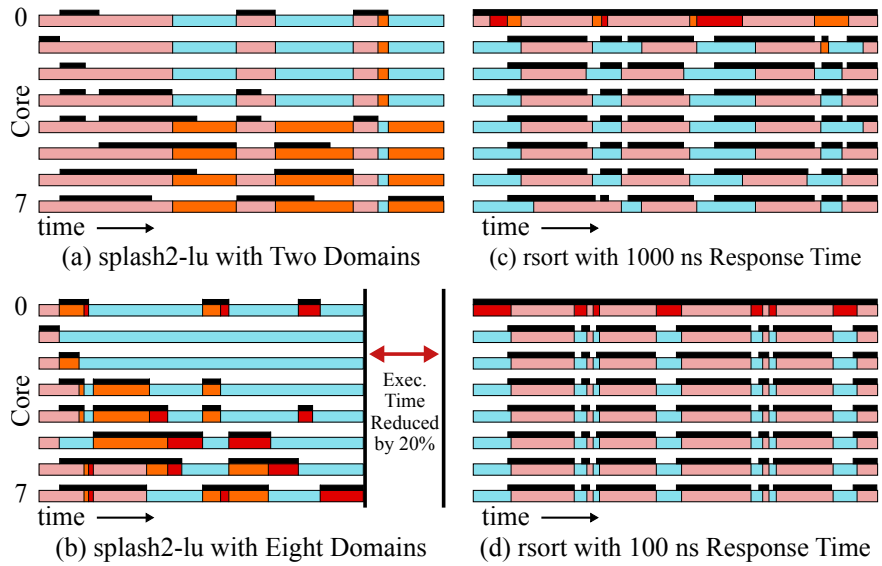


Figure 2.5: Application Activity Plots for FG-SYNC+ – Rows show controller decisions per-core (rest = blue, nominal = pink, sprint = orange, super-sprint = deep red). Horizontal black strips above cores show when that core is active. (a,b) illustrates the impact of using multiple voltage domains over the full execution of SPLASH-2 LU factorization; (c,d) illustrates the impact of faster voltage-settling response times over a small excerpt from the execution of radix sort.

and is forced to inefficiently run a quad at nominal or sprint, even though few cores in the quad are actually doing useful work. In Figure 2.5(b), per-core voltage regulation enables FG-SYNC+ to independently rest waiting cores, sprint several active cores, or even super-sprint one or two active cores, resulting in an execution time reduction of 20%.

This study motivates very fine-grain regulation in space to improve performance and energy efficiency. For the remainder of this work, we will assume per-core voltage regulation.

2.3.4 FG-SYNC+ with Fine-Grain Scaling in Time

We assume that after FG-SYNC+ makes a decision to change the voltage and frequency of a domain, it must wait until the voltage has settled before making a new decision. As in [KGWB08, CHM$^+$01], we assume that cores continue running even during voltage transitions. When scaling voltage up, the change in frequency must *lag* the change in voltage, and when scaling voltage down, frequency must *lead* the voltage. Both of these constraints ensure that the design always meets cycle time constraints. Together, this means that if FG-SYNC+ decides to scale the voltage up, the new frequency will not take effect immediately, and FG-SYNC+ cannot make a new decision until the new voltage has settled. Note that there is additional energy overhead during this transition

as the core slowly scales voltage while staying locked at the frequency of the lower DVFS mode. For example, when scaling from (0.7 V, frequency *f1*) to (1.0 V, frequency *f2*), transition energy is paid during the time the core runs at *f1* and is transitioning voltages between 0.7 V and 1.0 V.

Figure 2.3(d) illustrates the impact of these overheads on performance and energy efficiency. We use a simplistic model where we linearly increase the voltage-settling response time per 0.15 V step. First note that a relatively slow 1000 ns response time increases the likelihood that FG-SYNC+ will not be able to adjust to fine-grain activity imbalance and indeed the inability to rapidly make controller decisions leads to sharp slowdowns over the baseline. A response time of 100 ns allows FG-SYNC+ to adjust quickly to fine-grain activity imbalance in our applications; this is fast enough to closely track the results for ideal (0 ns) response time.

Energy efficiency is balanced as we scale to finer-grain response times. On one hand, slow response times actually *improve* energy efficiency because cores spend more time waiting at lower (and more energy-efficient) DVFS modes until voltage settles, while still doing useful work; on the other hand, fast response times also improve energy efficiency by enabling FG-SYNC+ to quickly switch to more energy-efficient modes in response to fine-grain activity imbalance.

Figure 2.5(c,d) illustrates the performance overhead of slow response times more clearly, comparing a partial execution of a *radix sorting* application kernel with 1 µs and 100 ns voltage-settling response times. The performance overhead of slow response time can be seen in Figure 2.5(c) from the delay between the time that the core becomes active (black on the activity strip) and the time that FG-SYNC+ raises the core frequency (color change from blue to red). In Figure 2.5(d), the faster 100 ns response time enables FG-SYNC+ to quickly adapt to fine-grain core activity, causing the black activity strips and FG-SYNC+ decisions to "line up".

2.3.5 FG-SYNC+ Summary

There are several important insights from this study: (1) to improve both performance and energy efficiency at the same average power, at least three levels are required and four levels results in additional benefits; (2) increasing the granularity of voltage scaling in space results in increased performance and energy; (3) systems require voltage settling response times on the order of 100 ns to exploit fine-grain activity balance.

2.4 FGVS Circuit Design: RPDNs

The three primary types of step-down voltage regulators are linear regulators, inductor-based switching regulators, and capacitor-based switching regulators. These regulators can be evaluated based on four key metrics: (1) *integration complexity*, i.e., does the regulator require extra non-standard fabrication steps?; (2) *area overhead and power density*, i.e., how much regulator area is required to deliver a certain amount of power?; (3) *power efficiency*, i.e., ratio of the output power to the supplied input power; and (4) *response time*, i.e., how fast can the target output voltage be adjusted?

Linear voltage regulators (also called linear dropout (LDO) regulators) are an example of a non-switching regulator. LDOs use a power MOSFET as a variable resistor, with a high-gain amplifier wrapped in a feedback configuration to reduce output resistance. At first glance, the lack of energy storage elements seems to imply LDOs will have much lower area overheads. However, a large decoupling capacitor is still required because the feedback loop in LDOs has limited bandwidth. As such, 10–15% of the chip area must be reserved for decoupling capacitance to maintain supply integrity for processor cores and logic during large current steps [HKB+05]. In addition, the maximum achievable power efficiency is the ratio of the output/input voltages since the LDO effectively acts as an adjustable resistance. This means that LDOs are highly inefficient for large voltage drops.

Inductor-based switching voltage regulators (also called Buck converters) are the traditional off-chip regulators of choice due to the potential for high power efficiency over wide voltage and current ranges; they also have excellent voltage regulation capabilities. However, in a fully on-chip buck converter, the efficiency is severely limited by the size and parasitics of the inductor. Reduction of these parasitics is the key to an efficient buck converter as shown in recently published work [HSH+05, ASL+09, KH11, KBW12]. These designs have reasonable efficiencies only for relatively low step-down ratios, which makes them less suitable for the wide dynamic range required for FGVS. These regulators also provide relatively low power densities on the order of $0.2\,\mathrm{W/mm^2}$. Unfortunately, solutions with higher power densities require magnetic materials, complicated post-fabrication steps, or interposer chips [WH08, SPW+11].

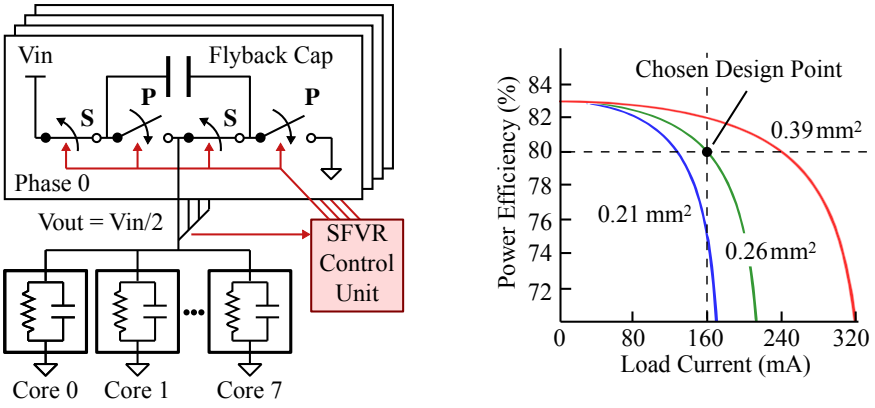
Capacitor-based switching voltage regulators (also called switched-capacitor (SC) regulators) work by alternately switching a set of capacitors with a given divide ratio from series (charge up) to

shunt configuration (discharge). This switching must be fast enough to maintain the output voltage across a load. SC regulators are capable of excellent efficiencies, however, they can only support certain discrete voltage divide ratios (e.g., 3:1, 2:1, 3:2) and usually require more than eight phases to reduce ripple losses [See09]. The regulation and output voltage range shortcomings of SC converters are balanced by their potential for higher power densities of 0.8–2 W/mm^2 [SNL$^+$10, LSA11, CMJ$^+$10] when using energy-dense on-chip capacitors. Note that in contrast to Buck converters, the energy density of MOS, MIM, and deep trench capacitors is sufficient to avoid the need for any off-chip or in-package energy storage elements. Due to the nature of operation, half of the capacitance in a SC regulator is always seen between the regulator output and ground thereby acting as an effective decoupling capacitance [LSA11]. This means that an explicit decoupling cap may not be necessary, which can further reduce the area overhead of SC regulators. Unlike buck converters which are fundamentally impossible to scale for smaller loads without incurring prohibitive losses, SC regulators can be scaled by simply adjusting the size of the capacitor and the switches. Consequently, SC regulators can be easily subdivided into modules that can be added together in parallel based on demand. All of the above reasons motivate our interest in exploring on-chip SC regulators for FGVS.

Based on the results from Section 2.3, a 4-level, 8-domain FG-SYNC+ configuration provided the best performance and energy efficiency. In the remainder of this section, we will consider three different integrated voltage regulator designs suitable for use with the target system described in Section 2.2: (1) a baseline design which uses a single integrated fixed-voltage regulator (SFVR); (2) multiple adjustable voltage regulators (MAVR) with one regulator per core; and (3) a new approach based on a reconfigurable power distribution network (RPDN).

2.4.1 SFVR: Single Fixed-Voltage Regulator

A single fixed-voltage regulator (SFVR) provides a good baseline to compare against more sophisticated regulation schemes. Figure 2.6(a) illustrates a basic 2:1 switched-capacitor design. In *series mode*, the flyback capacitor is connected in series with the load (cores), and the input voltage source charges up the flyback capacitor. In *parallel mode*, the flyback capacitor is connected in parallel with the load, and the input voltage source is disconnected. In parallel mode, the flyback capacitor acts as an energy source that is discharged to supply power to the cores. As the converter switches between the series and parallel modes, the output voltage will gradually



(a) Abstract Schematic

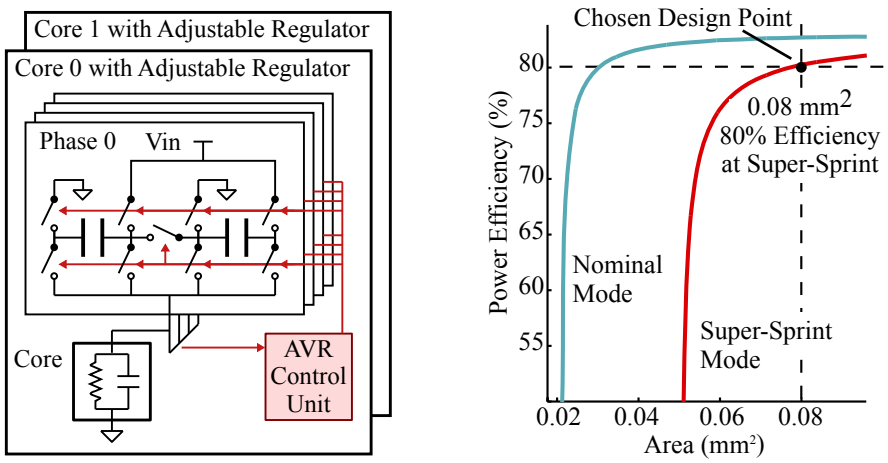
(b) Power Efficiency, Current, Area

Figure 2.6: SFVR – (a) 2:1 topology converts Vin to Vin/2 for eight cores; S = switches closed during serial mode; P = switches closed during parallel mode; control unit monitors Vout to regulate the switching frequency; 16 phases are included to reduce ripple (only four phases shown for simplicity). (b) for a fixed voltage, power efficiency varies as a function of output current and flyback capacitance area.

converge to half the input voltage. Faster switching frequencies reduce voltage ripple but decrease efficiency due to switching losses. The switching frequency is also used for fine-grain control of the output voltage. An SFVR control unit monitors the output voltage and adjusts the switching frequency in order to keep the output voltage constant across load current variations. Realistic SC regulators almost always include support for switching multiple phases of the signal in parallel to further minimize ripple. Larger flyback capacitors require more area, but can enable slower switching frequencies and therefore higher efficiencies for a given output voltage and load current. Figure 2.6(b) illustrates this trade-off using an analytical circuit-level model described in more detail in Section 2.5. For a fixed output voltage, as the regulator area increases, the curve moves to the right and broadens, indicating that (1) higher efficiencies can be achieved for the same output current and (2) higher output current can be achieved for the same efficiency. For our TSMC 65 nm process, we explored a variety of different SFVR designs and ultimately chose a configuration that can provide 80% efficiency at 1 V with an area of 0.26 mm^2 (4% of the core/L1 area). It may be possible to further reduce the area overhead by re-purposing the mandatory on-chip decoupling capacitance as flyback capacitance [LSA11].

2.4.2 MAVR: Multiple Adjustable-Voltage Regulators

To enable fine-grain voltage scaling in space and level, we require multiple adjustable voltage regulators (MAVR). Given the voltage levels from Section 2.3, we use the more complicated fly-



(a) Abstract Schematic

(b) Power Efficiency vs. Area

Figure 2.7: MAVR – (a) one adjustable voltage regulator (AVR) per core, only two shown for simplicity; control unit can configure flyback capacitance to convert Vin to Vin/2 and 3Vin/2; other intermediate voltages are possible by adjusting the regulation frequency; 16 interleaved phases are included to reduce ripple (only four phases shown for simplicity). (b) for a fixed output voltage and current, power efficiency varies as a function of area; area for nominal is over-provisioned for the sake of high efficiency at super-sprint.

back capacitor topology shown in Figure 2.7(a). For a 2.2 V input, MAVR achieves the highest efficiency at the following discrete voltage ratios: 1.0 V@2:1 = 82.7% and 1.33 V@3:2 = 80%. Adjusting the switching frequency enables the two remaining target voltage levels: 0.7 V@2:1 = 62% and 1.15 V@3:2 = 75%. Figure 2.7(b) illustrates the efficiency vs. area trade-off in MAVR. A regulator that must support both nominal and super-sprinting modes requires significantly more area compared to a regulator that only supports the nominal mode. Super-sprinting is simply not possible if the regulator area is less than 0.05 mm^2 since the regulator cannot switch fast enough to provide the required output current. For our TSMC 65 nm process, we explored a variety of different MAVR designs and ultimately chose a per-core regulator area of 0.08 mm^2 which allows efficient voltage regulation from resting to super-sprint. Due to the high output power variation between core operating modes, each AVR control unit must handle significantly larger switching frequency variation than its SFVR counterpart. In order to keep the output voltage stable at low power, the AVR control unit's feedback loop must be slow enough to avoid voltage overshoots; this in turn leads to long voltage-settling response times. Figure 2.8 uses detailed transistor-level simulations to illustrate the response time of various operating mode transitions for a single regulator in MAVR. Section 2.5 describes the methodology used for this analysis in more detail. Most transitions take several microseconds, with the nominal to super-sprint transition taking 2.9 µs. While MAVR does enable FGVS, it does so with high area overhead and long response times.

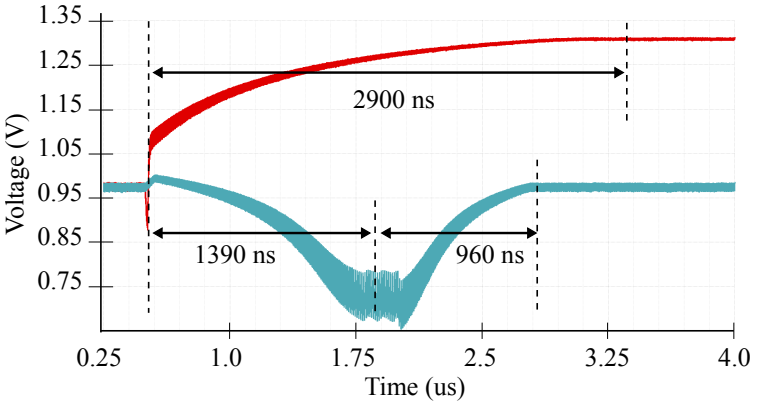


Figure 2.8: MAVR Transient Response – Transistor-level transient simulation of MAVR design with fixed capacitance per core.

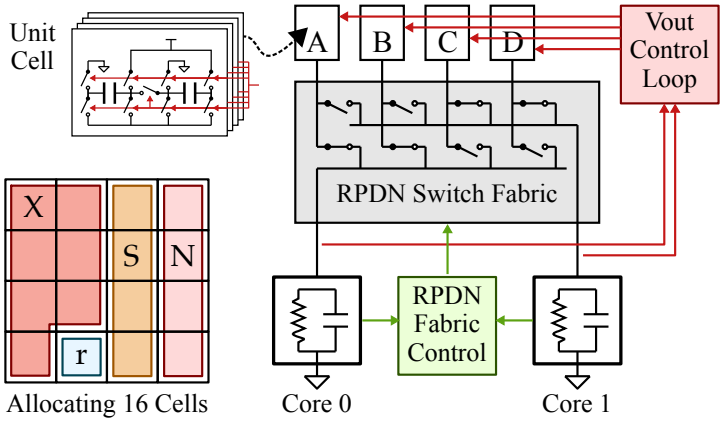


Figure 2.9: RPDN With Adjustable Unit Cells – All unit cells are designed for an adjustable output voltage from 0.7–1.33 V. Key RPDN blocks for two cores are shown. Diagram on the bottom left shows how unit cells are allocated across resting (r), nominal (N), sprinting (S), and super-sprinting (X) cores in the sub-RPDN.

2.4.3 RPDN: Reconfigurable Power Distribution Networks

Based on our insight from Section 2.3, we make the key observation that MAVR is significantly over-provisioned for FG-SYNC+. Each per-core regulator in MAVR must independently support the super-sprinting mode, but only one or two cores will ever be using this mode at any given time. While it might be possible to use thread migration and a fixed assignment of cores to voltage levels [RWB09, YLH+12], thread migration can introduce non-trivial performance and energy overheads. We take an architecture and circuit co-design approach to design reconfigurable power distribution networks (RPDNs) that meet the needs of FG-SYNC+ while reducing area overhead. RPDN allows sprinting cores to effectively "borrow" energy storage from resting cores to avoid over-provisioning the aggregate energy storage.

Figure 2.9 illustrates a simple example of an RPDN for two cores. The *RPDN control unit* configures the *RPDN switch fabric* to connect *RPDN unit cells* to supply power to each of the cores. In this example, there are four unit cells and each cell is a small switched-capacitor converter capable of 2:1 and 3:2 operation. The RPDN switch fabric is a two-input, two-output crossbar. The RPDN switch fabric is initially configured such that cells A and B supply core 0 while cells C and D supply core 1. If core 0 is waiting while core 1 is active, the RPDN switch fabric can be reconfigured such that cell A supplies low power to core 0 while cells B–D supply high power to core 1. Essentially, core 1 can borrow energy storage from core 0 on-demand.

The design in Figure 2.9 is greatly simplified to illustrate the basic concept of RPDNs. Our actual RPDN design includes 32 unit cells with eight phases per cell and can power eight cores. Preliminary estimates show that scaling the RPDN switch fabric across all eight cores incurs significant losses. In response, we partitioned the RPDN into two isolated sub-RPDNs. Each sub-RPDN has half of the 32 unit cells to distribute to a four-core partition. Each unit cell uses a multi-level SC regulator design that enables 2:1 and 3:2 step-down conversions, similar to the regulator shown in Figure 2.7(a), except with only 8 phases. Based on TSMC 65 nm transistor-level models, the sub-RPDN switch fabric introduces a 0.25–0.75% efficiency degradation with 8% extra converter area.

Figure 2.10 shows the efficiency vs. output power for a single core as a function of the number of cells allocated to that core for a cell area of 0.011 mm^2. Four cells are required to efficiently support the nominal mode, while seven cells are required to efficiently support the super-sprinting mode. Since the sprinting mode uses a different flyback capacitor topology, it is able to achieve reasonable efficiency with the same number of cells as the nominal mode. Resting mode consumes very little power, so a single cell is sufficient. The inset in Figure 2.9 shows how unit cells of one sub-RPDN can be allocated to four cores operating in four different modes. The two cores operating in the nominal and sprinting modes are allocated four cells each. The resting core only requires a single cell, so the super-sprinting core "borrows" three cells from the resting core. MAVR must provision for the worst case, so each per-core regulator must include flyback capacitance equivalent to seven cells. RPDN provides an average of just four cells per core, and then uses reconfiguration to create seven-cell regulators for super-sprinting on-demand.

The RPDN architecture offers obvious advantages in terms of area savings. Based on our analytical model described in Section 2.5, we compute the area overhead for SFVR, MAVR, and

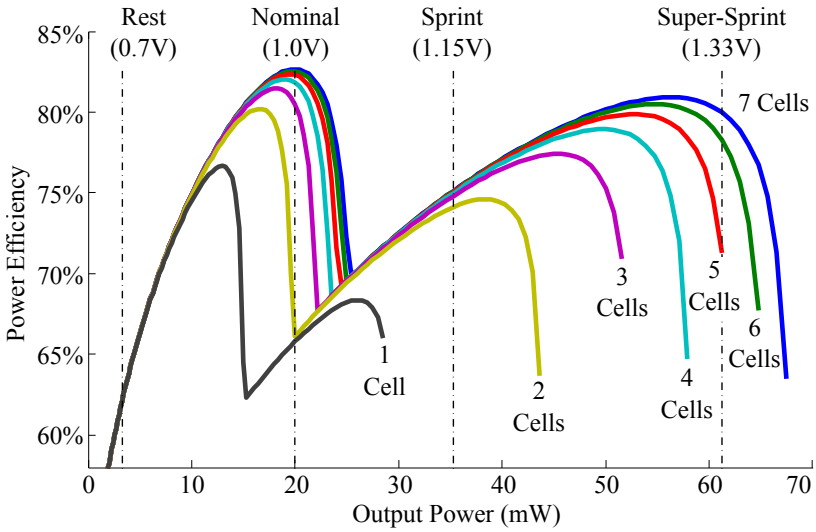


Figure 2.10: RPDN Power Efficiency vs. Output Power For Single Core – Each cell is 0.011 mm^2. The 7-cell RPDN curve also represents MAVR area. RPDN uses 1,4,4,7 cells for rest, nominal, sprint, and super-sprint, respectively.

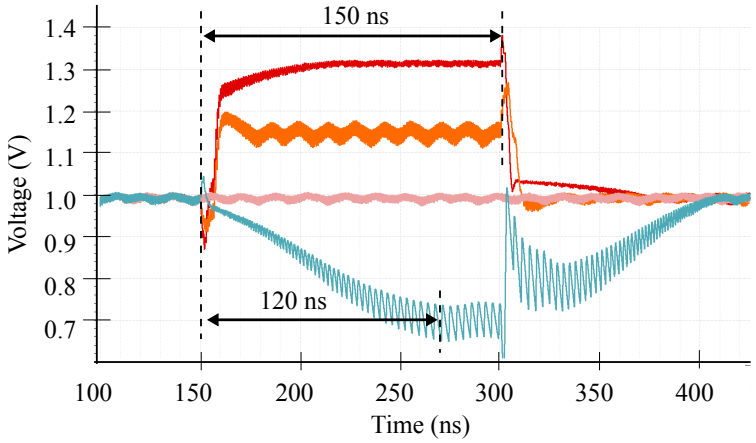


Figure 2.11: RPDN Transient Response – Transistor-level transient simulation of a sub-RPDN illustrating the benefit of capacitance reallocation. Four cores start at nominal, then three cores move to sprint, super-sprint, and rest and then back to nominal.

RPDN to be 4%, 10%, and 6% respectively. This means that RPDN provides area savings of 40% over MAVR when supporting per-core supply regulation across the same number of cores. In addition to reducing area overhead, RPDN also significantly reduces the voltage-settling response time. For resting cores, RPDN uses 15% of MAVR's area which allows the RPDN control loop to be much faster. Furthermore, when switching between different operating modes, RPDN changes capacitance in addition to the SC divide ratio and switching frequency. This means the RPDN control loop has to make a significantly smaller switching frequency adjustment in order to accommodate the new operating mode. Figure 2.11 shows the transient response for one sub-RPDN where each

	PDN Area	Power Efficiency for Vout =			Transient Response (ns)			Voltage Scaling	
	(mm^2)	0.7V	1.0V	1.33V	Min	Typ	Max	Space	Time
SFVR	0.26	n/a	80%	n/a	n/a	n/a	n/a	No	No
MAVR	0.64	62%	82.7%	80%	164	1950	3850	Yes	Yes
RPDN	0.37	62%	81.8%	80%	36	120	226	Yes	Yes

Table 2.1: Comparison of SFVR, MAVR, and RPDN

core switches to a different operating mode. The response time for the nominal to super-sprint transition takes just 150 ns.

2.4.4 Summary of Power Distribution Networks

Table 2.1 summarizes the trade-offs discussed throughout this section. While on-chip voltage regulation offers the potential for fast and flexible control, it also incurs various overheads. In the case of SFVR, no flexibility is offered. MAVR provides the flexibility for fine-grain voltage scaling, but at the cost of high area overhead and long response times. Finally, RPDN offers an interesting middle ground. RPDN enables the flexibility of MAVR with significantly reduced area overhead and faster response times.

2.5 Evaluation Methodology

We used a vertically integrated evaluation methodology that uses a mix of circuit-, gate-, register-transfer-, and architectural-level modeling. Circuit-level modeling is used to characterize each power distribution network (PDN); gate- and register-transfer-level modeling are used to build accurate area and energy models of our target embedded processor; and architectural-level modeling is used to analyze the system-level impact of each PDN.

2.5.1 Circuit-Level Modeling

We performed SPICE-level circuit simulations of all SC regulators with Cadence Spectre using models from a TSMC 65 nm process. We also used an analytical SC model to enable faster design-space exploration of regulator efficiency vs. power, supply voltage, and area as well as for estimat-

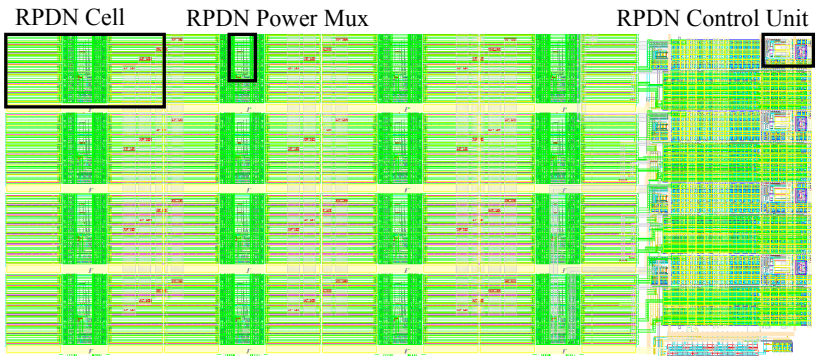


Figure 2.12: Tapeout-Ready Layout of a sub-RPDN Test Chip – We use the SPICE-level extracted model to validate accuracy of our analytical model for efficiency, area, and transient response.

ing the transient response. Our analytical SC model, which computes switching losses, gate drive losses, bottom plate losses, and series resistive losses, is based on prior work by Seeman [See09]. After carefully extracting capacitor density, bottom plate capacitance, and switch parasitics from the Cadence 65 nm PDK, our analytical model and Cadence simulation results match closely in both absolute efficiency and area numbers. To properly account for a realistic RPDN switch fabric and clock distribution overheads, we performed a full-chip layout of a 16-cell sub-RPDN (see Figure 2.12). SPICE-level simulations were also used to help determine the relationship between voltage and frequency for our cores across different operating modes. We used nine delay stages consisting of multiple FO4 loaded inverters, NAND, and NOR gates connected in a loop, such that the total delay in the loop matches our RTL cycle time for a given voltage. We used the change in delay vs. supply voltage as a model for core voltage-frequency scaling. Finally, we augmented our SC analytical model to account for shunt losses due to capacitor leakage. We used this augmented model to study scaling towards leakier processes, as described in Section 2.7.

2.5.2 Gate- and RTL Modeling

To estimate power, we created an instruction-level energy model derived from a realistic RTL implementation of an in-order, single-issue scalar core and L1 memory system. The RTL model is synthesized and placed-and-routed using a combination of Synopsys DesignCompiler, IC Compiler, and PrimeTime PX with a TSMC 65 nm standard-cell library characterized at 1 V. We then ran a suite of *energy microbenchmarks* that are each designed to measure the energy of a specific instruction. For example, the `addiu` energy microbenchmark warms up the instruction cache (to isolate the energy solely due to the instruction under test) and then executes 100 `addiu` instruc-

tions in sequence. We ran this microbenchmark on: (1) the synthesized gate-level design to obtain bit-accurate traces that are fed into PrimeTime power simulations for power estimates, and (2) on the RTL simulator to obtain cycle counts for the execution. Coupled with the cycle time of the placed-and-routed design, we can calculate the energy per `addiu` instruction.

Only a subset of instructions are characterized in this way. For instance, `sll` and `srl` are similar enough that we only needed to characterize one of these instructions. Separate energy microbenchmarks are used to quantify the energy per taken-branch versus a not-taken-branch. Similarly, we tested cases for load and store hits and misses separately. In general, we see a range of 60–75 pJ per arithmetic instruction, with higher ranges for long-latency and memory instructions. We used these results to build an energy dictionary containing the energy for every instruction. The energy dictionary can be applied to an RTL or cycle-level trace containing the distribution of dynamic instruction types to produce a total energy and power estimate for the simulation. Furthermore, we can leverage the voltage-frequency relationship derived from our circuit-level modeling to scale the energy and power of the nominal configuration to other voltage-frequency pairs.

2.5.3 Cycle-Level Modeling

We use the gem5 simulator [BBB+11] in syscall emulation mode to model a multicore processor with eight single-issue in-order cores, each with private 16 KB L1 I/D caches and sharing a 1 MB L2 cache. We have extended gem5 to enable architecture and circuits co-design in several ways.

Multithreading Support in Syscall Emulation Mode – We modified gem5's address space mapping to allow cores to share memory in syscall emulation mode, and added support for a simple multi-threading library that pins threads to cores.

Software Hints – We modified gem5 to toggle an activity bit in each core after executing the new activity hint instruction. We also added support for the "work left" hint instructions to pass thread progress information from parallelized loops to the hardware.

Dynamic Frequency Scaling Support – We modified gem5's clock domains and clocked object tick calculations to support dynamic frequency scaling. Cores can independently scale their frequency, but we centralized control of all clock domains in the FG-SYNC+ controller.

Integration with RTL-Based Power Model – We modified gem5 to capture detailed, per-core instruction counts that occur not only in each DVFS mode, but also in each DVFS mode *transition* to properly account for energy overheads during these transitions. Using these statistics and our energy dictionary, we calculate energy/power for each configuration.

Integration with Circuits – We integrated voltage settling response times from circuit-level simulations for each mode transition into our gem5 frequency scaling framework. We use these response times to delay frequency change events to simulate realistic voltage scaling. We also use our circuit-level analytical SC regulator model (verified with SPICE simulations) to integrate regulator energy efficiencies into our power model to obtain realistic energy and power overheads.

2.5.4 Application Kernels and Benchmarks

We use a variety of custom application kernels as well as selected PARSEC, SPLASH-2, and PBBS benchmarks on our architectural-level model to analyze the system-level benefit of various configurations (see Table 2.2).

bfs computes the shortest path from a given source node to every reachable node in a graph using the breadth-first-search algorithm and is parallelized across the wavefront using double buffering. *bilat* performs a bilateral image filter with a lookup table for the distance function and an optimized Taylor series expansion for calculating the intensity weight. *dither* generates a black-and-white image from a gray-scale image using Floyd-Steinberg dithering. Work is parallelized across the diagonals of the image, so that each thread works on a subset of the diagonal. A data-dependent conditional allows threads to skip work if an input pixel is white. *rsort* performs an incremental radix sort on an array of integers. During each iteration, individual threads build local histograms of the data, and then a parallel reduction is performed to determine the mapping to a global destination array. Atomic memory operations are necessary to build the global histogram structure. *kmeans* implements the k-means clustering algorithm. Assignment of objects to clusters is parallelized across objects. The minimum distance between an object and each cluster is computed independently by each thread and an atomic memory operation updates a shared data structure. Cluster centers are recomputed in parallel using one thread per cluster. *mriq* computes a calibration matrix used in magnetic resonance image reconstruction algorithms. *pbbs-dr* is a PBBS application for 2D Delaunay Mesh refinement. Work is parallelized across the bad triangles. Parallel threads move through reserve and commit phases and will only perform retriangulation if all

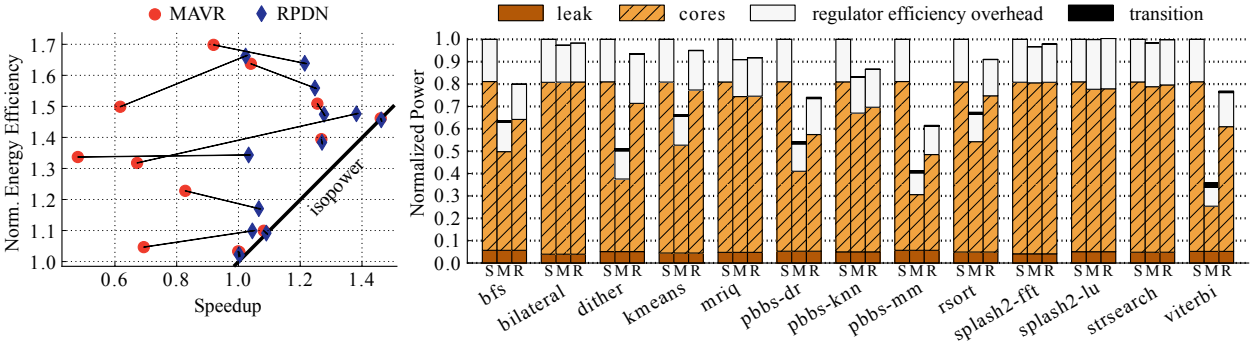
the neighbors they marked were reserved successfully. Newly generated bad triangles are assigned to other threads. *pbbs-knn* is a PBBS application that, given an array of points in 2D, finds the nearest neighbor to each point using a quadtree to speed up neighbor lookups. Quadtree generation is parallelized at each depth of the tree so that each thread works on a separate sub-quadrant. The quadtree is used to find the nearest neighbor to each point in parallel. *pbbs-mm* is a PBBS application for maximal matching on an undirected graph. Work is parallelized across the edges in the graph. Parallel threads move through reserve and commit phases. Threads attempt to mark the endpoints of the assigned edge with the edge ID. In the commit phase, threads will only mark its edge as part of the maximal matching if both endpoints were reserved successfully. *splash2-fft* is a SPLASH-2 benchmark that performs a complex 1D version of a radix-sqrt(n) six-step FFT algorithm. Cores are assigned contiguous sets of rows in partitioned matrices. Each core transposes contiguous sub-matrices from every other core and transposes one locally. *splash2-lu* is a SPLASH-2 benchmark that performs a matrix factorization into a lower and upper triangular matrix. Parallelization is across square blocks of size B and this parameter is picked so that blocks fit in the cache. *strsearch* implements the Knuth-Morris-Pratt algorithm to search a collection of byte streams for the presence of substrings. The search is parallelized by having all threads search for the same substrings in different streams. The deterministic finite automatas used to model substring-matching state machines are also generated in parallel. *viterbi* decodes frames of convolutionally encoded data using the Viterbi algorithm. Iterative calculation of survivor paths and their accumulated error are parallelized across paths. Each thread performs an add-compare-select butterfly operation to compute the error for two paths simultaneously, which requires unpredictable accesses to a lookup table.

2.6 Evaluation Results

In this section, we compare the SFVR, MAVR, and RPDN designs. We evaluate the performance, energy efficiency, and power of our applications as they run on our target system with each type of power distribution network (PDN). We choose a 4-level, 8-domain FG-SYNC+ controller based on the FGVS study in Section 2.3. From our circuit-level study in Section 2.4, we account for realistic voltage-settling response times and regulator power efficiencies in each DVFS mode for varying load currents.

		Performance			Energy			Trans	
App	**DInsts**	**SFVR (µs)**	**MAVR**	**RPDN**	**SFVR (µJ)**	**MAVR**	**RPDN**	**MAVR**	**RPDN**
bfs	58	101	1.04	1.25	17	0.61	0.64	350	693
bilateral	6540	3254	1.00	1.00	767	0.97	0.98	1	3
dither	2762	4121	0.67	1.38	749	0.76	0.68	360	1939
kmeans	483	355	0.69	1.05	74	0.96	0.91	309	970
mriq	8318	8409	1.27	1.27	1640	0.72	0.72	3	6
pbbs-dr	20057	39785	0.92	1.21	7010	0.59	0.61	429	2966
pbbs-knn	645	850	1.26	1.28	158	0.66	0.68	53	63
pbbs-mm	305	714	0.62	1.02	117	0.67	0.60	452	3836
rsort	268	220	0.83	1.07	42	0.81	0.85	336	754
splash2-fft	4146	2226	1.00	1.00	502	0.97	0.98	9	18
splash2-lu	7780	13045	1.46	1.46	2390	0.68	0.69	5	9
strsearch	1434	1101	1.08	1.09	212	0.91	0.92	17	28
viterbi	3522	4465	0.48	1.03	798	0.75	0.74	558	4599

Table 2.2: Application Performance and Energy – DInsts = dynamic instruction count in thousands; Trans = transitions per ms. MAVR/RPDN performance and energy results are normalized to SFVR.



(a) Normalized Energy Efficiency vs Speedup

(b) System-level Power Breakdowns for SFVR, MAVR, RPDN

Figure 2.13: System-Level Evaluation for SFVR, MAVR, and RPDN – (a) MAVR and RPDN energy efficiency vs. performance normalized to SFVR. Lines connect the same application using both MAVR and RPDN. Raw numbers are given in Table 2.2. (b) Power breakdowns for SFVR, MAVR, and RPDN (S, M, R, respectively) normalized to SFVR. Bar stacks show leakage power, aggregate core power, regulator power efficiency overhead, and transition power overhead.

Figure 2.13(a) compares MAVR and RPDN energy efficiency and speedup, both normalized to SFVR. The raw numbers are given in Table 2.2. MAVR has sharp slowdowns for many applications, modest speedups for others, and very high energy efficiency across most applications. Notice that sharp slowdowns generally occur for applications with higher transitions per millisecond (e.g., *dither, kmeans, viterbi*, see Table 2.2). This implies that MAVR response times are too slow for FG-SYNC+ to adapt to the fine-grain activity imbalance. RPDN has higher performance as well as higher energy efficiency across most applications, including those with higher transi-

tions per millisecond, all at similar average power compared to SFVR. Notice that the results in Figure 2.13(a) look very similar to those in Figure 2.3(d). Table 2.1 explains the similarity: the typical MAVR voltage-settling response time is on the order of 1000 ns while the typical RPDN response time is on the order of 100 ns. RPDN's order-of-magnitude faster response time is a key enabler for achieving good performance and energy efficiency when exploiting fine-grain activity imbalance with FGVS.

Figure 2.13(b) shows power breakdowns for each application running on our target system with each type of PDN. The results for SFVR include the power of eight cores running the application at nominal voltage, the power lost in the 2:1 SC converter with an 80% conversion efficiency, and leakage power overhead. MAVR consumes significantly less power than SFVR by resting waiting cores, but MAVR has difficulty exploiting this power slack to improve performance due to slow response times. The impact of response time is tightly linked to how often cores transition. A delayed decision is very likely to remain optimal for applications that transition only rarely; these applications have speedups even with slow response times (e.g., *splash2-lu, mriq, strsearch*). Applications with the greatest slowdowns consume the least power (e.g., *viterbi*). This indicates that they spend most of their time executing slowly in low-power modes. RPDN closely tracks the average power of SFVR by resting cores and trading power slack for improved performance. RPDN actually achieves *lower* average power compared to SFVR for some applications. The extra power slack is an opportunity for further optimization using a more aggressive online controller. The results for regulator power efficiency overhead show that RPDN does not sacrifice regulator efficiency in exchange for performance. Leakage power and transition power remain fairly small for each type of PDN.

Figure 2.13 shows that in general, MAVR can offer high energy efficiency but suffers significant slowdowns compared to SFVR due to slow response times. RPDN enables higher performance and energy efficiency at the same or lower average power compared with SFVR, while reducing the area overhead by 40% compared to MAVR. These results suggest RPDN is an attractive option for enabling realistic fine-grain voltage scaling in future embedded systems.

2.7 Discussion

In this section, we discuss the impact of di/dt noise, as well as RPDN scalability for higher core counts, higher power densities, and different technologies.

Noise – On-chip power management can potentially diminish di/dt noise issues. One key motivation for moving power management on-chip is to reduce the impact of PCB wires and package parasitics (e.g., bond pads). In addition, using on-chip step-down voltage converters reduces the package supply-plane impedance since lower current is delivered for the same power. Lastly, for on-chip SC regulators, a portion of the flyback capacitance effectively acts as additional decoupling capacitance. While the additional wiring required for RPDN will introduce some parasitic inductance at high frequency, on-chip wires are short compared to board- and package-level wires. Future work can potentially investigate a detailed characterization of di/dt noise for various integrated regulator designs.

Scaling Core Count – Scaling core count by directly scaling the RPDN switch fabric (i.e., each RPDN cell connected to every core) complicates wiring and has high efficiency losses. In this work, we addressed scalability by partitioning the RPDN into two sub-RPDNs, where each sub-RPDN is assigned to a cluster of four cores. We carefully picked our rest, nominal, sprint, and super-sprint levels such that the cells in a single RPDN partition can support any DVFS mode decision made by FG-SYNC+ at high efficiency. When scaling to larger core counts, there may be certain configurations where there simply is not enough intra-partition energy storage to support the desired operating modes. The FGVS controller would need to account for these scenarios and react accordingly. Future work can potentially explore more sophisticated RPDN switch fabrics. For example, one could imagine RPDN fabrics that provide just "nearest neighbor" connectivity or use multiple stages of switching.

Scaling to Higher Power Densities – Our target system has a power density of 25 $\mathrm{mW/mm^2}$ at nominal voltage, but our findings in this chapter still hold true for higher power densities. PDN area overhead increases roughly linearly with core power density; this means that high-power, high-complexity cores will still benefit from RPDN area savings over MAVR. For example, a $4\times$ increase in core power density to 100 $\mathrm{mW/mm^2}$ would have $4\times$ larger area overheads for the integrated PDN compared to the core, but the relative area savings of RPDN are the same (i.e.,

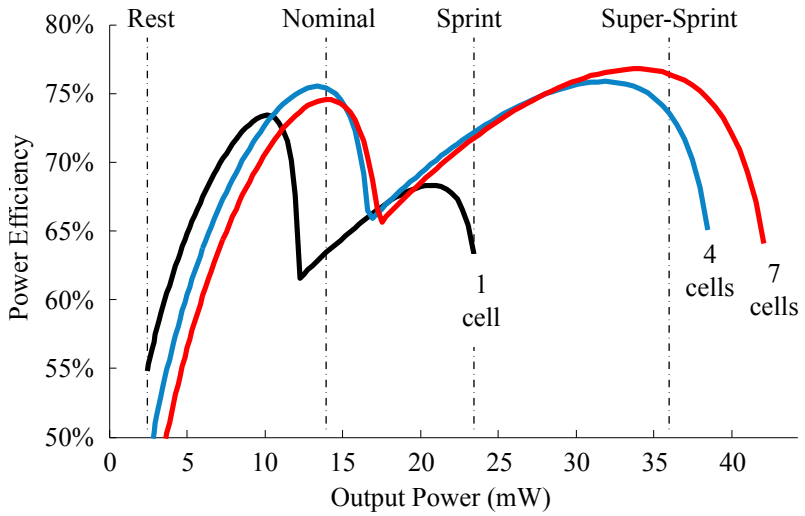


Figure 2.14: RPDN Power Efficiency vs. Output Power For Single Core in a Leakier 65 nm Process – Each cell is 0.015 mm^2. The 7-cell RPDN curve also represents MAVR area. RPDN tracks the optimal efficiency across all curves, either matching or outperforming MAVR in each operating mode.

RPDN area overhead of 24% versus MAVR area overhead of 42% still means RPDN saves 40% area compared to MAVR).

Scaling to Different Technologies – Integration of switching regulators on-chip is a recent phenomenon and is a direct result of the impacts of technology scaling. Earlier CMOS generations did not allow for integration of efficient switching regulators due to low-quality switches and low energy-density passives that required high switching frequencies. Analog components generally do not improve with aggressive scaling, but switching voltage regulators are an exception. Better switches in smaller technology nodes such as 28 nm and beyond are likely to improve SC regulator efficiency rather than degrade it. Furthermore, advances in technology that increase capacitor density (e.g., deep-trench capacitors [AKK+13, CMJ+10]) have the potential to make integrated regulators for high-power systems both relevant and efficient. Finally, as CMOS technology scales, there is potential for increased leakage. In this case, RPDN offers an additional benefit compared to MAVR. Figure 2.14 shows energy efficiency vs. output power for a single core as a function of the number of cells allocated to that core (similar to Figure 2.10), except for a leakier 65 nm process with rest, nominal, sprint, and super-sprint levels scaled accordingly. Notice that in a leakier process at low power, using seven cells results in lower power efficiency compared to using just one cell. RPDN enables reduced flyback-capacitor leakage by using just a single cell when cores are operating in a low-power resting mode.

2.8 Related Work

Most of the previous work in on-chip voltage regulation explores the design space between off-chip and on-chip regulators while focusing on the potential for energy savings at similar performance. [KGWB08] conducted sensitivity studies for on-chip Buck regulators and contrasted their lower efficiencies and faster response times with slow, efficient off-chip regulators. The authors took into account overheads for various off-chip and on-chip cases and concluded that on-chip per-core voltage regulation could reduce energy, albeit at large area overhead. The authors determined the optimal supply voltage for each benchmark offline and did not explore online controllers. [LCVR03] focused on multiprogrammed applications with an emphasis on saving energy on memory- vs. cpu-bound applications, rather than on exploiting fine-grain activity imbalance in multithreaded workloads.

Other works, such as [YLH+12] recognized similar trade-offs as we do in on-chip voltage conversion. The work proposed to use an S-factor-based algorithm to identify workloads with potential for energy savings and then to migrate these threads to a dedicated core powered by an on-chip regulator. With offline training, each application is identified and binned at runtime for migration. This approach was only shown to save energy with multiprogrammed workloads. Taking the same approach for multithreaded applications with fine-grain activity imbalance would likely incur high migration overhead, but the process of identifying workloads offline can complement our work by allowing our controllers to search for complex application-level patterns. In [DR07], an online learning algorithm for power scaling is proposed that could be implemented on top of our controller to help identify long-term application behaviors.

A switched-capacitor converter is used for dynamic voltage-frequency control in [JLB+15]. In this work, the SC converter alternates between different topologies while the core frequency tracks the output voltage ripple. [ASSK11, CCK07] considers Buck converters and focuses on jointly optimizing power consumption of the converter and the core by finding the lowest computational energy point; they considered only steady-state responses. [MPT+12] elected to switch cores between two supply rails. This approach requires dedicated supply rails and incurs non-trivial supply pin overhead, which is increasingly important in future technologies [HSN+07]. In addition, care must be taken when switching cores between different supply rails by either scheduling power gating events [RGH+09], reducing the supply plane impedance, or by increasing decoupling ca-

pacitance [DGP+12]. Integrated SC converters allow dynamic regulation without the need for large decoupling capacitance while simultaneously relaxing the impedance requirements placed on power-supply routing by utilizing higher off-chip voltages.

2.9 Conclusion

Recent trends in technology and the drive to integrate more functionality on chip have generated significant interest in an on-chip voltage regulation, with the goal of reducing cost and enabling fine-grain voltage scaling (FGVS). In this chapter we present a new controller, FG-SYNC+, specifically designed for FGVS. Our FG-SYNC+ analysis demonstrates the importance of exploiting fine-grain scaling in level, space, and time. We used insights from our analysis to motivate a new voltage regulation scheme based on the idea of reconfigurable power distribution networks (RPDNs). RPDNs avoid the need to over-provision per-core voltage regulators, thereby reducing regulator area overhead while simultaneously improving performance. Our promising results suggest that there is an important opportunity for architecture and circuit co-design of integrated voltage regulation in future systems.

CHAPTER 3
ASYMMETRY-AWARE WORK-STEALING RUNTIMES

Chapter 2 demonstrated great benefit from applying fine-grain power control to homogeneous multicore systems running general applications. This chapter broadens the scope to heterogeneous multicore systems while also specializing the techniques for the domain of productive task-based parallel runtimes. This work focuses on fine-grain voltage and frequency scaling for both big and little cores at microsecond timescales. In addition, while the previous chapter relied on simple static work distribution and sped up lagging cores, in this chapter we focus on productive task-based parallel runtimes that use sophisticated work-stealing algorithms for better load balancing. I argue that work-stealing algorithms are a natural fit for managing on-chip asymmetry at the software level. This chapter explores how these software runtimes can be made aware of underyling asymmetry in the architecture and VLSI layers to create more efficient schedules and to dynamically tune processing elements.

3.1 Introduction

Work stealing is a well-known approach to task distribution that elegantly balances task-based parallelism across multiple worker threads [BS84, RHH84]. In a work-stealing runtime, each worker thread enqueues and dequeues tasks onto the tail of its task queue. When a worker finds its queue empty, it attempts to steal a task from the head of another worker thread's task queue. Work stealing has been shown to have good performance, space requirements, and communication overhead in both theory [BL99] and practice [BJK+96, FLR98]. Optimizing work-stealing runtimes remains a rich research area [CL05, CM08, BM09, LSL12, DLS+09, CCHG12], and work stealing is a critical component in many popular concurrency platforms including Intel's Cilk++, Intel's C++ Threading Building Blocks (TBB), Microsoft's .NET Task Parallel Library, Java's Fork/Join Framework, X10, and OpenMP. Most of the past research and current implementations use *asymmetry-oblivious work-stealing* runtimes. In this work, we propose *asymmetry-aware work-stealing* (AAWS) runtimes, which exploit both *static asymmetry* (e.g., different core microarchitectures) and *dynamic asymmetry* (e.g., per-core dynamic voltage/frequency scaling) to improve the performance and energy efficiency of multicore processors.

Single-ISA heterogeneous processors integrate multiple cores with different microarchitectures onto a single die in order to provide distinct energy-performance operating points [KTR+04, KFJ+03]. These processors exhibit *static asymmetry* that is fixed at design time. Systems based on ARM's big.LITTLE architecture, which composes "big" ARM Cortex-A15/A57 out-of-order cores with "little" Cortex-A7/A53 in-order cores [Kre11, Gre11], are commercially available from Samsung [Gwe14b], Qualcomm [Gwe14a], Mediatek [Dem13], and Renesas [Gwe13]. There has been significant interest in new techniques for effectively scheduling software across these big and little cores, although most of this prior work has focused on either multiprogrammed workloads [KTR+04, KFJ+03, ARKK13, VCJE+12] or on applications that use thread-based parallel programming constructs [JSMP12, JSMP13, LLK09, SMQP09]. However, there has been less research exploring the interaction between state-of-the-art work-stealing runtimes and static asymmetry. A notable exception is Bender et al.'s theoretical work [BR02] and abstract discrete-event modeling [JB02] on an enhanced Cilk scheduler for heterogeneous systems. Others have also explored combining work-stealing with work-sharing, critical-path scheduling, and/or core affinity to more effectively schedule tasks across statically asymmetric systems [MWK+06, CCHG12, CIOQ15, CRB+15].

Dynamic voltage/frequency scaling (DVFS) is an example of *dynamic asymmetry* that is adjustable at runtime. Much of the prior work on DVFS has assumed off-chip voltage regulation best used for coarse-grain voltage scaling [BPSB00, IBC+06, IM06]. Recent architecture/circuit co-design of fine-grain DVFS (either through multi-rail voltage supplies [MPT+12, Dre11] or fully integrated voltage regulators [JLB+15, SGS+14, KGWB08, GTB+14, FBB+15, GSSK12]) suggests that sub-microsecond voltage transition times may be feasible in the near future. There has been significant interest in new techniques for exploiting fine-grain DVFS to improve the performance and/or energy efficiency of multiprogrammed workloads [KGWB08, EE11] or on applications that use thread-based parallel programming constructs [BM09, GTB+14, MPT+12, LMH04, CGR+08]. Again, there has been relatively little research exploring the interaction between work-stealing runtimes and dynamic asymmetry. A notable exception is recent work by Ribic et al. that proposes reducing the voltage/frequency of thieves and increasing the voltage/frequency of workers with deep task queues [RL14].

Recent studies have demonstrated the potential benefit of combining static and dynamic asymmetry [LPD+14, AML+10, GRSW04]. A key observation is that static asymmetry through het-

erogeneous core types offers larger marginal utility but must be fixed at design time, while dynamic asymmetry in the form of DVFS offers smaller marginal utility but can be varied at run time [AML+10]. These past works have focused on coarse-grain multiprogrammed workloads. To our knowledge, this is the first work to explore the interaction between static asymmetry, dynamic asymmetry, and work-stealing runtimes. We argue that work-stealing runtimes are a natural fit for managing asymmetry. Assuming fully strict programs with high-parallel slack [FLR98], a work-stealing runtime will naturally exploit asymmetry without modification; faster cores will execute tasks more quickly and then simply steal work as necessary from slower cores. However, our work shows there are still important opportunities for AAWS runtimes to improve performance and energy efficiency.

In Section 3.2, we develop a simple first-order model to provide insight into optimizing the aggregate throughput and energy efficiency of an AAWS runtime. Our model predicts that the maximum throughput will occur when the marginal utility (i.e., performance) vs. marginal cost (i.e., power) of each core is equal. This is an intuitive application of a fundamental principle in economics known as the *Law of Equi-Marginal Utility*. Others have also observed this important design guideline in the context of design-space exploration of processor microarchitecture [AML+10] and market-based multi-resource allocation in multicore processors [WM15]. We use numerical analysis to study the potential benefit of a marginal-utility-based approach in AAWS runtimes for both high-parallel (HP) and low-parallel (LP) regions.

In Section 3.3, we propose three new hardware/software mechanisms to enable AAWS runtimes: *work-pacing*, *work-sprinting*, and *work-mugging*. Work-pacing is a novel technique that directly applies a marginal-utility-based approach in the HP region by increasing the voltages of little cores and decreasing the voltages of big cores to improve both performance and energy efficiency. Work-sprinting is a novel technique similar to work-pacing that applies a marginal-utility-based approach in the LP region, while also exploiting additional power slack generated from resting waiting cores. Work-pacing and work-sprinting require a fine-grain DVFS controller specifically designed for an AAWS runtime as well as lightweight hint instructions that allow the AAWS runtime to inform the hardware when cores are active or waiting. While work-sprinting can provide some benefit in LP regions, it is also fundamentally limited by the work-stealing scheduler's task distribution algorithm. Our first-order modeling suggests that when distributing power among busy cores in the LP region, sprinting a little core while resting a big core is usually suboptimal com-

pared to resting a little core and sprinting a big core. To address this issue, we revisit previous theoretical work on work-mugging [BR02, JB02]. Work-mugging allows a waiting big core to "mug" an active little core by preemptively migrating a task from the little core to the big core. We describe a practical implementation of work-mugging in multicore systems that relies on user-level inter-core interrupts.

In Section 3.4, we outline our vertically integrated research methodology. We have developed our own C++ work-stealing runtime inspired by Intel TBB which uses Chase-Lev task queues [CL05] and occupancy-based victim selection [CM08]. We have ported 20 application kernels to our work-stealing runtime. These kernels are selected from the PBBS [SBF+12], PARSEC [BKSL08], Cilk [FLR98], and UTS [OHL+06] benchmark suites and represent diverse application-level characteristics. We evaluate the performance of two eight-core systems (4 big & 4 little; 1 big & 7 little) using the cycle-level gem5 simulator [BBB+11], and we evaluate power and energy efficiency using a detailed power/energy model that leverages component models within McPAT [LAS+13] as well as energy estimates from our own VLSI implementation of a single-issue in-order RISC processor. In Section 3.5, we use this methodology to explore the potential performance and energy efficiency benefits of AAWS runtimes. On a system with four big and four little cores, an AAWS runtime achieves speedups from 1.02–1.32× (median: 1.10×). At the same time, all but one kernel achieves improved energy efficiency with a maximum improvement of 1.53× (median: 1.11×).

The key contributions of this work are: (1) we develop a marginal-utility-based approach to both quantify the potential benefit of AAWS runtimes and motivate specific hardware/software techniques; (2) we propose new work-pacing and work-sprinting techniques that directly apply a marginal-utility-based approach within AAWS runtimes; (3) we provide a practical implementation of the previously proposed work-mugging technique suitable for use in AAWS runtimes; and (4) we use a vertically integrated research methodology that spans software, architecture, and VLSI to make the case that holistically combining static asymmetry, dynamic asymmetry, and work-stealing runtimes can improve both performance and energy efficiency in future multicore systems.

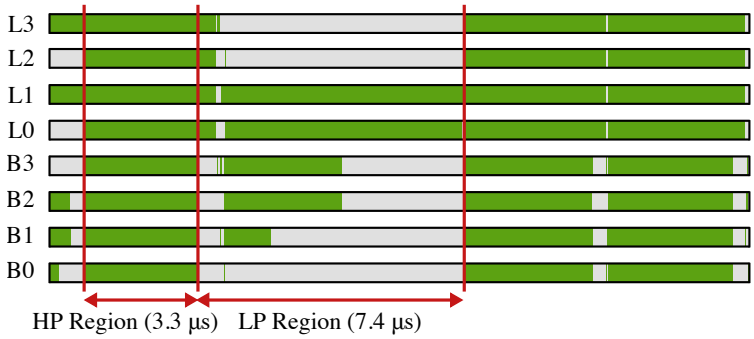


Figure 3.1: Activity Profile for Convex Hull Application on Statically Asymmetric System – Only a subset of the entire activity profile is shown. Cores L0–3 are "little" in-order cores; cores B0–3 are "big" out-of-order cores. Green = executing task; light-gray = waiting in work-stealing loop; HP = high-parallel; LP = low-parallel. See Section 3.4 for simulation methodology.

3.2 A Marginal-Utility-Based Approach

Figure 3.1 shows an activity profile for an example application running on a system with four big cores and four little cores and an asymmetry-oblivious work-stealing runtime. Notice that the application includes a mix of both high-parallel (HP) and low-parallel (LP) regions. During HP regions, the work-stealing runtime is able to adapt to the static asymmetry by distributing more tasks to the bigger cores resulting in good throughput and a relatively balanced profile. During LP regions, some cores are active, while other cores are waiting in the work-stealing loop. Notice that there is usually a small LP region near the end of an HP region, since the work-stealing runtime is unable to redistribute work at infinitely small granularities.

AAWS runtimes attempt to improve performance and energy efficiency in both the HP and LP regions by making the work-stealing runtime aware of the underlying static and dynamic asymmetry. In this section, we use first-order modeling and numerical analysis (similar to [HM08, WL08]) to motivate the three techniques we will explore in this chapter: *work-pacing*, which targets the HP region; *work-sprinting*, which focuses on the LP region; and *work-mugging*, which also focuses on the LP region.

3.2.1 First-Order Model

The scripts for our analytical model and a description of their usage are available online[1].

[1] AAWS analytical model scripts: https://github.com/cornell-brg/torng-aaws-scripts-isca2016

Consider a multicore system comprised of N_B big cores and N_L little cores, with N_{BA}, N_{BW}, N_{LA}, N_{LW} big/little active/waiting cores. Assume both big and little cores have the same nominal voltage (V_N) and nominal frequency (f_N), and that the system can individually scale the voltage of each core from V_{min} to V_{max}. Waiting cores can execute the work-stealing loop while "resting" at V_{min} to save power while still enabling correct execution.

We assume that frequency is a linear function of voltage (validated using circuit-level simulation, see Section 3.4). The frequency of each active core is thus:

$$\begin{aligned} f_{Bi} &= k_1 V_{Bi} + k_2 && (i = 1,2,...,N_{BA}) \\ f_{Lj} &= k_1 V_{Lj} + k_2 && (j = 1,2,...,N_{LA}) \end{aligned} \tag{3.1}$$

where k_1, k_2 are fitted parameters, f_{Bi} is the frequency of big core i, V_{Bi} is the voltage of big core i, and so on.

The throughput of an active core is measured in instructions per second (IPS) and is a function of the average instructions per cycle (IPC) of a given core type:

$$\begin{aligned} IPS_{BAi} &= IPC_B \; f_{Bi} && (i = 1,2,...,N_{BA}) \\ IPS_{LAj} &= IPC_L \; f_{Lj} && (j = 1,2,...,N_{LA}) \end{aligned} \tag{3.2}$$

To simplify our discussion we define $\beta = IPC_B / IPC_L$.

We use the aggregate throughput of all active cores as an approximation for the performance of the overall application. If we assume compute-bound tasks and perfect task-balancing through work-stealing in the HP region, then increasing the total throughput will indeed reduce the overall execution time of the HP region. The performance of the LP region is more subtle, since by definition the LP region cannot take advantage of work-stealing until more work is generated. Increasing the throughput of one core at the expense of another core may or may not improve execution time depending on when cores reach the next synchronization point. Fortunately, resting waiting cores in the LP region can generate power slack that can be reallocated to the active cores. This means in practice, we are usually increasing the performance of all active cores in the LP region, and thus using the aggregate throughput can still provide useful insight into how to scale the relative voltages of each core. Given these caveats, we model the total performance of the

multicore system as:

$$IPS_{tot} = \sum_{i=1}^{N_{BA}} IPS_{BAi} + \sum_{j=1}^{N_{LA}} IPS_{LAj} \tag{3.3}$$

The core power includes both dynamic and static power and is modeled as:

$$\begin{aligned} P_{BAi} &= \alpha_B \, IPC_B \, f_{Bi} \, V_{Bi}^2 + V_{Bi} \, I_{B,leak} && (i = 1,2,...,N_{BA}) \\ P_{LAj} &= \alpha_L \, IPC_L \, f_{Lj} \, V_{Lj}^2 + V_{Lj} \, I_{L,leak} && (j = 1,2,...,N_{LA}) \end{aligned} \tag{3.4}$$

The factors α_B and α_L capture the relative energy overhead of a big core compared to a little core. To simplify our discussion, we define $\alpha = \alpha_B/\alpha_L$ as the energy ratio of a big core to a little core at nominal voltage and frequency. We calculate the leakage current by assuming an architect targets leakage power to consume a certain percentage (denoted as λ) of the total power of a big core at nominal voltage. We assume a little core's leakage current is a fraction (denoted by γ) of the big core's leakage current. We use P_{BN}, P_{LN}, P_{BW}, and P_{LW} to refer to the power consumed by big and little cores running at nominal V_N or waiting at V_{min}.

The total power is the aggregate power across all cores:

$$P_{total} = \sum_{i=1}^{N_{BA}} P_{BAi} + \sum_{j=1}^{N_{LA}} P_{LAj} + N_{BW} \, (P_{BW}) + N_{LW} \, (P_{LW}) \tag{3.5}$$

3.2.2 Marginal Utility Optimization Problem

In Figure 3.2, we use our first-order model to generate energy vs. performance estimates for a system with four big and four little cores (denoted as 4B4L). The array of plotted points represent selections of different (V_{Bi}, V_{Lj}) pairs, with all estimates normalized to the nominal system (1.0 V, 1.0 V). We assume that all cores are busy with useful work. Points in the lower-right quadrant generally have higher voltage and frequency, suggesting that performance can easily be improved at the expense of energy efficiency and at higher power. Points in the upper-left quadrant generally have lower voltage and frequency, suggesting that energy efficiency can easily be improved at the expense of performance with lower power. However, points in the upper-right quadrant suggest that careful tuning of the voltages and frequencies of the big and little cores can potentially improve performance *and* energy efficiency at the same time.

Our goal is to create an optimization problem to find the pareto-optimal points in the upper-right quadrant. As a basic heuristic, we use average system power at nominal voltage to constrain our optimization problem, effectively targeting a pareto-optimal system that draws similar power

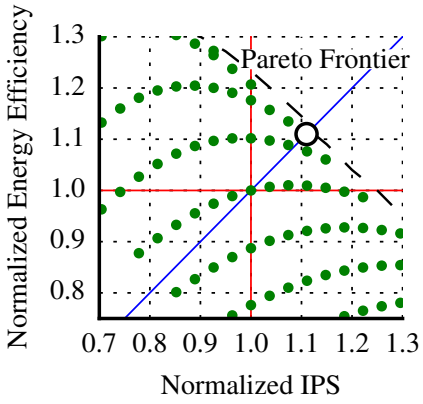


Figure 3.2: Pareto-Optimal Frontier for 4B4L System – Projected energy efficiency vs. performance of a busy 4B4L system across different (V_{Bi}, V_{Lj}) pairs. Points normalized to (1.0 V, 1.0 V) system. Diagonal line is isopower. Open circle = pareto-optimal isopower system. $\alpha = 3$, $\beta = 2$.

compared to the nominal system (denoted by the open circle). The target system power is therefore the aggregate power of all cores running at nominal voltage and frequency:

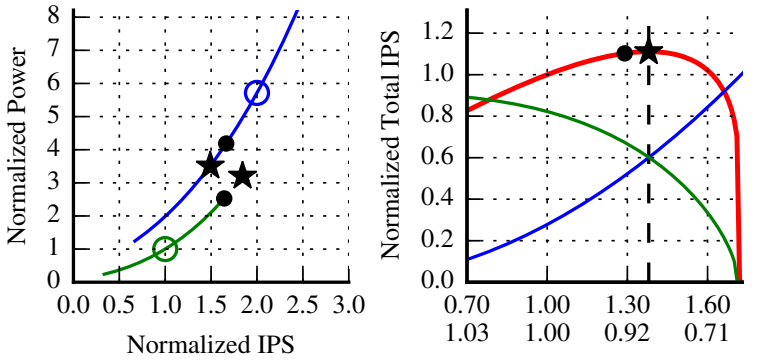
$$P_{target} = N_B\left(P_{BN}\right) + N_L\left(P_{LN}\right) \tag{3.6}$$

More specifically, our optimization problem searches for the optimal voltages for active big (V_{Bi}) and little (V_{Li}) cores such that the total throughput (IPS_{tot}) is maximized while maintaining the power target (P_{target}). We use the method of Lagrange multipliers to solve this optimization problem, and we rewrite the final result in terms of the marginal performance vs. marginal power as follows:

$$\frac{\partial P_{BAi}}{\partial IPS_{BAi}} = \frac{\partial P_{LAj}}{\partial IPS_{LAj}} (i = 1,2,...,N_{BA}; j = 1,2,...,N_{LA}) \tag{3.7}$$

This is an intuitive application of a fundamental principle in economics known as the *Law of Equi-Marginal Utility*. At the optimum operating point the marginal utility (i.e., performance) vs. marginal cost (i.e., power) of each core must be equal. If this was not the case, then an arbitrage opportunity would exist: we could "sell" expensive performance to reclaim power on one core and "buy" more performance at a cheaper price (power) on another core. Others have also recognized that the Law of Equi-Marginal Utility provides an elegant digital design principle [AML+10, WM15], although here we are applying this principle in a slightly different context.

Unfortunately, a closed-form solution for the optimum V_{Bi} and V_{Lj} can be complex, so in the remainder of this section we use numerical analysis to explore using a marginal-utility-based approach in both the HP and LP regions. Unless otherwise noted we will assume the following parameters: $k_1 = 7.38\times10^8$, $k_2 = -4.05\times10^8$, $V_N = 1$ V, $V_{min} = 0.7$ V, $V_{max} = 1.3$ V, $f_N = 333$ MHz, $\lambda = 0.1$, $\gamma = 0.25$. These parameters are derived from VLSI modeling for the target voltage range and system described in Section 3.4.



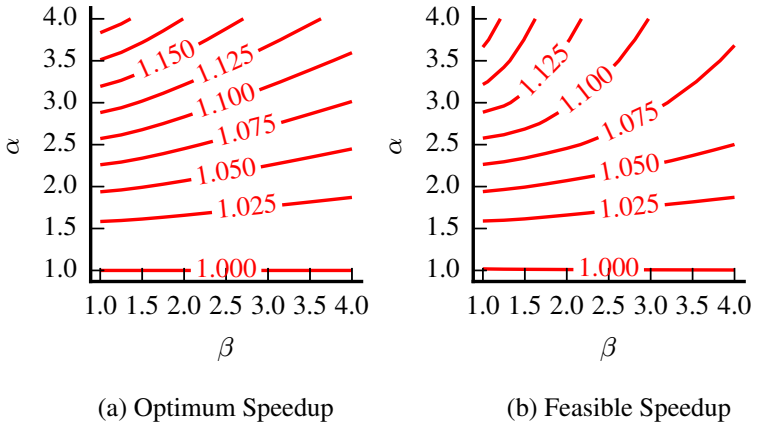
(a) Power vs. IPS (b) Total IPS vs. V_{Bi} and V_{Lj}

Figure 3.3: 4B4L System w/ All Cores Active – (a) Power vs. performance curves across the DVFS operating points for each core type, green = little, blue = big, circle = nominal; (b) blue = $\partial P_{BAi}/\partial IPS_{BAi}$ (axis not shown), green = $\partial P_{LAj}/\partial IPS_{LAj}$ (axis not shown), red = IPS_{tot} (axis on left) assuming V_{Lj} and V_{Bi} shown on x-axis (V_{Lj} on top, V_{Bi} on bottom) with constant P_{target}. (a–b) star = optimal operating point, dot = feasible operating point, $\alpha = 3$, $\beta = 2$.

3.2.3 Marginal Utility in the High-Parallel Region

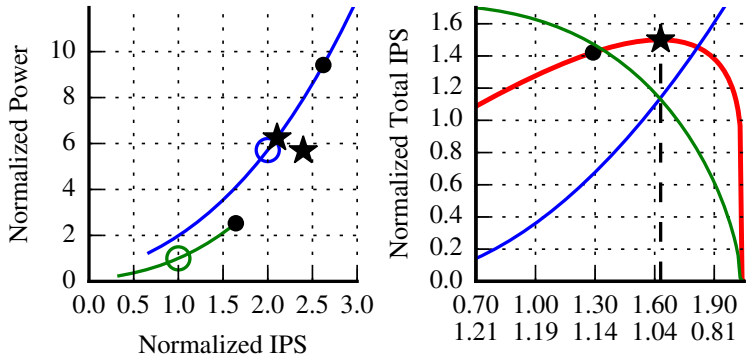
Figure 3.3 uses the first-order model developed in the previous subsections to plot the power and performance of a 4B4L system. This is a common configuration found in commercially available ARM big.LITTLE systems [Gwe14b, Gwe14a]. We can immediately see the benefit of static asymmetry in Figure 3.3(a). The big core offers higher performance at higher power, while the little core offers lower performance at lower power. Figure 3.3(b) shows the marginal utility of the big core (blue curve) and little core (green curve) as well as IPS_{tot}. As expected, IPS_{tot} is maximized when the marginal utilities are equal. The optimal operating point is $V_{Bi} = 0.86\,\text{V}$ and $V_{Li} = 1.44\,\text{V}$ with a theoretical speedup of $1.12\times$ over running all cores at V_N. Since $1.44\,\text{V} > V_{max}$, the best feasible operating point is $V_{Bi} = 0.93\,\text{V}$ and $V_{Li} = V_{max}$ with a theoretical speedup of $1.10\times$. Figure 3.4 shows how the optimal and feasible speedup varies as a function of α and β. A marginal-utility-based approach is most effective when the big core has moderate performance benefit for large energy overhead (i.e., $\alpha/\beta > 1.0$), which matches the conventional wisdom concerning big vs. little cores. This wisdom is supported by data collected by ARM during pre-silicon design-space exploration of Cortex-A7 and Cortex-A15 cores [Gre11] as well as by our own results (see Section 3.5).

This analysis suggests a marginal-utility-based approach can offer respectable speedups in the HP region, and thus motivates our interest in our new work-pacing technique. It is important to note, that a marginal-utility-based approach requires holistically considering static asymmetry, dynamic asymmetry, *and* a work-stealing runtime. With a thread-based parallel programming



(a) Optimum Speedup (b) Feasible Speedup

Figure 3.4: Theoretical Speedup for 4B4L System vs. α and β – (a) optimum speedup ignoring V_{min} and V_{max}; (b) = feasible speedup within V_{min} and V_{max}. Speedups relative to all cores running at V_N.



(a) Power vs. IPS (b) Total IPS vs. V_{Bi} and V_{Lj}

Figure 3.5: 4B4L System w/ 2B2L Active – Assume we rest inactive cores at V_{min}. See Figure 3.3 for legend. $\alpha = 3$, $\beta = 2$.

framework instead of work-stealing, slowing down the big core would likely create a significantly lagging thread hurting the overall execution time. Without static asymmetry, the Law of Equi-Marginal Utility tells us that the optimal approach is to simply run all homogeneous cores at V_N during the HP region. Without dynamic asymmetry, there is no opportunity to adaptively "trade" performance vs. power and thus no way to balance the marginal utilities in the HP region.

3.2.4 Marginal Utility in the Low-Parallel Region

Figure 3.5 plots the power and performance of a 4B4L system in the LP region with two active big cores and two active little cores. We can rest the waiting cores, generating power slack that can then be reallocated to the active cores. The resulting optimal operating point is $V_{Bi} = 1.02\,\text{V}$ and $V_{Li} = 1.70\,\text{V}$ with a theoretical speedup of $1.55\times$ over running all cores at V_N. Obviously running

the little core at 1.70 V is not feasible, so the best feasible operating point is $V_{Bi} = 1.16$ V and $V_{Li} = V_{max}$ with a theoretical speedup of 1.45×.

Note that we can potentially further improve performance by moving tasks from little to big cores. As shown in Figure 3.5(a) the little core often reaches V_{max} before it can completely exploit the power slack generated from resting cores. Moving tasks to a big core increases peak performance and thus can help accelerate LP regions. As an example, assume there is a single remaining task in a 4B4L system and we must decide whether to execute this task on a little or big core. Our first-order model predicts that the optimum operating point when using a little core is $V_L = 2.59$ V, but the feasible operating point is V_{max} with a theoretical speedup of 1.6× over running this task on the little core at V_N. If we instead move this final task to a big core, the optimum operating point is $V_B = 1.51$ V, and the feasible operating point is again V_{max} with a theoretical speedup of 3.3× over running this task on the little core at V_N. Moving work from little to big cores in the LP range can significantly improve performance if we take into account the feasible voltage range.

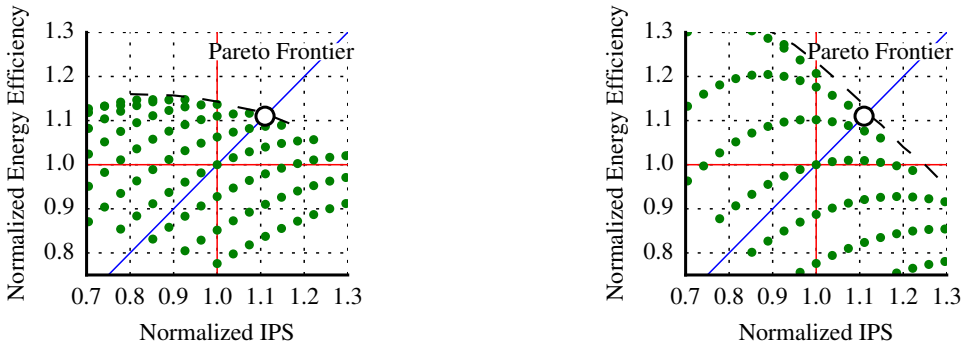
This analysis suggests that a marginal-utility-based approach can be useful in the LP region, but our analysis also motivates our interest in a practical implementation of work-mugging. Work-mugging can preemptively move work from little to big cores, and thus helps keep big cores busy during the LP region. Again a holistic approach is required: using just dynamic or static asymmetry during the LP region is unlikely to fully exploit the generated power slack.

3.2.5 Leakage Sensitivity Study

The analytical modeling used in this section has assumed that the big core consumed 10% static leakage power (i.e., $\lambda = 0.1$) at nominal voltage and frequency (refer to the parameters listed at the end of Section 3.2.2). In this subsection, we discuss and sweep the leakage parameter λ in order to analyze the first-order impact of leakage on opportunities for balancing marginal utility in a 4B4L system. The results show that the high-level conclusions of this section remain the same across a wide range of leakage power assumptions.

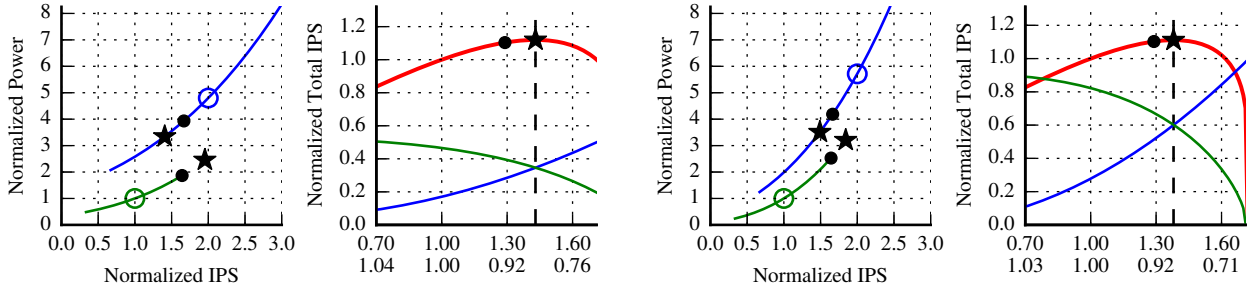
Comparison with High and Low Leakage – We split each of Figures 3.2 to 3.5 into two versions, one with higher leakage (i.e., $\lambda = 0.5$) and one with lower leakage (i.e., $\lambda = 0.1$).

Compare the two plots in Figure 3.6 (versions of Figure 3.2). To first order, the system on the left with higher leakage tends to achieve lower energy efficiency across all points, where each point corresponds to a different (V_{Bi}, V_{Lj}) pair. This result is not surprising because leakage power scales



(a) $\lambda = 0.5$ (b) $\lambda = 0.1$

Figure 3.6: Leakage Comparison for Pareto-Optimal Frontier of 4B4L System – Respins of Figure 3.2 with different leakage parameters. (a) high leakage with $\lambda = 0.5$, (b) low leakage with $\lambda = 0.1$. The plot shows the projected energy efficiency vs. performance of a busy 4B4L system across different (V_{Bi}, V_{Lj}) pairs. Points normalized to (1.0 V, 1.0 V) system. Diagonal line is isopower. Open circle = pareto-optimal isopower system. $\alpha = 3$, $\beta = 2$.



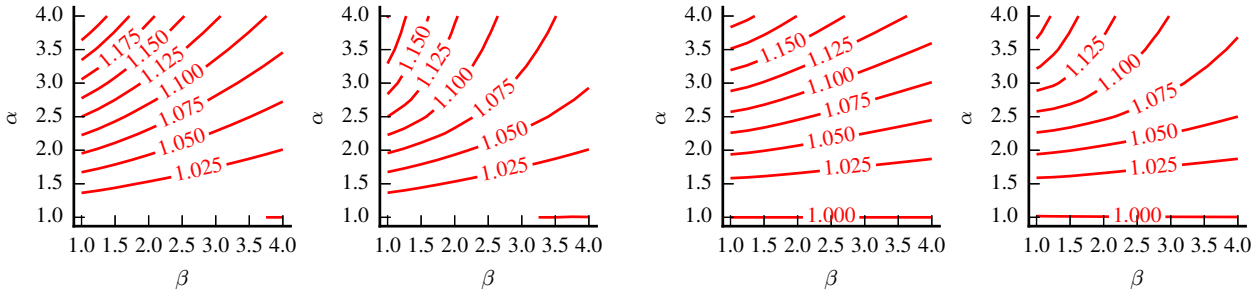
(I) Power vs. IPS (II) Total IPS vs. V_{Bi} (V_{Lj}) (I) Power vs. IPS (II) Total IPS vs. V_{Bi} (V_{Lj})

(a) $\lambda = 0.5$ (b) $\lambda = 0.1$

Figure 3.7: Leakage Comparison for 4B4L System w/ All Cores Active – Respins of Figure 3.3 with different leakage parameters. (a) high leakage with $\lambda = 0.5$, (b) low leakage with $\lambda = 0.1$. For both cases, (I) Power vs. performance curves across the DVFS operating points for each core type, green = little, blue = big, circle = nominal; (II) blue = $\partial P_{BAi}/\partial IPS_{BAi}$ (axis not shown), green = $\partial P_{LAj}/\partial IPS_{LAj}$ (axis not shown), red = IPS_{tot} (axis on left) assuming V_{Lj} and V_{Bi} shown on x-axis (V_{Lj} on top, V_{Bi} on bottom) with constant P_{target}. (I–II) star = optimal operating point, dot = feasible operating point, $\alpha = 3$, $\beta = 2$.

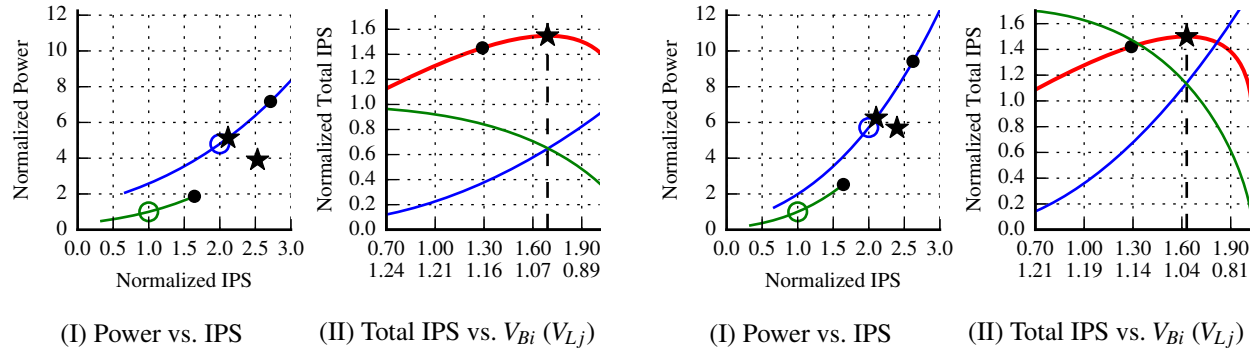
linearly with voltage, while dynamic power scales cubically with voltage. Note that the pareto-optimal isopower point denoted by the open circle does not change, implying that the potential benefit of exploiting marginal utility imbalance using the AAWS techniques is likely to be similar for systems with either lower or higher leakage.

Compare Figure 3.7(a) and Figure 3.7(b) (versions of Figure 3.3) as well as Figure 3.9(a) and Figure 3.9(b) (versions of Figure 3.5). We can see how leakage impacts the slope of the power versus performance curves across the DVFS operating range for each core type. Lower leakage results in steeper slopes. Again, this is not surprising due to the scaling of leakage power and dynamic power. In lower leakage systems, dynamic power dominates, which results in a greater



(a) $\lambda = 0.5$ (b) $\lambda = 0.1$

Figure 3.8: Leakage Comparison for Theoretical Speedup for 4B4L System vs. α and β – Respins of Figure 3.4 with different leakage parameters. (a) high leakage with $\lambda = 0.5$, (b) low leakage with $\lambda = 0.1$. For both cases, (I) optimum speedup ignoring V_{min} and V_{max}; (II) = feasible speedup within V_{min} and V_{max}. Speedups relative to all cores running at V_N.

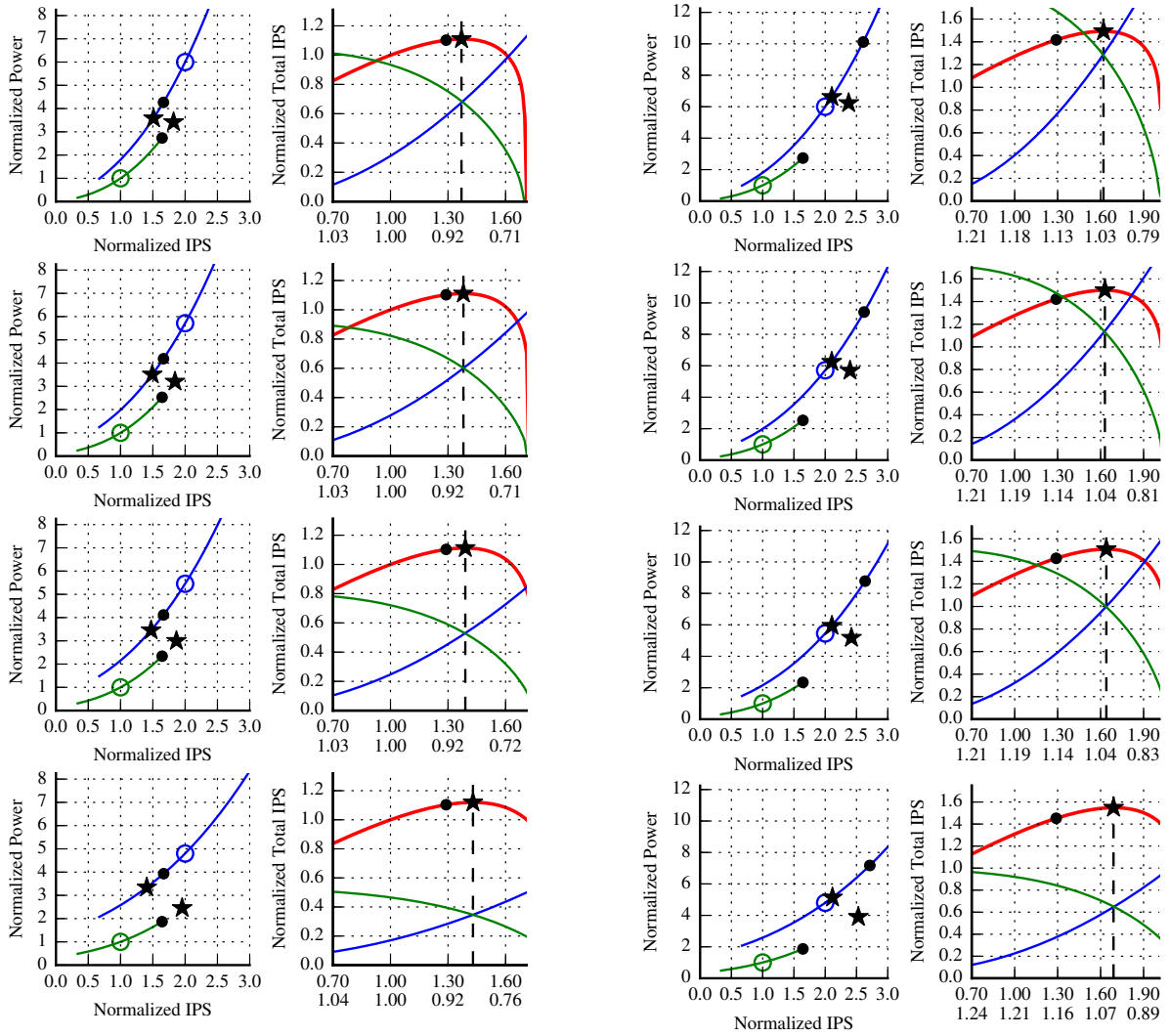


(a) $\lambda = 0.5$ (b) $\lambda = 0.1$

Figure 3.9: Leakage Comparison for 4B4L System w/ 2B2L Active – Respins of Figure 3.5 with different leakage parameters. (a) high leakage with $\lambda = 0.5$, (b) low leakage with $\lambda = 0.1$. For both cases, assume we rest inactive cores at V_{min}. See Figure 3.5 for legend. $\alpha = 3$, $\beta = 2$.

relative impact on system power when scaling voltage. In higher leakage systems, static power is more significant, resulting in a shallower curve. In both Figure 3.7 and Figure 3.9, note that the highest throughput does not change significantly and still occurs when the marginal utility curves intersect (i.e., when the marginal utilities are equal). The high-level conclusions of this section are therefore the same for both low-leakage and high-leakage systems.

Finally, compare Figure 3.8(a) and Figure 3.8(b) (versions of Figure 3.8). We see that the theoretical speedups do not vary significantly with leakage. Recall that the analytical modeling thus far has assumed $\beta = 2$ and $\alpha = 3$. For both higher and lower leakage, the theoretical optimum speedup is roughly the same (i.e., between 1.100 and 1.125) for the 4B4L system. Similarly, for both higher and lower leakage, the feasible speedup within V_{min} and V_{max} is roughly the same (i.e., about 1.100).



(a) 4B4L System w/ All Cores Active

(b) 4B4L System w/ 2B2L Active

Figure 3.10: Leakage Parameter Sweeps – Each row of plots is identical to Figures 3.3 and 3.5, but with a different value of λ. The first row sets $\lambda = 0.0$; second row sets $\lambda = 0.1$; third row sets $\lambda = 0.2$; fourth row sets $\lambda = 0.5$.

Leakage Sweep – We now sweep the leakage parameter λ in order to analyze the first-order impact of leakage on opportunities for balancing marginal utility in a 4B4L system. Figure 3.10(a) and Figure 3.10(b) are new plots that do not appear in the previous subsections. Each row in Figure 3.10(a) corresponds to Figure 3.3 with a specific value of λ, and similarly, each row in Figure 3.10(b) corresponds to Figure 3.5 with a specific value of λ. Starting from the first row with zero leakage and moving down to the fourth row with 50% leakage, we see that the power versus performance curves slowly become shallower for both systems, just as we saw earlier.

In both Figure 3.10(a) and (b), notice that the optimal operating point (denoted by the star) occurs at slightly higher V_L as leakage increases. For example, the optimum V_L in Figure 3.10(b)

with zero leakage is 1.62 V, but with $\lambda = 0.5$, the optimum V_L is 1.69 V. With higher leakage, the little cores must travel further along the power versus performance curve to balance marginal utility with the big core. Notice that the *feasible* operating point does not change, since the little core quickly hits V_{max} in both figures. Finally, notice that the highest throughput still occurs where the marginal utility curves intersect.

Conclusions – We presented side-by-side comparisons for different leakage values with the AAWS analytical model. We also briefly surveyed the impact of leakage by sweeping λ. We find that the high-level conclusions of this section remain consistent across a wide range of leakage power assumptions.

3.3 AAWS Runtimes

In this section, we describe how AAWS runtimes can use three new hardware/software techniques: *work-pacing*, *work-sprinting*, and *work-mugging*. We also describe two simpler techniques, *serial-sprinting* and *work-biasing*, which we include in our aggressive baseline runtime.

3.3.1 Work-Pacing and Work-Sprinting

Work-pacing uses a marginal-utility-based approach to maximize throughput in the HP region by increasing the voltage of little cores and decreasing the voltage of big cores. Work-sprinting combines the power slack generated from resting waiting cores in the LP region with a marginal-utility-based approach to again maximize throughput. An efficient implementation of work-pacing and work-sprinting requires lightweight changes to both the hardware and software.

Work-pacing and work-sprinting require the AAWS software runtime to inform the hardware of when threads are either actively executing tasks or waiting in the work-stealing loop. We propose instrumenting the work-stealing loop in the AAWS runtime with hint instructions, such that each hint instruction toggles an activity bit indicating the status of each core. This is similar to past work on lightweight DVFS controllers for applications that use thread-based parallel programming frameworks [MPT+12, GTB+14]. When a worker thread enters the work-stealing loop, it will wait until its second steal attempt before using the hint instruction to toggle the activity bit. When combined with occupancy-based victim selection [CM08] as opposed to random victim

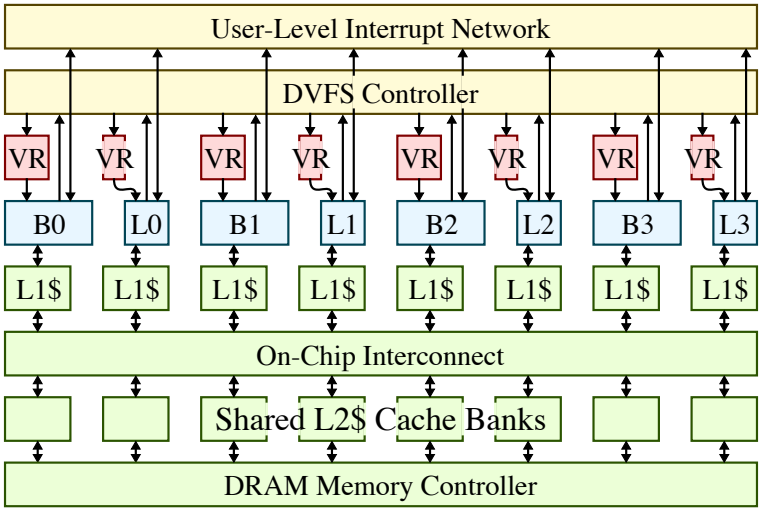


Figure 3.11: 4B4L System w/ Hardware Support for AAWS Runtimes – Work-pacing and work-sprinting require fully integrated voltage regulators for fine-grain DVFS (i.e., per-core, sub-microsecond scaling to potentially many voltage levels) as well as a customized DVFS controller. Work-mugging requires user-level interrupts to enable rapid communication between cores. B0–B3 = big cores, L0–L3 = little cores, VR = fully integrated voltage regulator.

selection [FLR98], this avoids unnecessary activity bit transitions that could adversely impact the customized DVFS controller described later in this subsection. Although this elegant approach is reactive, it also avoids the need for prediction heuristics [CGR+08, BM09].

As shown in Figure 3.1, HP and LP region timescales can be on the order of a few microseconds. Unfortunately, traditional off-chip switching regulators can have voltage scaling response times on the order of tens to hundreds of microseconds [BPSB00, PSCP10]. Multi-rail voltage supplies offer very fast response times on the order of tens of nanoseconds [MPT+12, Dre11], but they offer limited voltage levels making them a poor fit for the many voltage levels required by our marginal-utility-based approach. We propose leveraging recent circuit-level research on fully-integrated switching regulators [LSA11,LCSA13,KBW12]. Architects are increasingly making the case for integrated voltage regulation as a way to reduce system cost and enable fine-grain DVFS for per-core, sub-microsecond scaling to potentially many voltage levels [JLB+15, KGWB08, GTB+14, YLH+12, GSSK12]. Indeed, the Intel Haswell processor uses in-package inductors with on-chip regulators to provide fast-changing per-core supplies [Kan13].

Figure 3.11 illustrates a 4B4L system that can potentially leverage an AAWS runtime. Each core has its own private L1 instruction and data cache along with a shared, banked L2 cache. The activity bits controlled by the hint instructions are read by a global DVFS controller, which then decides how to set the supply voltages of fully integrated per-core voltage regulators. While

distributed DVFS controllers are certainly possible [WM15], the added complexity is probably not warranted for smaller-scale systems. We propose using a simple lookup-table-based DVFS controller to map activity information into appropriate voltage levels. The marginal-utility-based analysis from the previous section along with estimated values for α and β can be used to create a lookup table that performs generally well across a wide range of application kernels. For a 4B4L system, there are five possible values for the number of active little (big) cores, including zero. The lookup table would therefore include 25 entries. More sophisticated adaptive algorithms that update the lookup tables based on performance and energy counters are possible and an interesting direction for future work.

3.3.2 Work-Mugging

The goal of work-mugging is to preemptively migrate work from little cores to big cores during the LP region. Our first-order modeling argues for using the big cores when possible during the LP region, since bigger cores have a higher feasible performance limit. Previous theoretical work [BR02] and abstract discrete-event modeling [JB02] have made the case for work-mugging, although these past works did not propose a concrete way to actually implement work-mugging. As with work-pacing and work-sprinting, an efficient implementation of work-mugging requires lightweight changes to both the hardware and software.

We propose using fast user-level interrupts to enable one core to redirect the control flow of another core. These user-level interrupts are essentially a simple implementation of previous ideas on asynchronous direct messages [SYK10] and active messages [vECGS92]. A "mugger" can use a new `mug` instruction to mug a "muggee". The `mug` instruction takes two input values stored in registers. One indicates which core to mug and the other contains the address of the user-level interrupt handler. Other than the handler address, all data is communicated through shared memory. This form of user-level interrupts requires a simple, low-bandwidth inter-core network with approximately four-byte messages (see Figure 3.11).

We modify the AAWS runtime so that when a worker thread running on a big core enters the work-stealing loop, it will attempt to steal work twice before considering work-mugging. If at least one little core is active, then the big core selects a little core to mug using the `mug` instruction. Each core's runtime keeps information about that core's activity and task queue occupancy in shared memory, so that every core can easily determine which cores are active and which cores

are waiting. The mugger and muggee store their user-level architectural state to shared memory, synchronize at a barrier, and then load the other thread's user-level architectural state. The overall effect is that a big core begins executing a task which was previously executing on a little core, while the little core enters the work-stealing loop. A subtle yet critical implementation detail involves ensuring that a big core always executes the sequential part of the program after the parallel region (i.e., logical thread 0). Otherwise, a little core can end up executing the serial region. We modify the AAWS runtime so at the end of a parallel region, logical thread 0 checks to see if it is running on a big core. If not, it simply uses a `mug` instruction to mug any big core.

While work-mugging can help preemptively move work from little to big cores, it can also cause additional L1 cache misses as the task's working set is gradually migrated. This overhead is somewhat mitigated by the good locality properties of a work-stealing runtime and by the fact that little cores do not mug work back from a big core during the LP region.

3.3.3 Serial-Sprinting and Work-Biasing

Instruction hints can also be used to inform the hardware of truly serial regions (as opposed to an LP region which just happens to have one task remaining). An obvious extension is to allow the DVFS controller to sprint the single active big core during these serial regions. Since this is a relatively straight-forward optimization, we include *serial-sprinting* in our baseline work-stealing runtime. Our application workloads have relatively short serial regions. So while serial-sprinting does not hurt performance, it also does not offer much benefit (approximately 1–2%).

A simple non-preemptive scheme we call *work-biasing* involves preventing little cores from stealing work unless all big cores are active. Work-biasing has the effect of "biasing" work towards the big cores, but the non-preemptive nature of work-biasing means there are few opportunities to have a meaningful impact on the work distribution. Indeed, the prior theoretical work on work-mugging also suggests the importance of preemption [BR02]. Even so, work-biasing can sometimes have a small benefit (approximately 1%) and never hurts performance, so we include work-biasing in our baseline work-stealing runtime.

We choose to include serial-sprinting and work-biasing in our baseline runtime to ensure that it is as aggressive as possible, but this does mean our baseline runtime includes a very limited form of asymmetry awareness.

Technology	TSMC 65nm LP, 1.0V nominal voltage
ALU	4/10-cycle Int Mul/Div, 6/6-cycle FP Mul/Div, 4/4-cycle FP Add/Sub
Little Core	1-way, 5-stage in-order, 32 Phys Regs, 333MHz nominal frequency
Big Core	4-way, out-of-order, 128 Phys Regs, 32 Entries IQ and LSQ, 96 Entries ROB, Tournament Branch Pred, 333MHz nominal frequency
Caches	1-cycle, 1-way, 16KB L1I, 1-cycle 2-way 16KB L1D per core; 20-cycle, 8-way, shared 1MB L2; MESI protocol
OCN	Crossbar topology, 2-cycle fixed latency
DRAM	200ns fixed latency, 12.8GB/s bandwidth SimpleMemory model

Table 3.1: Cycle-Level System Configuration

3.4 Evaluation Methodology

We use a vertically integrated research methodology spanning software, architecture, and VLSI. In this section, we describe our target system, applications, baseline runtime, cycle-level performance modeling, and energy modeling.

3.4.1 Target System

Although much of our analysis is applicable to larger high-performance systems, we focus on the smaller embedded systems that are already integrating static and dynamic asymmetry [Gwe14b, Gwe14a, Dem13, Gwe13]. Table 3.1 includes details on the core microarchitectures and memory system. We study two eight-core configurations: a 4B4L system with four big and four little cores similar to commercial products [Gwe14b, Gwe14a] and a 1B7L system with one big core and seven little cores. We target a 32-bit RISC architecture with 32 general-purpose registers and hardware floating point. We specifically target an older technology (TSMC 65nm LP) and a lower-frequency, single-issue, in-order little core for two reasons. First, we have our own VLSI implementation of such a core and thus can pursue a more rigorous energy-modeling strategy (see Section 3.4.5). Second, we have access to SPICE-level models of integrated voltage regulators in this technology that help us to accurately estimate DVFS transition overheads. We expect the high-level conclusions of our work to hold for high-performance target systems.

3.4.2 Benchmark Suite

We have ported 20 C++ application kernels to our RISC architecture and work-stealing runtime. These kernels are selected from the PBBS (v. 0.1) [SBF+12], PARSEC (v. 3.0) [BKSL08], Cilk (v. 5.4.6) [FLR98], and UTS (v. 2.1) [OHL+06] benchmark suites and represent diverse application-level characteristics (see Table 3.3). We include two datasets for *qsort* and *radix*, since they exhibit strong data-dependent variability. *bfs-d* and *bfs-nd* capture the impact of deterministic execution for the same problem. Determinism is a desirable trait that can mitigate the difficulties of reasoning about both correctness and performance in complex systems [SBF+12]. We list MPKI for each app, showing that our application kernels are fairly compute-bound. We select applications with varied parallelization methods. In addition to the conventional `parallel_for` construct, our selections exhibit recursive spawn-and-sync parallelization as well as nested loop parallelism (*sampsort* and *uts*). Most PBBS benchmarks parallelize execution with reserve-and-commit phases to create determinism. For *sptree* and *mis*, we choose the non-deterministic versions which use atomic memory operations to synchronize work. When running on our target systems, our application kernels achieve respectable parallel speedup and yet vary widely in the number of tasks and sizes of tasks. For detailed kernel descriptions, see [SBF+12, BKSL08, FLR98, OHL+06].

3.4.3 Work-Stealing Runtime

Work-stealing runtimes can be divided into several categories (e.g., language-based vs. library-based, child-stealing vs. continuation-stealing). While we believe that our approach can be applied to various categories, in this work we choose to focus on a C++ library-based implementation similar in spirit to Intel TBB. We support syntax similar to TBB's `parallel_for` and `parallel_invoke`. Our runtime uses child-stealing and supports automatic recursive decomposition of parallel loops (similar to Intel TBB's `simple_partitioner`). We use non-blocking, dynamically sized Chase-Lev task queues [CL05] and occupancy-based victim selection [CM08]. We have carefully optimized our runtime to minimize memory fences, atomic memory operations, and false sharing.

We have compared the performance of our baseline runtime to Intel Cilk++ and Intel TBB on five application kernels from PBBS running natively on an eight-core Intel x86 platform. We use large datasets and many trials so that it takes ≈30 seconds to run one serial application. The

	Cilk++	TBB	Baseline	Baseline vs. TBB
dict	4.02	5.02	5.53	+10%
radix	7.05	4.87	5.58	+14%
rdups	3.96	4.36	4.54	+4%
mis	2.75	2.42	2.40	-1%
nbody	7.37	7.10	6.95	-3%

Table 3.2: Performance of Baseline Runtime vs. Intel Cilk++ and Intel TBB on Real System – Numbers are speedups vs. scalar implementation. Cilk++ = original Cilk implementation of PBBS apps compiled with Intel C++ Compiler 14.0.2. TBB = ported PBBS apps using `parallel_for` with Intel TBB 4.4 build 20150928. Baseline = ported PBBS apps using `parallel_for` with our baseline work-stealing runtime. Each configuration uses eight threads running on an unloaded Linux server with two Intel Xeon E5620 processors.

									Speedup					
				DInst	Num	Task	Opt IO			1B7L		4B4L		L2
Name	Suite	Input	PM	(M)	Tasks	Size (K)	Cyc (M)	ERatio	O3	vs O3	vs IO	vs O3	vs IO	MPKI
bfs-d	pbbs	`randLocalGraph_J_5_150K`	p	36.0	2588	14	113.2	2.8	2.2	2.3	5.1	2.9	6.5	14.8
bfs-nd	pbbs	`randLocalGraph_J_5_150K`	p	58.1	3108	19	113.2	2.8	2.2	1.8	4.0	2.4	5.3	12.3
qsort-1	pbbs	`exptSeq_10K_double`	rss	18.8	777	24	26.1	2.5	1.7	2.8	4.7	3.2	5.4	0.0
qsort-2	pbbs	`trigramSeq_50K`	rss	20.0	3187	6	38.9	3.1	1.9	3.3	6.3	4.6	8.7	0.0
sampsort	pbbs	`exptSeq_10K_double`	np	37.5	15522	2	26.1	2.5	1.7	2.5	4.2	3.0	5.1	0.11
dict	pbbs	`exptSeq_1M_int`	p	45.1	256	151	101.5	2.8	1.7	4.0	6.9	5.1	8.8	7.0
hull	pbbs	`2Dkuzmin_100000`	rss	14.2	882	16	31.6	2.1	2.2	3.4	7.5	4.4	9.8	6.0
radix-1	pbbs	`randomSeq_400K_int`	p	42.4	176	240	83.1	2.2	1.8	2.7	4.7	3.1	5.5	7.7
radix-2	pbbs	`exptSeq_250K_int`	p	35.1	285	123	56.6	2.1	1.8	2.8	4.9	3.1	5.5	7.5
knn	pbbs	`2DinCube_5000`	p, rss	83.3	3499	23	139.3	2.8	1.7	6.0	9.9	7.0	11.5	0.02
mis	pbbs	`randLocalGraph_J_5_50000`	p	5.8	3230	2	11.6	3.6	2.3	3.8	9.0	4.3	10.1	3.5
nbody	pbbs	`3DinCube_180`	p, rss	56.6	485	116	75.1	2.9	1.6	5.6	8.7	7.1	11.1	0.01
rdups	pbbs	`trigramSeq_300K_pair_int`	p	51.2	288	156	108.4	2.6	1.7	3.5	5.9	4.2	7.1	7.6
sarray	pbbs	`trigramString_120K`	p	42.1	2434	16	114.7	2.5	2.3	2.6	6.0	2.9	6.8	10.0
sptree	pbbs	`randLocalGraph_E_5_100K`	p	18.9	482	39	57.2	2.8	2.1	3.0	6.3	3.5	7.3	4.9
clsky	cilk	`-n 128 -z 256`	rss	42.0	3645	11	70.4	2.4	1.7	5.1	8.6	6.2	10.5	0.02
cilksort	cilk	`-n 300000`	rss	47.0	2056	22	76.2	3.7	1.3	5.7	7.3	6.3	8.1	2.3
heat	cilk	`-g 1 -nx 256 -ny 64 -nt 1`	rss	54.3	765	54	64.9	2.3	2.1	4.2	8.8	5.7	11.7	0.04
ksack	cilk	`knapsack-small-1.input`	rss	30.1	78799	0.3	25.9	2.4	1.9	2.3	4.3	2.7	5.0	0.0
matmul	cilk	`200`	rss	68.2	2047	33	118.8	2.0	3.6	2.7	10.0	4.8	17.4	0.0
bscholes	parsec	`1024 options`	p	40.3	64	629	52.7	2.4	1.9	4.2	7.9	5.5	10.4	0.0
uts	uts	`-t 1 -a 2 -d 3 -b 6 -r 502`	np	63.9	1287	50	82.6	2.3	2.0	4.4	8.8	5.8	11.6	0.02

Table 3.3: Application Kernels – Suite = benchmark suite. Input = input dataset & options. PM = parallelization methods: p = `parallel_for`, np = nested `parallel_for`, rss = recursive spawn-and-sync. DInsts = dynamic instruction count in millions. Num Tasks = number of tasks. Task Size = average task size in thousands of instructions. Opt IO Cyc = number of cycles of an optimized *serial* implementation on an in-order core. ERatio = energy ratio of the serial implementation on O3 over IO (i.e., α in Section 3.2.1). O3 = speedup of the serial implementation on O3 over IO (i.e., β in Section 3.2.1). 1B7L = speedup on one big and seven little cores. 4B4L = speedup on four big and four little cores. L2 MPKI = L2 misses per one thousand instructions with the parallelized implementation and baseline runtime on one core.

speedup results over optimized serial implementations are shown in Table 3.2. Our runtime has similar performance to Intel TBB and is sometimes slightly faster due to the fact that our runtime is lighter weight and does not include advanced features like C++ exceptions or cancellations from within tasks. Section 3.5 uses cycle-level simulation to show that our baseline runtime achieves very reasonable speedups on both 4B4L and 1B7L systems. These real-system- and simulation-based results provide compelling evidence for the strength of our baseline runtime. As mentioned in Section 3.3.3, we also add serial-sprinting and work-biasing (limited forms of asymmetry awareness) to our baseline runtime to ensure it is as aggressive as possible. Our AAWS runtime extends this baseline runtime as described in Sections 3.3.1 and 3.3.2.

3.4.4 Cycle-Level Performance Modeling

We use the gem5 simulator [BBB+11] in syscall emulation mode for cycle-level performance modeling of our target systems. Heterogeneous systems are modeled by combining modified `O3CPU` and `InOrderCPU` models. We modified gem5 to toggle an activity bit in each core after executing the hint instructions. We modified gem5's clock domains and clocked object tick calculations to support dynamic frequency scaling. Cores can independently scale their frequency, but we centralized control of all clock domains in a DVFS controller specialized for AAWS. As described in Section 3.3.1, we model a lookup-based DVFS controller that maps activity information into appropriate voltage levels. We use tables similar to those described in FG-SYNC+ [GTB+14]. We slightly modify this approach by separating little-core activity bits from big-core activity bits. The number of active little cores and active big cores then maps to appropriate voltage levels according to the marginal utility model.

We use SPICE-level models of integrated voltage regulators in this technology to accurately estimate DVFS mode transition overheads. The transition time from 0.7 V to 1.33 V is roughly 160 ns with a sophisticated scheme as described in [GTB+14]. We model transition overheads linearly with 40 ns per 0.15 V step in gem5 to capture the general trend, and we include this overhead in our results. However, transitions happen infrequently with an average of 0.2 transitions per ten microseconds across our benchmarks (maximum of 0.7). We ran a sensitivity study sweeping transition overhead to 250 ns per step and saw less than 2% overall performance impact. Throughout our experiments, we assume that cores can continue executing through the voltage transition

at the lower frequency, and we also assume that new decisions cannot be made until the previous transition completes.

We added support for the new `mug` instruction which can cause one core to initiate an interrupt on another core. Specific overheads are difficult to isolate but are modeled in our simulators: pipeline-flush overhead is captured through the gem5 interrupt-handling mechanism; register state (32 GPRs, exception cause and EPC, thread-ID reg) swapping is done via memory in the exception handler and captured through gem5 modeling of cache coherence and misses; instruction/data cache migration overhead is captured the same way; we estimate the inter-core interrupt latency to be on the order of an L2 access and thus add an explicit 20-cycle latency per-mug. Because we use multithreaded workloads, we do not model TLB invalidations. Thread-swapping assembly includes about 80 instructions of mugging assembly code. We observe that work-mugging happens infrequently (less than 40 per million instructions) and that performance is generally insensitive to work-mugging overheads. We ran a sensitivity study sweeping the interrupt latency to 1000 cycles and saw less than 1% overall performance impact.

3.4.5 Energy Modeling

To help estimate energy of a little core, we developed a realistic RTL implementation of an in-order, single-issue scalar core and L1 memory system. The RTL model is synthesized and placed-and-routed using a combination of Synopsys Design Compiler, IC Compiler, and PrimeTime PX with a TSMC 65 nm LP standard-cell library characterized at 1 V. Analysis of the placed-and-routed design indicates each core is approximately 0.75 mm^2 and can run at 333 MHz at 1 V. We predict that more aggressive RTL and circuit design could increase this clock frequency by 2× or more. We then ran a suite of 65 *energy microbenchmarks* that are each designed to measure the energy used by various components in the core for each instruction. For example, the `addiu` energy microbenchmark warms up the instruction cache (to isolate the energy solely due to the instruction under test) and then executes 100 `addiu` instructions in sequence. We ran this microbenchmark on the synthesized gate-level design to obtain bit-accurate traces that are fed into PrimeTime power simulations for power estimates. Coupled with the cycle time of the placed-and-routed design, we can calculate the energy per unit used by the `addiu` instruction.

We experimented with a purely McPAT-based energy model, but we had difficulty validating the McPAT in-order energy model against our own VLSI results. Since we do not have access

to RTL for an aggressive superscalar out-of-order processor, we used a hybrid approach. We run the same energy microbenchmarks mentioned above on gem5's in-order model to generate event counts (e.g., register file reads, instruction cache accesses, integer ALU accesses). We then use our VLSI implementation to carefully calculate the energy for each event. We have developed a simple energy modeling tool which takes the event counts and our own component-level energy numbers and produces a total energy estimate. We iterate to ensure that the overall energy of each energy microbenchmark correlates between our VLSI implementation and our energy model. We then use McPAT's component-level models to estimate the energy of various structures within an out-of-order core but not within an in-order core. Since the absolute energy numbers from McPAT and our VLSI implementation do not necessarily match, we use the energy of a component that is present both in our VLSI implementation and McPAT (e.g., the integer ALU or register file read access) to normalize the McPAT component models. Finally, we account for pipeline register overhead and leakage power. We then run our energy microbenchmarks on our VLSI implementation, the gem5 in-order model, and the gem5 out-of-order model, and generate detailed energy breakdowns for every component. We carefully compare all breakdowns for each microbenchmark to ensure that our energy model matches our intuition.

SPICE-level simulations were used to determine the relationship between frequency and voltage for our cores across different operating modes. We used nine delay stages consisting of multiple FO4 loaded inverters, NAND, and NOR gates connected in a loop, such that the total delay in the loop matches our RTL cycle time for a given voltage. We used the change in delay vs. supply voltage as a model for core voltage-frequency scaling, and found the linear model described in Section 3.2.1 to be a good fit. We use the first-order model developed in Section 3.2.1 to estimate the energy as a function of DVFS scaling.

3.5 Evaluation Results

In this section, we evaluate the performance and energy efficiency of our AAWS runtime with *work-pacing*, *work-sprinting*, and *work-mugging* against our baseline system.

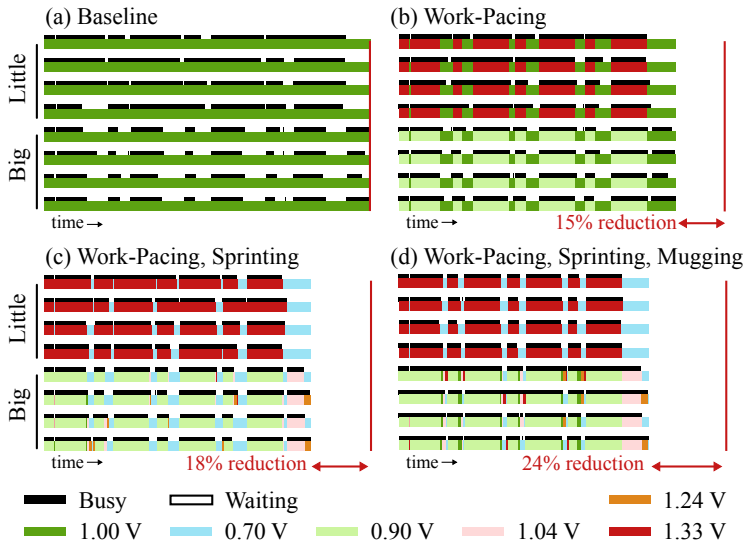


Figure 3.12: Activity Profiles for *radix-2* on 4B4L – Execution times of (b), (c), and (d) normalized to (a). Each row corresponds to a core's activity (black strip) and DVFS operating mode (colored strip) over time. (a) baseline 4B4L system; (b) applying work-pacing reduces HP region; (c) combining work-pacing and -sprinting reduces both HP and LP regions; (d) the complete AAWS runtime with work-pacing, sprinting, and mugging reduces execution time by 24%.

3.5.1 Performance of Baseline Work-Stealing Scheduler

Table 3.3 provides detailed performance and energy statistics for optimized serial code running on our single-core systems as well as for the parallelized versions running on our 1B7L and 4B4L systems. The big out-of-order core shows reasonable energy efficiency overhead and speedup compared to the little in-order core, similar to reported ratios collected by ARM during pre-silicon design-space exploration [Gre11]. The 4B4L system strictly increases performance over the 1B7L system, although we observe that the additional big cores do not always provide much performance benefit. Figure 3.12(a) shows per-core activity of the baseline 4B4L system executing *radix-2*. Notice that the execution time of *radix-2* is limited by LP regions created by lagging little cores.

3.5.2 Performance Analysis of Work-Pacing, Work-Sprinting, and Work-Mugging

In this subsection, we evaluate the performance benefit of work-pacing, work-sprinting, and work-mugging on both target systems. Figure 3.14 shows detailed execution time breakdowns for the (a) 4B4L system and the (b) 1B7L system. Each group of bars represents a single application, and bars incrementally add our techniques. To aid our evaluation, breakdowns within each bar represent the time spent in the serial region (*serial*), the HP region (*HP*), and the LP region. The

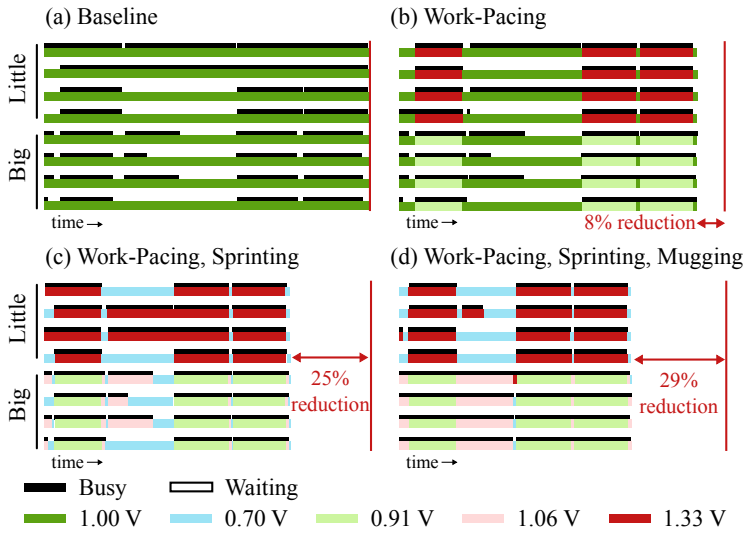


Figure 3.13: Activity Profiles for *hull* on 4B4L – Execution times of (b), (c), and (d) normalized to (a). Each row corresponds to a core's activity (black strip) and DVFS operating mode (colored strip) over time. (a) baseline 4B4L system; (b) applying work-pacing reduces HP region; (c) combining work-pacing and -sprinting reduces both HP and LP regions; (d) the complete AAWS runtime with work-pacing, sprinting, and mugging reduces execution time by 29%.

LP region is further broken down into three categories. First, we isolate the LP region within which the number of inactive big cores is too few to mug all work. In this region, the number of *big inactive* cores is fewer than the number of *little active* cores (*BI<LA*). Second, we isolate the LP region within which inactive big cores can mug all work from little cores. In this region, the number of *big inactive* cores matches or exceeds the number of *little active* cores (*BI>=LA*). Lastly, we gather the remaining LP region in which mugging is not possible into a single "other LP" category (*oLP*). The system with all of our techniques together is represented by *base+psm*. Note that the bar with work-mugging alone (*base+m*) serves as a comparison point without marginal utility techniques.

We evaluate work-pacing by comparing *base+p* to *base*. Work-pacing achieves reasonable performance benefits despite all cores often being completely busy (i.e., in the HP region) simply by applying a marginal-utility-based approach to the core voltages and frequencies. However, optimizing instantaneous system throughput can either mitigate or exacerbate load imbalance with realistic task sizes. For example, in the execution of *sarray* (4B4L), *radix-1* (1B7L and 4B4L), and *hull* (1B7L), benefits are outweighed by newly created lagging threads. Similarly, *qsort-1* on 4B4L sees smaller benefit because although the small HP region is sped up, the critical big core that generates new work is slowed down. Load balancing can also sometimes *improve* when

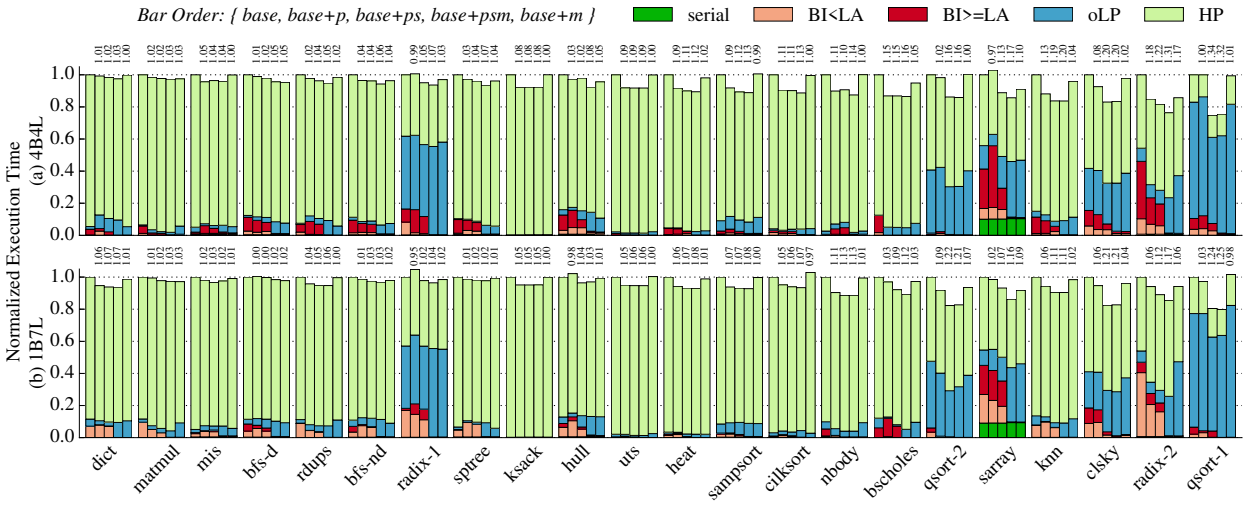


Figure 3.14: Normalized Execution Time Breakdown – (a) 1B7L configuration; (b) 4B4L configuration. Each group of bars represents a single application. First bar within each group is baseline with work-biasing and serial-region sprinting (*base*). Additional bars incrementally add our techniques: work-pacing (*base+p*); work-pacing and work-sprinting (*base+ps*); work-pacing, work-sprinting, and work-mugging (*base+psm*); work-mugging alone (*base+m*). All execution times are normalized to the baseline (*base*), and speedup is printed above each bar. Kernels are sorted by speedup for 4B4L *base+psm* from left to right. See Section 3.5 for breakdown explanations.

lagging threads are coincidentally eliminated, thereby reducing the LP region (e.g., *radix-2* 1B7L and 4B4L, *qsort-2* 1B7L). With $\alpha \approx 3$ and $\beta \approx 2$ (see Table 3.3), our analytical modeling suggests up to 12% benefit in the HP region (see Figure 3.4). Our results suggest that although work-pacing can sometimes achieve these predicted speedups, newly lagging threads can also limit the potential benefit. We can significantly mitigate this effect by applying work-sprinting in the LP region.

We evaluate work-sprinting by comparing *base+ps* to *base+p*. Work-sprinting rests waiting cores for power slack and then applies a marginal-utility-based approach to tune the voltage and frequency of active cores within the LP region. Notice that applications with large LP regions (i.e., combination of *BI<LA*, *BI>=LA*, and *oLP* regions) have the greatest performance improvements (e.g., *qsort-1, radix-2, clsky, knn, sarray, qsort-2*). Negative latency impacts from work-pacing are also significantly reduced (e.g., *sarray* 4B4L, *qsort-1* 4B4L, *radix-1* 1B7L and 4B4L). Note that *radix-1* sees little benefit because all four big cores are busy in the LP region, and the resting little cores do not generate enough power slack to sprint. Datasets can significantly impact the size of LP regions. In particular, note that *qsort-1* sorts an exponentially distributed dataset, making tasks very short or very long and creating large LP regions that work-sprinting can then exploit. Although work-sprinting allows little cores to sprint lagging threads, it is often optimal to move

the work onto big cores and sprint the big cores when possible (i.e., *BI>=LA* regions). We add the final AAWS technique, work-mugging, to take advantage of these opportunities for further benefit.

We evaluate work-mugging by comparing *base+psm* to *base+ps* as well as *base+m* to *base*. Work-mugging helps the LP region by moving work from slow little cores to fast big cores. First, notice that work-mugging eliminates all *BI<LA* and *BI>=LA* regions (i.e., all opportunities for work mugging are exhausted). In general, the movement of work between cores can result in smaller speedups (e.g., *bscholes* 1B7L, *sptree* 4B4L) as well as larger speedups (e.g., *hull* 4B4L, *radix-2* 4B4L, *sarray* 1B7L). Work-mugging provides the greatest benefit on applications with large *BI>=LA* regions, in which all work can be mugged onto fast big cores. Performance benefit is more limited in the *BI<LA* region in which some work must be left executing slowly on little cores. Although rarely the case, mugging overheads can cause minor slowdown. For example, *qsort-1* on 1B7L experiences slight slowdown with *base+m* compared to *base* when many tiny tasks at the end of the sort are quickly spawned and successively mugged onto the big core, incurring mugging overhead each time.

Figures 3.12(b-d) show activity profiles for *radix-2* executing on the 4B4L system (similar plots for *hull* shown in Figures 3.13(b-d)). By comparing Figure 3.12(a) and (b), we can see that during the HP region, the AAWS runtime tunes performance by raising the voltage of little cores and lowering the voltage of big cores (also see Figure 3.13(a) and (b)). In Figure 3.12(c), we see that the AAWS runtime rests waiting cores and then sprints the remaining active cores (also see Figure 3.13(c)). Most of the leftover work in the LP region is executing on little cores, limiting the performance benefit from work-sprinting. The little cores quickly reach their maximum voltage and frequency and leave a large amount of power headroom unused. Finally, Figure 3.12(d) shows work-mugging shifting tasks from little cores to big cores in the LP region for a total execution time reduction of 24% (*hull* reduces by 29% in Figure 3.13(d)).

In summary, work-pacing can provide performance benefit in the HP region as suggested by our analytical modeling, but due to realistic task sizes, the overall performance can vary widely when applied alone. The benefits of work-sprinting and work-mugging depend heavily on the presence of LP regions, which can vary with the application, the algorithm, and the dataset. Either technique can be applied independently to improve performance in the LP region. However, when non-ideal work-stealing schedules result in the presence of *BI>=LA* regions, work-sprinting and work-mugging can complement each other to provide the largest speedups. Finally, work-mugging and

DVFS transition overheads have minor performance impacts, likely due to the relative infrequency of mugs and DVFS transitions.

3.5.3 Performance Versus Energy Analysis

Figure 3.15 shows detailed performance and energy efficiency results for the 4B4L system. Notice that the general trend is higher performance and higher energy efficiency at similar power (i.e., tracking the isopower line). A more sophisticated adaptive DVFS control scheme with performance and energy counters could track the isopower line more closely, but a simple lookup-table-based DVFS controller is less complex and can generally improve both performance and energy efficiency with slight power variation. Figure 3.15(a) compares *base*, *base+p*, and *base+ps* and shows that work-pacing alone can improve both performance and energy efficiency for many applications at similar power. Several applications suffer reduced performance and energy efficiency due to the creation of lagging threads. Work-sprinting increases performance and energy efficiency, mitigating the impact of lagging threads. Detailed energy breakdown data (not shown) suggests that work-pacing and work-sprinting save energy because: (1) big cores execute at low-voltage operating modes, and (2) slower big cores do less work, allowing work to be stolen and executed on more energy-efficient little cores.

Figure 3.15(b) compares *base*, *base+psm*, and *base+m*. We show results for *base+m* as a comparison point without marginal utility techniques. Detailed energy breakdown data suggests that work-mugging significantly reduces the busy-waiting energy of cores in the steal loop, which are operating at nominal voltage and frequency without work-sprinting. We therefore notice that *base+m* improves both performance and energy efficiency. The complete AAWS runtime (*base+psm*) provides the greatest improvements across all applications. The strengths and weaknesses of work-pacing, work-sprinting, and work-mugging complement each other to elegantly adapt to diverse application-level behavior.

3.6 Related Work

While the MIT Cilk project helped recently popularize work-stealing [BL99, BJK+96, FLR98], the general idea dates to at least the early 1980's [BS84, RHH84]. There has been tremendous

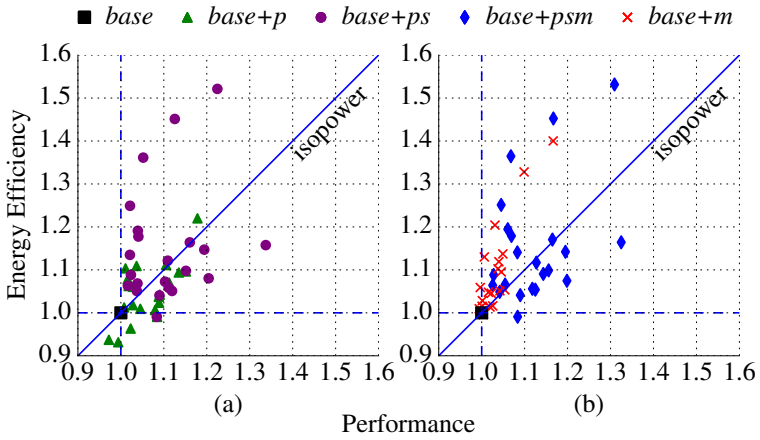


Figure 3.15: Energy Efficiency vs. Performance – Each point represents one application kernel's performance and energy efficiency running with a particular subset of the AAWS techniques, normalized to the same application kernel running on the baseline 4B4L system. Black markers on the origin represent *base* in both plots. An isopower line is drawn for reference, where points below the line are higher power and points above the line are lower power compared to the baseline. (a) *base*, *base+p*, *base+ps*. (b) *base*, *base+psm*, *base+m*.

work over the past decade on work-stealing runtimes (e.g., optimized task queue implementations [CL05], alternative victim selection strategies [CM08, BM09], efficiently supporting reduction operations across tasks [LSL12]). However, very little work explores the interaction between work-stealing schedulers and either static or dynamic asymmetry with a few notable exceptions.

Bender and Rabin proposed work-mugging as a way to migrate work from slow to fast cores and analyzed the theoretical impact of work-mugging on performance [BR02]. Follow-up work by Jovanović and Bender used high-level discrete-event simulation to explore the potential benefits and overheads of work-mugging [JB02]. We build on this earlier work with a realistic implementation and a new context. Chronaki et al. propose a dynamic task scheduler with constrained work stealing that attempts to schedule critical tasks to big cores [CRB+15]. This technique is most effective in applications with low parallel slack. Costero et al. [CIOQ15] group a big and little core together into a virtual core for the work-stealing runtime, and use a completely separate customized scheduler within the virtual core. Chen et al. propose the workload-aware task scheduler (WATS) which uses a combination of history-based task-execution-time prediction and task affinity [CCHG12]. WATS demonstrates good performance but is relatively complex and uses randomized victim selection in the baseline work-stealing runtimes. Previous work has shown the benefit of occupancy-based victim selection [CM08]. Ribic et al. proposed a work-stealing runtime that exploits dynamic asymmetry to improve energy efficiency, and they report reasonable energy benefits with modest performance loss on a real system [RL14]. Our proposal is fundamentally

different in that it focuses on improving both performance and energy efficiency by exploiting both static and dynamic asymmetry.

There has been a wealth of research on scheduling for statically asymmetric systems [KTR+04, KFJ+03, ARKK13, VCJE+12, JSMP12, JSMP13, LLK09, SMQP09, MWK+06]. Most closely related to our work are techniques that accelerate applications written with a thread-based parallel programming framework [JSMP12,JSMP13,LLK09,SMQP09,MWK+06,CIOQ15]. For example, Joao et al. propose bottleneck identification and scheduling which migrates programmer-identified bottlenecks to big cores [JSMP12], Lakshminarayana et al. propose progress performance counters to accelerate lagging threads [LLK09], and Joao et al. propose utility-based acceleration to accelerate both lagging and bottleneck threads [JSMP13]. These prior works focus on traditional thread-based parallel programming frameworks as opposed to task-based frameworks based on state-of-the-art work-stealing runtimes. They do not explore the interaction between static and dynamic asymmetry.

DVFS is perhaps one of the most well-studied techniques for power management [BPSB00, IBC+06, IM06, Dre11, JLB+15, SGS+14, GTB+14, KGWB08, EE11, BM09, MPT+12, LMH04, CGR+08]. Most closely related to our own work are techniques that accelerate multithreaded applications. For example, Miller et al. and Godycki et al. both propose instrumenting a traditional thread library to reactively sprint lagging threads in LP regions [MPT+12, GTB+14]. Cai et al. and Bhattacharjee et al. use instrumentation or prediction to rest waiting threads and sprint lagging threads [BM09, CGR+08]. While there are certainly similarities between this prior work and AAWS, there are unique opportunities involved in exploiting static and dynamic asymmetry within the context of a state-of-the-art work-stealing runtime.

Finally, some important work studies the tradeoffs between static and dynamic asymmetry, albeit with multiprogrammed workloads [LPD+14, AML+10, GRSW04]. Azizi et al. use a similar marginal-utility-based approach to explore circuit/architecture co-design and the impact of voltage scaling, but their work is purely within the context of VLSI design as opposed to adaptive scheduling for runtime systems [AML+10]. Lukefahr et al. argue that heterogeneous microarchitectures trump DVFS, but the study is within the context of a novel reconfigurable core (as opposed to static asymmetry) and uses multiprogrammed workloads scheduled optimally offline [LPD+14]. A key conclusion in these works is that heterogeneous microarchitectures can offer steeper utility curves,

while DVFS offers a shallower tradeoff. We see the same tradeoff in Figure 3.3(a), and we exploit this in our marginal-utility-based approach.

3.7 Conclusion

To our knowledge, this is the first work to explore the interaction between static asymmetry (in the form of heterogeneous microarchitectures), dynamic asymmetry (in the form of fine-grain DVFS), and work-stealing runtimes. We argue that work-stealing runtimes are a natural fit for managing asymmetry, but we also argue that there are unique opportunities for an asymmetry-aware work-stealing runtime. Through a mix of first-order modeling, numerical analysis, runtime software development, architecture-level simulation, and VLSI energy modeling, we have attempted to make the case that holistically combining static asymmetry, dynamic asymmetry, and work-stealing runtimes can improve performance and energy efficiency in future multicore systems.

CHAPTER 4
ULTRA-ELASTIC COARSE-GRAIN RECONFIGURABLE ARRAYS

Chapters 2 and 3 focused on fine-grain voltage and frequency scaling for general-purpose cores. This chapter narrows the focus to coarse-grain reconfigurable arrays (CGRAs). Here, I explore fine-grain voltage and frequency scaling for each tile and memory subbank at reconfiguration timescales, which may vary from hundreds of nanoseconds to milliseconds. CGRAs have become increasingly popular as specialized compute fabrics due to their potential for high performance while reducing control and data-movement energy. However, widespread industry adoption has been limited due to the complexity of compiler scheduling algorithms that must optimize across many constraints. Recent work on *elastic CGRAs* promises to significantly mitigate compiler-level challenges with hardware-managed flow control. In this chapter, I propose *ultra-elastic CGRAs* which capitalize on new opportunities in elastic CGRAs, enabling support for configurable per-tile fine-grain power control and significantly improved dataflow efficiency.

4.1 Introduction

Emerging application domains including machine learning, self-driving vehicles, augmented and virtual reality, and intelligence on the edge have increased the demand for energy-efficient specialized compute. Coarse-grain reconfigurable arrays (CGRAs) flexibly map dataflows to a spatial array of simple processing elements (PEs) and communicate intermediate data directly between PEs to reduce expensive data-movement energy in the memory hierarchy. CGRAs have been investigated with novel control schemes [PPA+13], novel network and routing [GHS11, GHN+12], efficient data movement [PSC+19], in the context of near-DRAM compute [GK16], and many others [VBP+16, BHME18, PPM09, FWC+18, wav18, OEPM09, MVV+03a, MVV+03b, ECF96, SLL+00].

However, extracting optimal performance and energy efficiency from the vast compute resources available in CGRAs remains a key challenge. The compiler is perhaps the most challenging aspect of CGRA optimization, as it must optimize utilization of the entire CGRA subject to resource contraints [GK16, GHS11, GHN+12, OEPM09, BHME18], potentially support predication [GK16, GHS11, GHN+12], potentially adapt to dynamic timing behavior unknown at compile

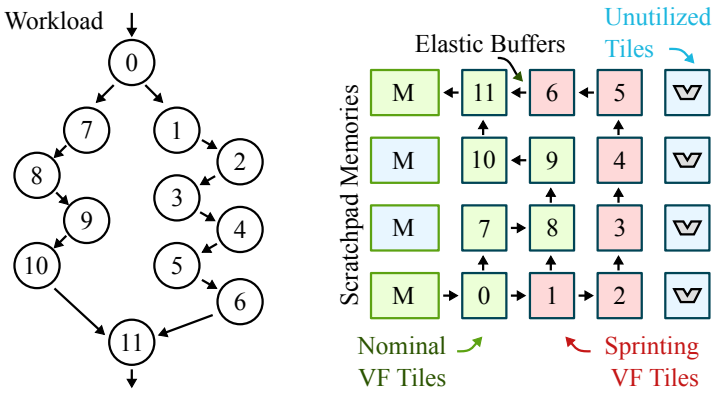


Figure 4.1: UE-CGRA Vision – The UE-CGRA platform enables configurable per-tile fine-grain DVFS. A dataflow graph (DFG) is mapped onto a 4×4 UE-CGRA using elastic buffers to reduce compiler complexity. The critical path is sprinted to a higher voltage and frequency and unutilized tiles are power-gated.

time (e.g., memory access latencies) [HIT+13], and minimize overheads across both space and time. As a result, compiler complexity has long been an area of active research [MVV+03a, MVV+03b, ECF96, SLL+00, OEPM09]. Despite the increasing popularity of CGRAs, these challenges have largely limited widespread industry adoption [KCC+12, wav18].

Recent work on *elastic CGRAs* [HIT+13] promises to significantly mitigate compiler-level challenges with hardware-managed control flow at the cost of modest area, timing, and energy overheads. Elastic CGRAs take a new approach based on latency-insensitive interfaces to determine control and data flow dynamically at runtime as operands become available at each tile. With exact scheduling of operations no longer fixed in advance, complex scheduling is elegantly handled with token propagation at the latency-insensitive interfaces. Other works integrate elasticity in their interconnects as well [GHS11, GHN+12, FWC+18]. The architecture presented in [HIT+13] demonstrated modest overheads (26% area overhead, 8% critical path overhead, and 53% energy overhead) in exchange for significantly reduced compiler complexity, making a strong case for elasticity as a key enabler for encouraging more widespread adoption of CGRA beyond academia.

Elastic CGRAs also open opportunities to explore new hardware techniques that can capitalize on their latency-insensitive interfaces. Specifically, fine-grain dynamic voltage and frequency scaling (DVFS) can enable small elastic per-tile voltage and frequency domains, allowing chains of connected tiles to "rest" at lower voltages and frequencies to save energy, while other chains of connected tiles can "sprint" at higher voltages and frequencies to improve performance. Fine-grain DVFS has already been demonstrated to improve both performance and energy for multicores in academia [TCM+09, GTB+14, TWB16, KGWB08, LZW+15] and in industry [Kan13, MBH+14,

Kan17]. However, enabling fine-grain DVFS at the PE level is not straightforward. Per-domain fully integrated voltage regulators and PLLs would occupy more area than the PE, and common alternative clocking schemes based on ring oscillators suffer from high phase noise. Furthermore, the resulting asynchronous crossings add synchronization latency and require specialized expertise, methodologies, and verification tools [Cum08]. Exponentially rising non-recurring engineering costs motivate a greater focus on mitigating verifiability challenges [KZVT17]. Previous work has explored functional-unit-level fine-grain DVFS in various contexts, including within an out-of-order processor [SAD$^+$02], for inelastic CGRAs [JTH$^+$13, JBH$^+$13], for arbitrary logic partitionings [MYN$^+$11], and for individual circuits in isolation [YCZ12]. However, many assumptions are made (e.g., per-domain PLLs, asynchronous FIFOs, ignoring synchronization latency) that make achieving both high performance and high energy efficiency a challenging prospect.

In this chapter, we propose *ultra-elastic CGRAs (UE-CGRAs)*, a novel extension to elastic CGRAs carefully co-designed with the elastic control circuitry to enable configurable per-tile fine-grain DVFS with reasonable overheads. Figure 4.1 illustrates the vision for the UE-CGRA platform. An example dataflow graph (DFG) with a simple parallel fork-join pattern is mapped onto a 4×4 UE-CGRA, which interconnects all tiles with elastic buffers (i.e., tiles wait for all of their operands before firing). Unutilized tiles are power-gated as in previous literature [wav18]. Remaining tiles can be configured for different voltages and frequencies to execute the DFG more efficiently. For example in Figure 4.1, data moves through the long path more quickly, enabling the joining tile to fire several cycles earlier than would otherwise be possible. We propose carefully designed ratiochronous interfaces and multi-rail power supplies to enable fine-grain DVFS at low overhead. The rest of this chapter describes the complete UE-CGRA platform including the analytical model (Section 4.2), compiler support (Section 4.3), the architectural template (Section 4.4), and the VLSI circuitry (Section 4.5) before evaluating our results.

This work makes the following five contributions: (1) a UE-CGRA analytical model for rapid exploration of performance and energy trade-offs; (2) a UE-CGRA compiler with a simple two-phase power configuration pass; (3) a UE-CGRA architectural template which implements the UE-CGRA analytical model; (4) a detailed overview of the VLSI circuitry required for per-tile fine-grain DVFS with reasonable overheads; and (5) an evaluation of the UE-CGRA design space using a vertically integrated research methodology spanning register-transfer-, gate-, and transistor-level modeling.

4.2 UE-CGRA Analytical Modeling

We provide intuition for the ultra-elastic CGRA computational model using first-order analytical performance and power modeling. The analytical model is composed of a discrete-event performance simulator modeling dataflow on the UE-CGRA computational model as well as a first-order power model to estimate energy.

4.2.1 Discrete-Event Performance Model

We designed a simple discrete-event simulator that models the performance of a dataflow graph (DFG) executing on both an elastic CGRA and an ultra-elastic CGRA.

Figure 4.2(a) illustrates a toy dataflow graph with thirteen nodes, two live-ins, one live-out, and one cycle. Each node in the DFG "fires" when all of its input tokens have arrived along the incoming edges. If any input token has not yet arrived, the node applies backpressure. The node also stalls if the downstream node is not ready to accept a new token. As time advances, tokens flow along the DFG until they exit the graph.

The discrete-event simulator models variation in performance by ticking nodes running at higher voltages at higher frequencies and ticking nodes running at lower voltages at lower frequencies. We model two-entry queues for each edge, and we also model wire delays (i.e., tokens may only propagate after a cycle time of delay). Specifically, Figure 4.2(b) shows performance estimates on the x-axis for a simple sweep over different voltage and frequency settings for each node across all nodes in the DFG (see Section 4.6 for how we selected voltage-frequency pairs). The circled point corresponds to a 1.4 $\times$ speedup achieved by "sprinting" the six-node cycle, which propagates the critical token more quickly. Note that non-critical nodes "rest" and include live-ins and live-outs (representing SRAMs) and other operators.

Our performance model is able to execute more complex DFGs with multiple live-ins, live-outs, parallel fork-joins, cyclic inter-iteration dependencies, and with many nodes. Note that any details that impact mappability to a real CGRA (e.g., routing) are abstracted away. To more closely model hardware, we restrict the number of incoming and outgoing edges to four for each node in each direction. Note that we only model CGRAs with single-cycle operations. Compared to a detailed hardware implementation, our simulator assumes that all nodes map to a unique tile and that any tile can communicate with any other tile.

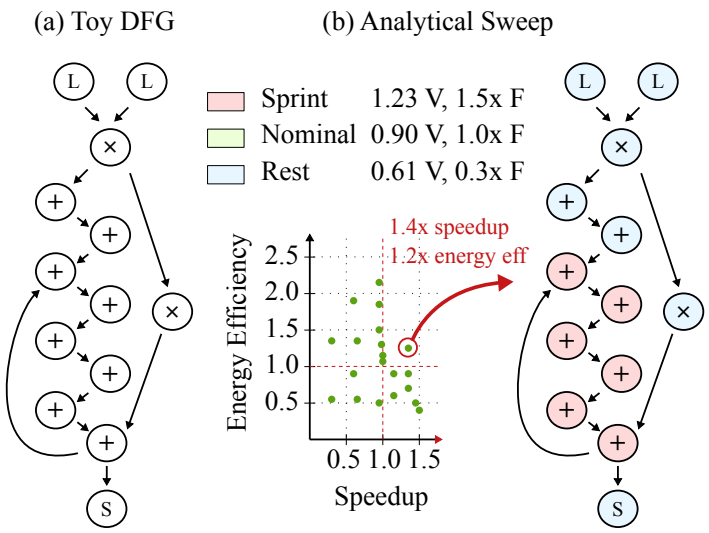


Figure 4.2: UE-CGRA Intuition with Analytical Modeling – A toy dataflow graph is used to provide intuition on UE-CGRA execution. (a) DFG with thirteen nodes, two live-ins, one live-out, and one cycle. (b) A simple sweep over individual voltages and frequency settings across all nodes results in varying performance and energy efficiency.

4.2.2 First-Order Energy Model

Consider a CGRA array comprised of N_T tiles of which N_{TA} are configured active and N_{TI} are configured inactive. The CGRA has N_S SRAM subbanks along the top and bottom perimeters of which N_{SA} banks are configured active and N_{SI} banks are configured inactive.

We assume that frequency is a polynomial function of voltage (validated using circuit-level simulation, see Section 4.6). The frequency of each active tile and SRAM subbank is:

$$f_{TAi} = k_1 V_{TAi}^2 + k_2 V_{TAi} + k_3 \qquad (i = 1,2,...,N_{TA})$$

$$f_{SAj} = k_1 V_{SAj}^2 + k_2 V_{SAj} + k_3 \qquad (j = 1,2,...,N_{SA})$$

where k_1, k_2, and k_3 are fitted parameters, f_{TAi} is the frequency of tile i, V_{TAi} is the voltage of tile i, and so on.

We assume the throughput of an active tile or SRAM subbank is measured in iterations per second (IPS) and that the throughput of any active tile or SRAM subbank is equal to the throughput of the entire CGRA. This is valid because each token processed by the CGRA has visited each active tile and each SRAM subbank exactly once. Tokens that flow along cyclic recurrence edges do not count twice, as they represent inputs for the next iteration of computation:

$$IPS_{Ai} = IPC_{CGRA} \qquad (i = 1,2,...,N_{TA})$$

Power estimation is intimately tied to both throughput and latency, and these parameters depend on the dataflow in the DFG. We use our discrete-event performance simulator to estimate both throughput and latency for the DFG. We later use this throughput estimate to calculate dynamic energy, and we also use the latency estimate to calculate static energy. Note that these raw numbers are never used in isolation. All analytical results are normalized and reported relative to another design point.

The tile and SRAM subbank powers includes both dynamic and static power and are modeled as:

$$\begin{aligned} P_{TAi} &= \alpha_{i,op}\, IPC_{CGRA}\, f_{TAi}\, V_{TAi}^2 + V_{TAi}\, I_{T,leak} && (i = 1,2,...,N_{TA}) \\ P_{SAj} &= \alpha_{sram}\, IPC_{CGRA}\, f_{SAj}\, V_{SAj}^2 + V_{SAj}\, I_{S,leak} && (j = 1,2,...,N_{SA}) \end{aligned} \tag{4.1}$$

The factor $\alpha_{i,op}$ is in the set $\{\alpha_{mul}, \alpha_{add}, \alpha_{sll}, \alpha_{srl}, \alpha_{and}, \alpha_{cp0}\}$ and captures the relative energy overhead of a tile executing the `op` operation at nominal voltage V_N and nominal frequency F_N compared to a tile executing the `mul` operation at the same voltage and frequency. Note that α_{sram} is similarly defined relative to α_{mul}.

We calculate the leakage current by assuming an architect targets leakage power to consume a certain percentage (denoted as γ) of the total power of a tile executing a multiply at nominal voltage and frequency.

$$\gamma = \frac{V_N\, I_{T,leak}}{\alpha_{mul}\, IPC_{CGRA}\, f_N\, V_N^2 + V_N\, I_{T,leak}} \tag{4.2}$$

We assume an SRAM bank's leakage current is a multiplicative factor (denoted by β) of the tile's leakage current.

$$I_{S,leak} = \beta\, I_{T,leak} \tag{4.3}$$

We use P_{TI} and P_{SI} to refer to the power consumed by tiles and SRAM banks which are inactive. We assume that these values are both zero, indicating that power-gated tiles consume no additional power.

The total power is the aggregate power across all tiles and SRAMs:

$$P_{total} = \sum_{i=1}^{N_{TA}} P_{TAi} + \sum_{j=1}^{N_{SA}} P_{SAj} + N_{TI}\,(P_{TI}) + N_{SI}\,(P_{SI}) \tag{4.4}$$

4.2.3 Analytical Case Study

The analytical results shown in Figure 4.2(b) sweep different voltage and frequency settings for each node across all nodes in the DFG using our analytical model. The energy for each node is modeled with the specific operator running at a specific voltage and frequency. We make the following assumptions for parameters in the UE-CGRA analytical energy model: k_1 = -1161.6, $k_2 = 4056.9$, $k_3 = 1689.1$, $V_N = 0.9$ V, $V_{min} = 0.61$ V, $V_{max} = 1.23$ V, $f_N = 750$ MHz, $\gamma = 0.1$, $\beta = 2.0$, $\alpha_{sram} = 0.82$ (per 4 kB subbank), $\alpha_{mul} = 1.0$, $\alpha_{add} = 0.30$, $\alpha_{sll} = 0.37$, $\alpha_{srl} = 0.35$, $\alpha_{cp0} = 0.23$, $\alpha_{and} = 0.30$, $\alpha_{or} = 0.33$, $\alpha_{xor} = 0.42$, $\alpha_{eq} = 0.23$, $\alpha_{ne} = 0.23$, $\alpha_{gt} = 0.25$, $\alpha_{geq} = 0.25$, $\alpha_{lt} = 0.25$, $\alpha_{leq} = 0.25$, $\alpha_{bps} = 0.11$. These parameters are derived from VLSI modeling for the target voltage range and system described in Section 4.7.

The circled point combines sprinting and resting for 1.4 × speedup and 1.2 × energy efficiency. Note that sprinting the six-node cycle increases energy, but resting non-critical nodes reduces energy (in particular, live-ins and live-outs represent power-hungry SRAMs). The results also suggest that resting can enable 2.2 × energy efficiency at similar performance.

4.3 UE-CGRA Compiler

In this section, we introduce the UE-CGRA compiler with a simple two-phase power configuration pass. Figure 4.3 provides an overview of our LLVM-based compiler toolflow (see Section 4.6 for details). The compiler takes the C/C++ source code of a compute kernel without any modifications and generates a mapping onto a target UE-CGRA with a power mapping (i.e., DVFS configurations for each tile). The UE-CGRA compiler internally leverages the UE-CGRA analytical model to determine the optimal DVFS modes (i.e., voltage and frequency pairs) for each tile. Unused tiles are first power-gated as in previous literature [wav18]. We then apply the algorithm described in Figure 4.4, which conducts the mapping and employs heuristics to achieve a worst-case time complexity of $O(NM)$ for N nodes and M possible power modes (e.g., rest, nominal, sprint).

The algorithm first takes advantage of the producer-consumer relationship in dataflow graphs and groups single chains of nodes (i.e., each node in the chain has one input and one output) into a single logical power domain, significantly reducing the search space. The algorithm then

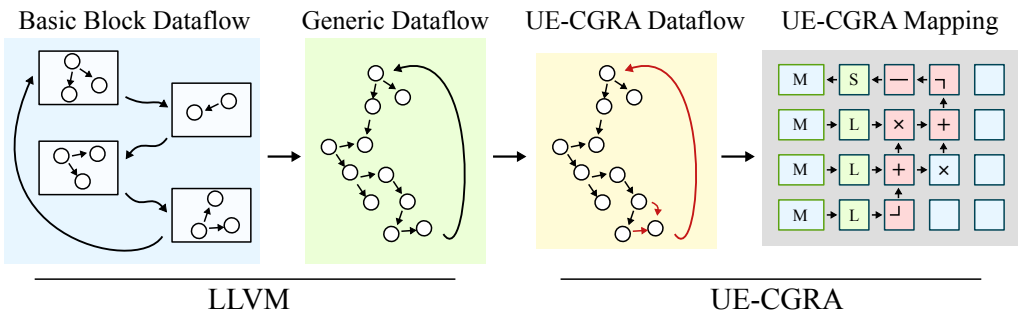


Figure 4.3: UE-CGRA Compiler Flow – The UE-CGRA compiler builds from LLVM to compile simple C programs iteratively into dataflow graphs. The generic dataflow graph is converted into a UE-CGRA dataflow graph targeting the UE-CGRA architectural template. This DFG gets mapped to the physical UE-CGRA with concrete dimensions and routing. Finally, a power mapping pass configures the voltages and frequencies with a performance target, an energy efficiency target, or both.

```
1:  procedure PowerMapping(N, T)
2:      for each node n in N do
3:          M(n) = V('sprint')
4:      end for
5:      G = GroupNodes(N)
6:      for each group g in G do
7:          M(g) = V('rest')
8:          if MeasureEnergyDelay(CGRA) < 1.0 then
9:              M(g) = V('nominal')
10:             if MeasureEnergyDelay(CGRA) < 1.0 then
11:                 M(g) = V('sprint')
12:             end if
13:         end if
14:     end for
15:     for each tile t in T do
16:         ConstrainTileModes(t)
17:     end for
18:     ConstrainLeakageCrossings(CGRA)
19: end procedure
```

Figure 4.4: UE-CGRA Compiler Power Mapping Algorithm – The power mapping algorithm maps tiles to rest, nominal, and sprint power modes. Heuristics are used to maximize energy-delay product with a preference for higher performance. N = set of all nodes; T = set of all tiles; $M(n)$ = power mode of node n; $V(x)$ = voltage level for string x.

initializes each node n to a power mode $M(n)$ at sprint voltage $V('sprint')$ before continuing with the *energy-delay-optimization* phase and the *constraint* phase:

Energy-Delay-Optimization Phase – In this phase, the compiler considers each group of nodes and tries to rest all nodes in the group for an improved energy-delay product. Since all nodes are initialized to sprint, performance is already maximized and the compiler slowly trades performance for greater factors of energy efficiency. The algorithm greedily attempts to rest first

for the greatest potential energy-efficiency benefit before attempting nominal, and then rolling back to sprint. The `MeasureEnergyDelay()` function estimates results using the analytical model. At the end of this phase, we achieve a power mapping with an energy-delay product strictly greater than that of the starting configuration.

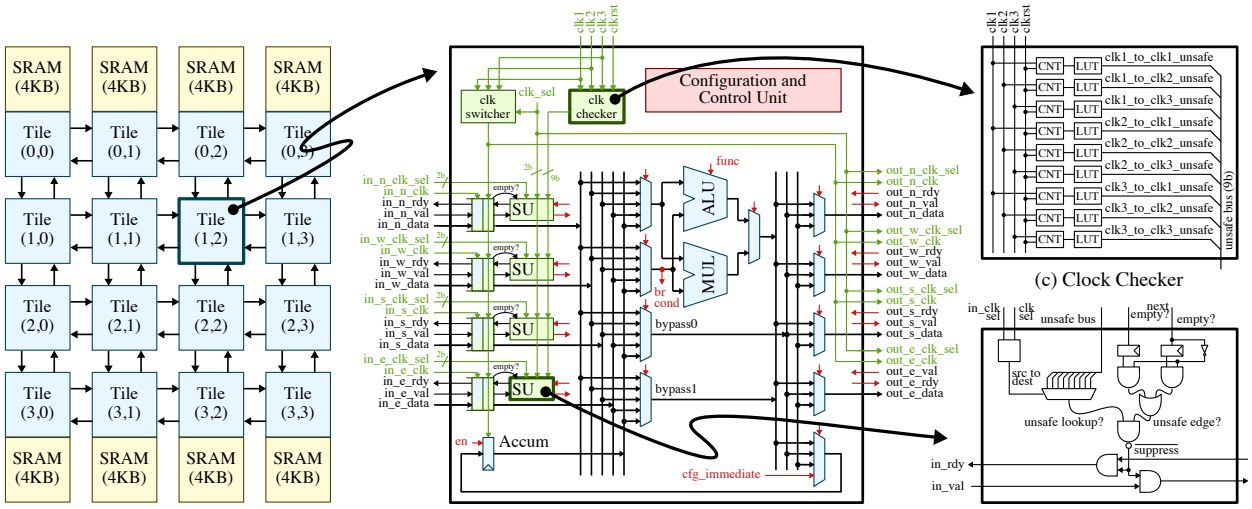
Constraint Phase – The nodes in the DFG may *fold* onto the same physical tile and therefore be limited to run at the same voltage and frequency. For example, a tile may execute a multiply with two inputs while bypassing a third unrelated input to an adjacent tile (i.e., routing through a busy tile). In this phase, the algorithm identifies all nodes mapped to tile t and chooses a single power mode in the event of disagreement. The `ConstrainTileModes()` function implements a small energy-delay optimization search across the options. Finally, the algorithm handles an additional constraint on multi-voltage-domain crossings. Low-voltage domains may have significant leakage current when interfacing with high-voltage domains or simply be unable to communicate (e.g., 0.61 V to 1.23 V). The algorithm identifies such crossings and either increases the resting voltage to nominal voltage (0.90 V) or inserts a new node at nominal voltage.

The algorithm in Figure 4.4 prioritizes performance by seeding the initial state to the maximum performance point (i.e., all sprinting). A variation of the algorithm can also prioritize energy by initializing all nodes to resting mode.

4.4 UE-CGRA Architecture

In the previous section, we introduced the UE-CGRA compiler flow that maps C programs and produces configurations for a UE-CGRA instance. In this section, we describe the UE-CGRA architectural template that composes tiles into a complete UE-CGRA. Figure 4.5(a-b) illustrate the block diagram of a 4×4 UE-CGRA and the components within a tile. The tile is carefully architected to enable both compute and bypassing of data (i.e., routing) on the same cycle. The design points include:

Input Queues – The input queues from each cardinal direction are not only elastic but are also designed to interface correctly between two clock domains with known phase relationship (see Section 4.5). Input queues are normal queues without bypass or pipeline behavior in order to break the critical paths at the queue boundaries across the entire CGRA. Without pipeline behavior, the



(a) 4x4 UE-CGRA (b) UE-CGRA Tile (c) Clock Checker (d) Suppressor Unit

Figure 4.5: UE-CGRA Architecture and VLSI – Detailed block diagrams: (a) a 4×4 UE-CGRA with 4 kB SRAM banks; (b) a UE-CGRA tile contains input queues, muxing, an operator block, a configuration register, a multi-purpose register (for accumulation and storing constants), a control unit with support for select functionality `phi` and branches `br`. Supported operations: `cp0`, `cp1`, `add`, `sub`, `sll`, `srl`, `and`, `or`, `xor`, `eq`, `ne`, `gt`, `geq`, `lt`, `leq`, `mul`, `phi`, `br`, `nop`. (c) Clock checker to detect safe and unsafe edges in ratiochronous clock-domain crossings; (d) Suppressor unit to disable handshakes on unsafe cycles.

queues must have two entries in order to support full throughput when all tiles are communicating on the same synchronous clock.

Multi-Purpose Register – A small single-entry register lives alongside the input queues and can be used for various purposes including storing configured constants, accumulating values, and implementing phi node behavior for recurrence edges (i.e., cycles in the DFG).

Muxing – Four input muxes select between the four input queues and the multi-purpose register. These supply the operands for the compute operator as well as for two bypass paths. Bypassing enables any input queue to forward data to any output, allowing messages to route through tiles executing other operations. Five output muxes select between the compute operator output and the bypass messages before forwarding the message towards one of the four cardinal directions or towards the multi-purpose register.

Operator – The compute operator supports the following operations in a 32-bit datapath: `cp0`, `cp1`, `add`, `sub`, `sll`, `srl`, `and`, `or`, `xor`, `eq`, `ne`, `gt`, `geq`, `lt`, `leq`, `mul`, `phi`, `br`, `nop`. These operations include control flow (handled as data flow). The multiply operation truncates the upper half of the result so that the inputs and outputs have identical bitwidths.

Phi Support – Phi nodes have the semantics of a flexible `select` operation and will fire when either of two configured input messages arrive. Phi behavior is paired with a recurrence initializa-

tion unit in the control unit which sends a single valid handshake immediately after reset. Together, these enable tiles to initialize loops (i.e., cycles) with valid data (e.g., an iteration variable). Because tiles wait for inputs, if these inputs are not initialized, the tile will never fire.

Branching Support – Branches are decoupled from the branch condition computation. A tile configured for a branch accepts a message and a select signal which determines whether the message is redirected towards one output or towards another. The branch bit is tapped from one operand which is then used as the condition. Note that all control flow is converted into data flow.

The complete UE-CGRA is illustrated in Figure 4.5(a) and is composed of a grid of UE-CGRA tiles interconnected with queues. Tiles along the north and south perimeters of the array can access SRAMs through memory adapters that convert elastic messages to and from SRAM requests. The configuration phase leverages the existing data network to forward configuration messages systolically through the array from top to bottom. When the correct messages have arrived at each tile, the bits are flashed into the configuration registers. The UE-CGRA has 26 configuration bits which fits within the bandwidth of an inter-tile message. If the configuration message has a larger bitwidth than the datapath, the configuration phase can be serialized over multiple transactions.

4.5 UE-CGRA VLSI

The previous sections have built upon assumptions about VLSI support in the UE-CGRA platform. In this section, we describe the UE-CGRA VLSI circuitry to enable per-tile fine-grain DVFS with reasonable overheads. Figure 4.5(b-d) illustrate the primary components of our clocking scheme.

Ratiochronous Clock-Domain Crossings – UE-CGRA tiles communicate synchronously over ratiochronous clock-domain crossings. The ratiochronous design pattern enforces *rational* clocking relationships across domains (e.g., frequency ratios of 1-to-3, 2-to-3). While less flexible than a fully asynchronous approach, this requirement quantizes an otherwise infinite space of possible clocking relationships, enabling industry-standard static timing analysis techniques to verify timing between domains. The ratiochronous family of cross-domain interfaces was initially proposed by Sarmenta et al., who published the seminal paper on rational clocking for phase-aligned clock domains [SPW95].

Suppressor Unit – Ratiochronous crossings have phasic relationships that may require suppression of "unsafe" edges. For example, clock edges at a 2-to-3 crossing repeat periodically at the least common multiple of six. Many of these edges are not aligned and are therefore "unsafe" for transmitting data. Because CGRA performance can be significantly affected by frequent periodic stalling, we extend prior work with an elasticity-aware suppressor unit that enables safe crossings in the presence of backpressure. Specifically, Figure 4.5(c) implements a traditional unsafe-edge detector [SPW95] for each crossing between three potential clock domains. Figure 4.5(d) shows how the empty signal from the input queues is used to implement two edge detectors that allow handshakes on unsafe edges as long as data has been enqueued for longer than one local clock cycle.

Bisynchronous Queues – We select a simple two-element bisynchronous normal queue to interface between two tiles at different clock domains. Messages sent are source-synchronous, meaning the writing clock is sent along with the data to be written. Because communication between same-clock-frequency tiles is common, the queue depth must be two elements even if clock frequencies vary. Static timing analysis is applied assuming nominal frequency across all tiles. Our design relies on suppressors to eliminate unsafe crossings at the architecture level, meaning that ASIC tools do not need to address these concerns at the VLSI level. We specifically do *not* use asynchronous queues as is assumed in most other literature [SAD$^+$02, YCZ12, JTH$^+$13], as these complicate verification [Cum08] and add two-to-three-cycle synchronization latency penalties which can significantly reduce performance in the context of a CGRA. Note that synchronization latencies in asynchronous queues only appear when the queues are empty or full, but this is precisely the common case in the context of a CGRA.

Clock Switchers and Dividers – We select between clocks within each tile with a traditional glitchless clock switcher and generate rational clocks with traditional counter-based clock dividers (e.g., divide by two, divide by three). Each of these units must be reset with a dedicated clock reset before the global reset is deasserted. This enables the clock switcher to generate the first edges, which are then used to reset the registers within the tile.

Multi-Rail Supply Voltages – The research community has studied fine-grain DVFS enabled either through multi-rail voltage supplies [MPT$^+$12,Dre11,TCM$^+$09] or with fully integrated voltage regulators [JLB$^+$15,SGS$^+$14,KGWB08,GTB$^+$14,FBB$^+$15,TWB16,GSSK12,LZW$^+$15]). In this work, we select a traditional multi-rail supply scheme with three voltages. This enables fast

	br	add	sub	and	mul	cmp	or	xor	shl	shr	phi	load	store	rec
latnrm	4	5	1	0	4	1	1	0	1	0	5	4	1	2
fft	1	10	3	0	4	1	0	0	0	0	1	4	4	1
susan	5	7	1	0	2	1	0	0	0	0	5	3	1	5
blowfish	3	8	0	4	0	1	3	3	0	3	3	5	1	2

Table 4.1: Benchmark Kernels – Operator decomposition for each kernel DFG, including the number of recurrence edges (`rec`).

reconfiguration time on the order of nanoseconds [TCM+09], at the cost of increased pin count and hierarchical power grid overhead. Because voltages in the UE-CGRA platform are only changed at configuration time, inductive di/dt rush currents can be avoided by gradually scaling voltages in multiple stages.

4.6 Methodology

We use a vertically integrated research methodology spanning software, architecture, and VLSI. In this section, we describe our benchmarks, LLVM compiler details, architectural modeling, and VLSI modeling.

4.6.1 Benchmarks

Table 4.1 lists the benchmarks we evaluate. For each benchmark, we list the operation composition with one operation per column. We also list the number of recurrence edges, representing the number of inter-iteration dependencies in each kernel's DFG. We map only the innermost loop. Each of the benchmarks fits within our 8×8 UE-CGRA.

4.6.2 Compiler

Our LLVM pass is implemented based on version 3.8.0. We generate a DFG for each kernel and then structurally map the DFG onto the UE-CGRA. We implement a control dataflow graph (CDFG) analysis pass to generate the CDFG (i.e., with both control and data dependency edges) for each compute kernel. The CDFG is tuned (e.g., control-dependency edges are converted to data-dependency edges) to generate a new DFG that contains only the set of operations supported

by the UE-CGRA architecture. Our mapper then attempts to map the DFG onto the UE-CGRA architecture. To be specific, we map one DFG node at a time onto an available tile in the UE-CGRA. If a dependent DFG node is already mapped, a valid path to route the data dependencies is calculated using Dijkstra's shortest-path algorithm. If a valid routing path is not found, we try to map the DFG node onto other available tiles. Finally, if no tiles are left, the most recently mapped DFG node will be re-mapped onto another tile and the process repeats until a valid mapping is reached.

4.6.3 Architecture and VLSI Modeling

We designed a parameterizable elastic CGRA (E-CGRA) and UE-CGRA within a productive Python-based hardware modeling framework [LZB14, JIB18]. We designed the RTL within this framework, translated to Verilog, and drove our testing with Python-based test harnesses. We designed and imported Verilog for all clock-related circuitry in the design.

We then pushed each 8×8 CGRA through an ASIC toolflow using a combination of Synopsys and Cadence tools (i.e., Design Compiler, Innovus, PrimeTime PX, and VCS) targeting a TSMC 28 nm process. We ran synthesis, place-and-route, gate-level simulation, and power estimation. We also ran SPICE-level simulations to help determine the relationship between voltage and frequency for our process technology across different operating modes. We used 21 delay stages consisting of multiple FO4 loaded inverters, NAND, and NOR gates connected in a loop, such that the total delay in the loop matches our gate-level cycle time for a given voltage. We used the change in delay vs. supply voltage as a model for tile voltage-frequency scaling.

4.6.4 Energy Modeling

We model energy for our 8×8 E-CGRA and our 8×8 UE-CGRA with simulation-driven gate-level power estimation using Synopsys VCS and PrimeTime PX. We simulated each benchmark separately on both the E-CGRA and the UE-CGRA gate-level model with the modified frequencies at nominal voltage. We model the energy of the entire UE-CGRA with a collection of data from both the E-CGRA and the UE-CGRA VLSI implementations. Specifically, this is accounted as the sum of the energy of the E-CGRA tiles, the suppression logic, and the clock switcher in all 64 UE-CGRA tiles. We scaled each tile to the new voltage according to first-order power scaling equations, re-accounted for leakage energy, and added global clock energy (which is not scaled to

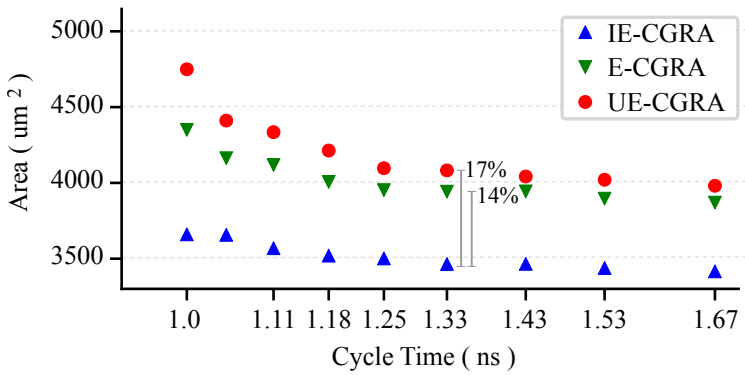


Figure 4.6: Tile Area vs. Cycle Latency – Tile areas of IE-CGRA, E-CGRA, and UE-CGRA across different cycle time targets.

a different voltage). The gate-level simulation is driven by 100 iterations of randomized input data for `fft`, `susan`, and `latnrm`, and 32 iterations for `bf`.

We also use simulation-driven gate-level power estimation for each CGRA tile (see Section 4.7.1) and propagate these values back to the UE-CGRA analytical model as α_{op} values.

4.7 Results

In this section, we evaluate ultra-elastic CGRAs against inelastic CGRAs and elastic CGRAs. We compare tile area and energy before studying CGRA area and cycle time. We then illustrate our UE-CGRA compiler mapping for each of our benchmarks along with their power configuration mappings. Finally, we evaluate performance and energy for each benchmark comparing an 8×8 E-CGRA and an 8×8 UE-CGRA.

4.7.1 Tile Area and Energy

We evaluate the energy and area of a single IE-CGRA tile, E-CGRA tile, and UE-CGRA tile in a 28 nm technology. Figure 4.6 illustrates the tile areas of IE-CGRA, E-CGRA, and UE-CGRA across different cycle time targets. Area increases as expected for more aggressive cycle time targets. In general, the E-CGRA tile and UE-CGRA tile are similar in size with a 14% and 17% overhead over the IE-CGRA tile at a 750 MHz clock target (1.33 ns). Notably, the E-CGRA has similar overheads over the IE-CGRA as studied in [HIT+13].

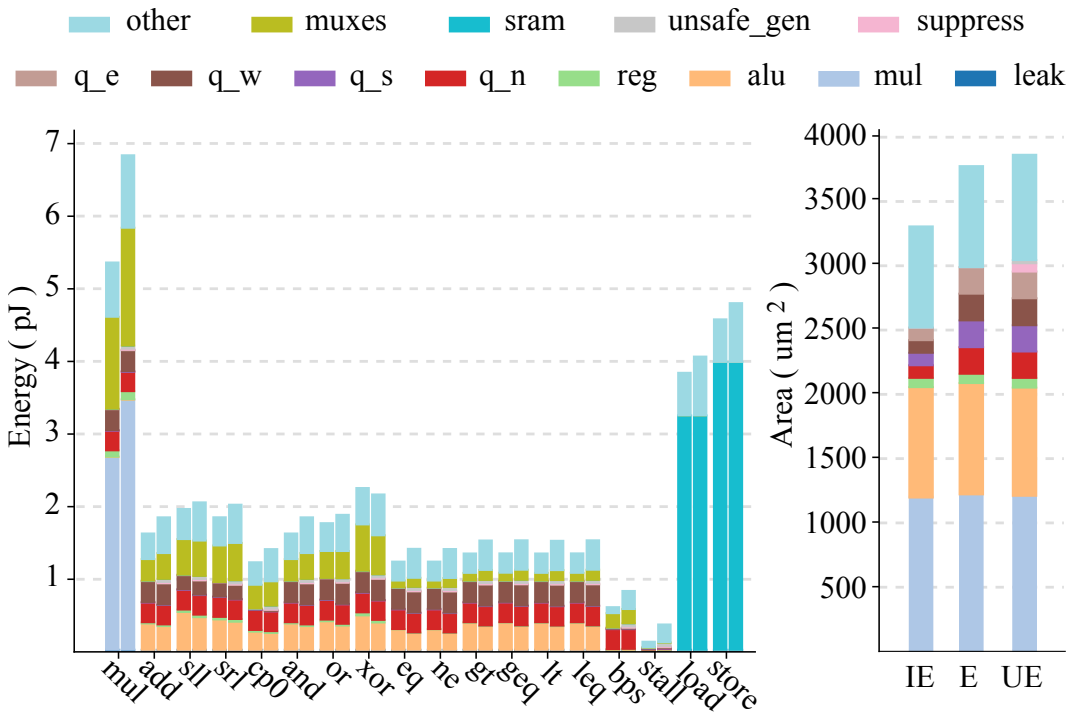


Figure 4.7: Tile Energy and Area Breakdowns – Tile energy breakdowns for each configurable operation across E-CGRA (left bar stack) and UE-CGRA (right bar stack) with a 750 MHz clock target in TSMC 28 nm. Tile area breakdowns are also shown for IE-CGRA, E-CGRA, and UE-CGRA. {`q_e`, `q_w`, `q_s`, `q_n`} = queues; `reg` = multi-purpose register; {`unsafe_gen`, `suppress`} = UE-CGRA VLSI circuitry; `mul` = multiply; `add` = add; {`sll`, `srl`} = shifts; {`and`, `or`, `xor`} = bitwise ops; {`eq`, `ne`, `gt`, `geq`, `lt`, `leq`} = compare; `bps` = bypass; `stall` = stall; {`load`, `store`} = SRAM access.

Figure 4.7 shows the energy breakdown for an E-CGRA tile and a UE-CGRA tile across all available operations. It also shows the area breakdown within each tile. On average, the UE-CGRA tile consumes 21% more energy across all operations as compared to an elastic tile. The unsafe clock-domain crossing suppression logic (field *unsafe_gen* and *suppress* in Figure 4.7), contributes to a minimal 1.3% energy overhead compared to an E-CGRA tile. Figure 4.7 also breaks down area overheads of each tile. Compared to an IE-CGRA tile with limited flow control, the UE-CGRA tile requires an extra 17% area. The area for logic that implements new functionality (e.g., unsafe crossing suppression) contributes only a small fraction. Instead, most of the area overhead comes from the same flow control logic necessary in the E-CGRA tile, as is evidenced by the similar 14% area overhead of an elastic tile over an inelastic tile.

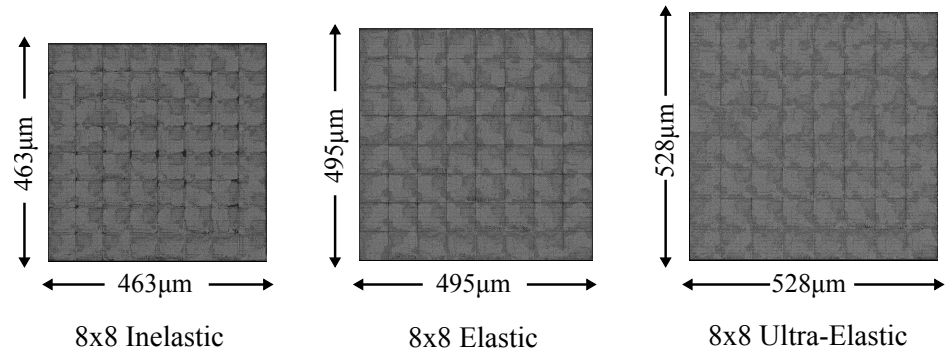


Figure 4.8: CGRA Layouts – CGRA layout for IE-CGRA, E-CGRA, and UE-CGRA targeting 750 MHz in TSMC 28 nm.

4.7.2 CGRA Area and Cycle Time

Figure 4.8 shows the layout of the three 8×8 CGRAs. The IE-CGRA has the smallest area (463 µm by 463 µm) due to its simpler architecture and lack of flow control support. The E-CGRA has slightly larger area (495 µm by 495 µm) with more flexible, hardware-managed flow control. The UE-CGRA has an area overhead of 14% compared to an E-CGRA due to three global clock networks, global clock dividers, and per-tile clock switchers and suppression logic.

All three CGRAs target a 750 MHz nominal clock frequency. The IE-CGRA meets the 750 MHz target, while the E-CGRA and UE-CGRA meet 747 MHz and 708 MHz, respectively. The length of the critical path grows as we add flow control and fine-grain DVFS functionality to the CGRA. All three of our CGRA implementations can be more aggressively tuned for higher clock frequencies.

4.7.3 Mapping Kernels

We map our application kernels to an 8×8 UE-CGRA to evaluate the potential benefits for both performance and energy. The UE-CGRA compiler power mapping algorithm assigns to each tile one of three rationally related DVFS power modes: *rest* (0.61 V, 0.33 × frequency), *nominal* (0.90 V, 1.00 × frequency), and *sprint* (1.23 V, 1.50 × frequency). For reference, Figure 4.9 and Figure 4.10 illustrate in detail how each operation is mapped to an 8×8 UE-CGRA.

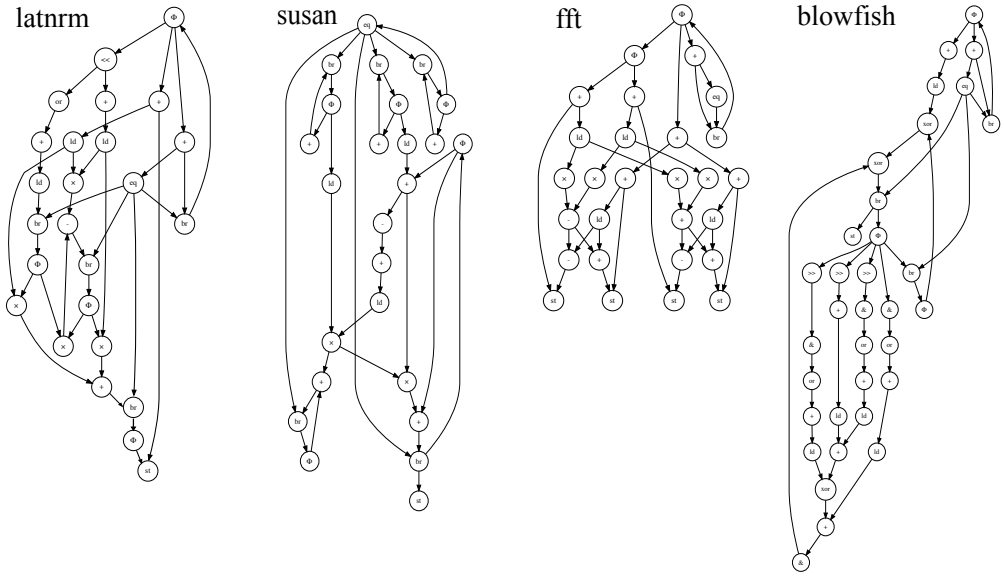


Figure 4.9: Kernel DFGs

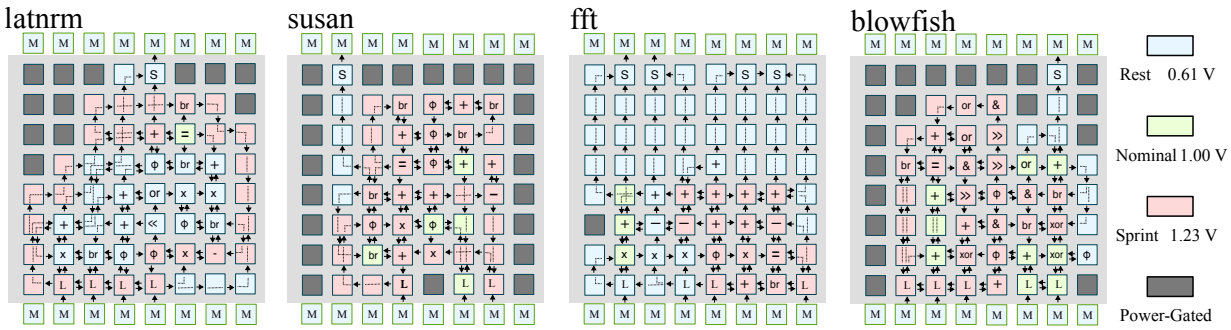


Figure 4.10: Kernel DFGs Mapped to UE-CGRA – Detailed mappings for each kernel on an 8 × 8 UE-CGRA.

4.7.4 Performance and Energy Efficiency

Performance and gate-level energy estimates are listed in Table 4.2 for the 8 × 8 UE-CGRA relative to the baseline 8 × 8 E-CGRA with all tiles running at nominal voltage and frequency. The results leverage the UE-CGRA compiler with the power mapping algorithm directed to prioritize energy. Our evaluation shows that an 8 × 8 UE-CGRA is able to improve energy efficiency by up to 2.18 × over an 8 × 8 E-CGRA on the selected kernels. The energy-optimizing DVFS strategy identifies tiles on non-critical paths that process transactions only once every few cycles and configures these tiles to rest at lower voltages and frequencies while still (slowly) computing.

Performance estimates for the throughput-optimizing DVFS strategy are listed in Table 4.3. These results show that the 8 × 8 UE-CGRA can improve throughput over the 8 × 8 E-CGRA by up to 1.48 × across our benchmarks, which matches our intuition for similar DFGs from Section 4.2.

	Analytical Model		VLSI Model	
	Perf	E.Eff	Perf	E.Eff
fft	1.00	1.85	0.98	2.04
susan	1.00	1.31	0.99	1.58
latnrm	1.00	1.80	0.90	2.18
bf	0.99	1.41	0.87	1.67

Table 4.2: Energy-Optimized UE-CGRA Results – Performance and energy efficiency of the energy-optimizing DVFS strategy on the 8×8 UE-CGRA normalized to the 8×8 E-CGRA. Results are shown for each kernel with both the analytical model estimate and running on the UE-CGRA VLSI implementation with gate-level energy estimation.

	Analytical Model	VLSI Model
	Perf	Perf
fft	1.46	1.44
susan	1.50	1.48
latnrm	1.49	1.35
bf	1.48	1.44

Table 4.3: Performance-Optimized UE-CGRA Results – Performance for the throughput-optimizing DVFS strategy on the 8×8 UE-CGRA normalized to the 8×8 E-CGRA. Results are shown for each kernel with both the analytical model estimate and running on the UE-CGRA VLSI implementation.

Figure 4.10 shows how the UE-CGRA sprints tiles on the critical path to accelerate the throughput of the entire dataflow graph. Both the analytical and VLSI results include startup overhead (i.e., the time for the first token to propagate through the DFG) and the performance is therefore slightly less than the expected $1.5\times$.

4.8 Related Work

Reconfigurable spatial architectures are an active area of research [GHN+12, GHS11, GK16, PPA+13, VBP+16, BHME18, PPM09, FWC+18, wav18, OEPM09, MVV+03a, MVV+03b, ECF96, SLL+00]. Well-known CGRAs include ADRES [MVV+03b], RaPiD [ECF96], and MorphoSys [SLL+00]. While some literature focuses on static CGRAs with single-functionality tiles, the majority of traditional CGRA research focuses on dynamic CGRAs which reconfigure tile functionality at runtime [OEPM09, PPM09] with dedicated scheduling algorithms [MVV+03a]. More recently, less traditional CGRAs have become popular including the DySER architecture with circuit-switched

interconnection [GHS11, GHN+12], the triggered instructions paradigm [PPA+13], the HRL architecture which combines CGRA datapaths with FPGA-inspired control hardware in the context of near-DRAM compute [GK16], and CGRAs targeting vision, object detection, and learning applications [VBP+16, BHME18, FWC+18, wav18], including in industry [wav18]. The tremendous flexibility of CGRAs and the importance of reducing data movement energy [PSC+19] promises a continuing interest in the CGRA space in multiple research communities.

Fine-grain DVFS has been studied with multi-rail voltage supplies [MPT+12, Dre11, TCM+09] and with fully integrated voltage regulators [JLB+15, SGS+14, KGWB08, GTB+14, FBB+15, TWB16, GSSK12, LZW+15]). Kim et al.'s initial system-level analysis of per-core DVFS sparked significant interest in the architecture community based on the energy-saving potential of per-core regulators [KGWB08]. While the majority of works prioritized saving energy, Godycki et al. proposed leveraging the power headroom from resting inactive cores to sprint active ones for performance [GTB+14]. Torng et al. proposed a similar scheme targeting work-stealing runtimes based on specialized knowledge of steal loop activity from hints embedded in the runtime [TWB16]. These works primarily explored fine-grain DVFS at the core granularity where per-domain regulators and PLLs might still be practical. Dreslinski explored multi-rail fine-grain DVFS to mitigate the performance loss of near-threshold compute [Dre11]. Truong et al. built a 167-processor array with per-core DVFS and provided a detailed account of the multi-rail supplies with power switches [TCM+09]. Interest in per-core DVFS is also increasing in industry [Kan13, MBH+14, Kan17].

There have been relatively fewer works that explore fine-grain DVFS at the PE level. Semeraro et al. [SAD+02] explored fine-grain DVFS across subunits of an out-of-order processor. They assumed asynchronous FIFOs and per-domain PLLs, both of which are likely to be intractable for CGRAs with very small domains. Jafri et al. [JTH+13] explored fine-grain DVFS for CGRAs targeting parallel application kernels mapped onto different partitions of the CGRA, running each full kernel at a different power mode. They configure the PEs themselves into dedicated tiles running synchronizer logic. Muramatsu et al. [MYN+11] explored fine-grain DVFS for a RISC-V core partitioned into a 6×7 grid, with each partition running at the lower of two voltages until a canary circuit signals a timing violation warning. Jafri et al. [JBH+13] explored a ratiochronous design approach in the context of CGRAs, but they focused their study on power and energy and

not on performance. They again configure PEs into synchronizers, and they do not explore the characteristics of elasticity.

There is a long history of research on GALS systems [Cha84, GOK+06, TGL07, IM02]. The most well-known GALS implementations were based on a pausible-clocking scheme with per-domain ring oscillators. The chips built at ETH Zurich were particularly well-known for this approach [GOK+06]. Intel has experimented with a mesochronous GALS chip with 80 tiles running at the same frequency but with unknown phase [VHR+07]. More recently, NVIDIA's research team designed a spatial accelerator with GALS domains enabled by pausible bisynchronous fifos [KKV+18].

The ratiochronous family of cross-domain interfaces was initially proposed by Sarmenta et al., who published the seminal paper on rational clocking for phase-aligned clock domains [SPW95]. Chabloz et al. continued exploring rational clocking and focused particularly on the case without phase alignment as an alternative approach to fully asynchronous GALS [CH09, CH10b, CH10a, Cha12, CH13]. Mekie et al. explored utilizing knowledge of the protocol to detect when data would not be transmitted, using this extra information to simplify the design of the ratiochronous interface. These works focused on the interface and communication mechanisms rather than architectural implications. The work most similar to this work in terms of the approach to ratiochronous design was proposed by Yadav et al. [YCZ12], who proposed a similar scheme of phase-aligned ratiochronous blocks dynamically supplied with PMOS power switches to enable fine-grain DVFS. Their approach primarily focused on single blocks in isolation and did not explore ratiochronous communication across multiple blocks, and they also assumed asynchronous FIFOs for correct interfacing. Our work differentiates by (1) avoiding the use of asynchronous FIFOs, (2) reorganizing the clock generation scheme to preserve phase relationships across domains, (3) relaxing the restrictive requirement for the critical path to run at PLL clock frequency, and (4) leveraging static timing analysis across clock domain crossings. Another closely related work for the approach to ratiochronous design is the RIRI scheme, which also generates rational clocks with a counter-based circuit and which also focuses on communication across phase-aligned boundaries [WWFY11]. However, their interface circuits were not designed to handle sophisticated communication micro-protocols. They also restricted the fast clock domain to meet timing with the PLL clock, which our work does not require, and they do not consider power-supply challenges.

4.9 Conclusion

This work explored the potential of UE-CGRAs to better orchestrate dataflow for improved performance and energy efficiency. We demonstrated that co-designing the compiler, architecture, and VLSI can enable UE-CGRA tiles to sprint through bottlenecks in portions of the DFG while resting tiles that execute less critical paths. We showed that careful VLSI design can preserve strict latency requirements between tiles and mitigate leakage concerns. Overall, UE-CGRAs promise to make CGRAs even stronger candidates for energy-efficient specialized compute that can flexibly target many emerging application domains.

CHAPTER 5
SILICON PROTOTYPING WITH FINE-GRAIN VOLTAGE AND FREQUENCY SCALING

The previous chapters explored fine-grain power-control techniques with analytical modeling, cycle-level modeling, RTL modeling, gate-level modeling, and SPICE-level modeling. This chapter describes my work on four silicon prototypes to support various aspects of my thesis. I tested two of these prototypes in our digital ASIC/FPGA prototyping lab, with the other two chips tested by collaborators. The chips include a mixed-signal test chip and three digital ASIC test chips. Of these chips, I was the project lead for two chips (BRGTC1 in IBM 130 nm [TWS+16] and BRGTC2 in TSMC 28 nm [TJAH+18]) and Cornell University student lead for the DARPA-funded, multi-university project on developing the Celerity SoC in TSMC 16 nm [AAHA+17, DXT+18, RZAH+19]. For the DCS test chip, I helped with full-custom design and also worked on the post-silicon testing process [BTG+17]. Figure 5.1 shows a timeline of these four silicon prototypes in the scope of my PhD career.

For each silicon prototype in the following sections, I will describe what the artifact is, what the research purpose is, how the chip connects to my thesis research, and my role in the project.

5.1 Dynamic Capacitance Sharing (DCS)

The DCS mixed-signal test chip contains four monolithically integrated switched-capacitor DC-DC converters in 65 nm CMOS. The tapeout was led by Waclaw Godycki and Ivan Bukreyev from Professor Alyssa Apsel's research group. Figure 5.2 shows an annotated chip plot. The final characterization of the test chip was published in the IEEE Transactions on Circuits and Systems I (IEEE TCAS I) [BTG+17].

5.1.1 Research Purpose

The purpose of the DCS test chip was to directly implement and characterize the voltage regulator designs that support reconfigurable power distribution networks as presented in Chapter 2, which was completed entirely in simulation with cycle-level modeling, RTL modeling, VLSI modeling, and SPICE-level modeling. DCS is an acronym that stands for dynamic capacitance sharing, describing the novel circuits technique for dynamically sharing small units of capacitance across

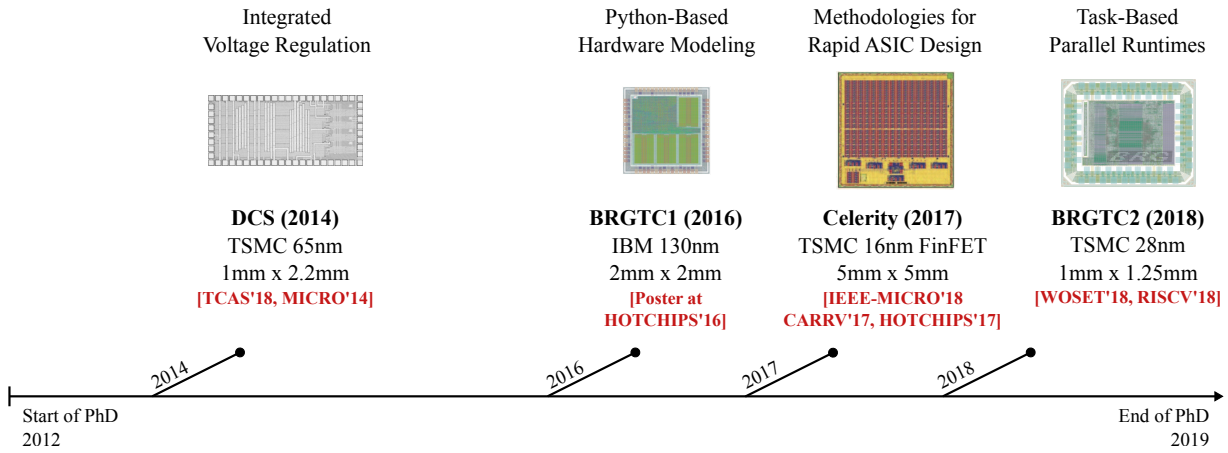


Figure 5.1: Timeline of Silicon Prototypes – Four silicon prototypes that support my thesis research and how they were published. *DCS* is a mixed-signal test chip supporting Chapter 2, and *BRGTC1, BRGTC2, and Celerity* are digital ASIC test chips that inspire Chapter 4 as well as future research directions.

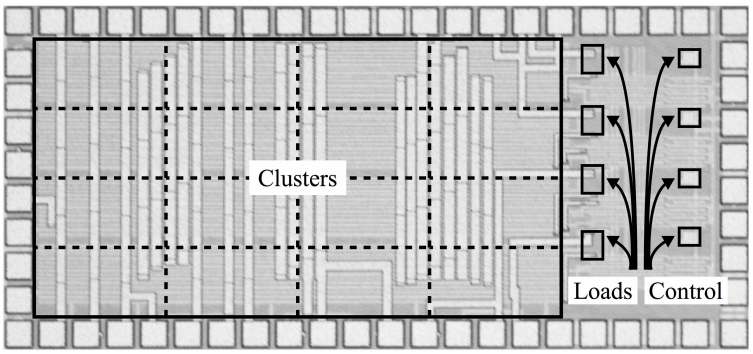


Figure 5.2: DCS Test Chip – Four monolithically integrated switched-capacitor DC-DC converters in 65 nm CMOS demonstrating the dynamic capacitance sharing technique.

multiple on-chip switched-capacitor voltage regulators for significantly reduced on-chip area and order-of-magnitude faster voltage transition times. The test chip has sixteen clusters and four regulator control loops and is fabricated in 65-nm bulk CMOS.

The primary goal for a system of DCS converters is to enable a wide range of output voltages at high efficiency to multiple loads. Our chip characterization results show that these goals were achieved. For a 2.3 V input, the fabricated DCS converters achieved 0.742 V at 38.1 mA to 1.367 V at 298 mA output with peak efficiency of 70.9% at 550 mW/mm^2 power density. Regulator area for the four-load network was reduced by up to 70% when operating under a power constraint compared with the stand-alone per-load regulators capable of supporting an equivalent range of operating voltages. In short, our DCS converters achieved efficiency competitive with the state of the art [See09] with the additional desired functionality.

These silicon results confirm the potential for applying the ideas presented in Chapter 2 to real systems. On this project, I was in charge of the full-custom layout for the digital control blocks that interconnect all reconfigurable clusters. I also worked on post-silicon validation at the bench to characterize the chip.

5.1.2 Chip Details

The majority of the chip area is dominated by the sixteen reconfigurable clusters. Each cluster has an area of 0.0766 mm2. Switches, drivers, and flying capacitance occupy 90% of the cluster area, while DCS-specific circuitry occupies 9.4%, with the power demux (5.9%) and clock generation unit (2.6%) as the major components. Aside from the clusters, the loads and control loops are also annotated. Each on-chip load is comprised of a 3-bit thermometer-encoded NMOS transistor bank and a parallel MOS capacitor of about 377 pF. Varying the number of connected parallel transistors from zero to seven allows rapid load current steps. Each control loop selects one of the four off-chip reference voltage biases through an analog mux. At runtime, any control loop can rapidly switch between any of the available reference voltage biases. Cluster allocation, load configuration, and reference voltage bias selection are all controlled by an on-chip serial peripheral interface and can be reconfigured at runtime, emulating a global DVFS controller. Cluster allocation configurations are programmed off-chip and stored in registers in each cluster, allowing configuration change on the fly for transient measurements.

To summarize, we found that DCS converters are well-suited for on-chip integration with multiple DVFS-enabled loads. We show that for cubic power scaling under a power constraint, DCS can reduce required energy-storage area by up to 70%. For a given fixed area, DCS VRs can increase the range of efficient output voltage regulation while simultaneously improving transient response times.

5.2 Batten Research Group Test Chip 1 (BRGTC1)

BRGTC1 is a 2×2 mm 1.3 M-transistor test chip in IBM 130 nm designed using PyMTL, a Python-based hardware modeling framework developed by my colleagues within my research

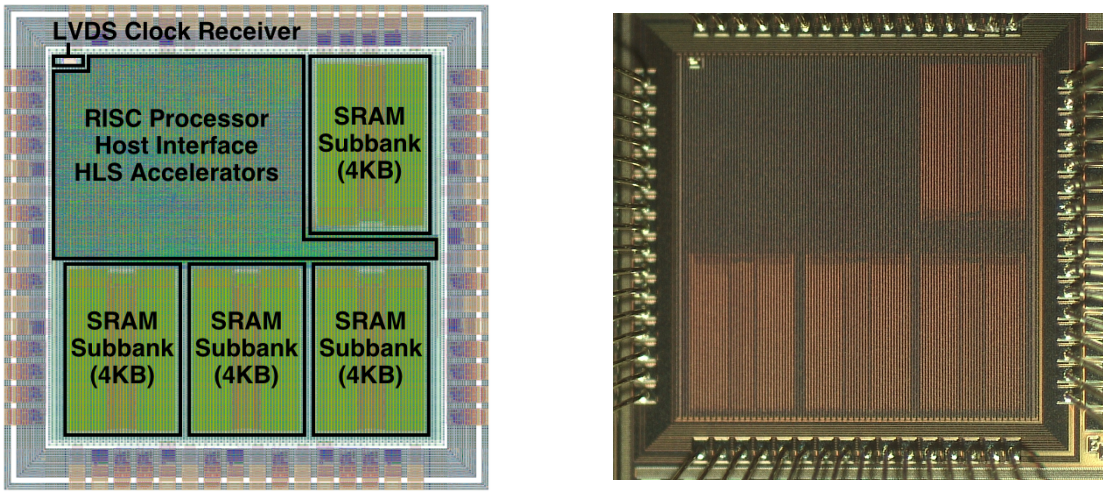


Figure 5.3: BRGTC1 Chip Plot and Die Photo – A 2×2 mm 1.3 M-transistor test chip in IBM 130 nm designed and implemented using our new PyMTL hardware modeling framework. The chip includes a simple pipelined 32-bit RISC processor, custom LVDS clock receiver, 16 KB of on-chip SRAM, and application-specific accelerators generated using commercial C-to-RTL high-level synthesis tools.

group (see [LZB14, JIB18]). Figure 5.3 shows an annotated chip plot and a die photo of the chip. This work appears as a Hot Chips 2016 student poster [TWS+16].

5.2.1 Research Purpose

The research purpose of the chip was to silicon-validate PyMTL-generated RTL. PyMTL brings compelling productivity benefits to hardware design and verification. The key idea of the framework is to embed a hardware design language within the high-level Python host language and then to leverage Python's powerful features (e.g., parameterization, reflection) as well as access to rich libraries (e.g., full-featured software testing frameworks) to rapidly design, verify, and compose hardware. See the PyMTL publications from my research group for more details [LZB14, JIB18]. We designed this chip with the second version of PyMTL, but PyMTL3 is available.

This project concluded with successful post-silicon testing of the chip, indicating that our PyMTL-driven pre-silicon design methodology was robust enough for future research and helping to provide insight into new ways to make PyMTL more useful for hardware design. Our design methodology (see Figure 5.4) uses PyMTL not only for design but also for composition with a commercial high-level synthesis (HLS) toolflow. The PyMTL RTL design is translated to Verilog RTL to target both a Xilinx-based FPGA backend and a Synopsys-based ASIC backend. The PyMTL testing framework features integration with mature, full-featured software testing tools (e.g., *pytest*). PyMTL-based testing enables using the same test vectors to validate the PyMTL

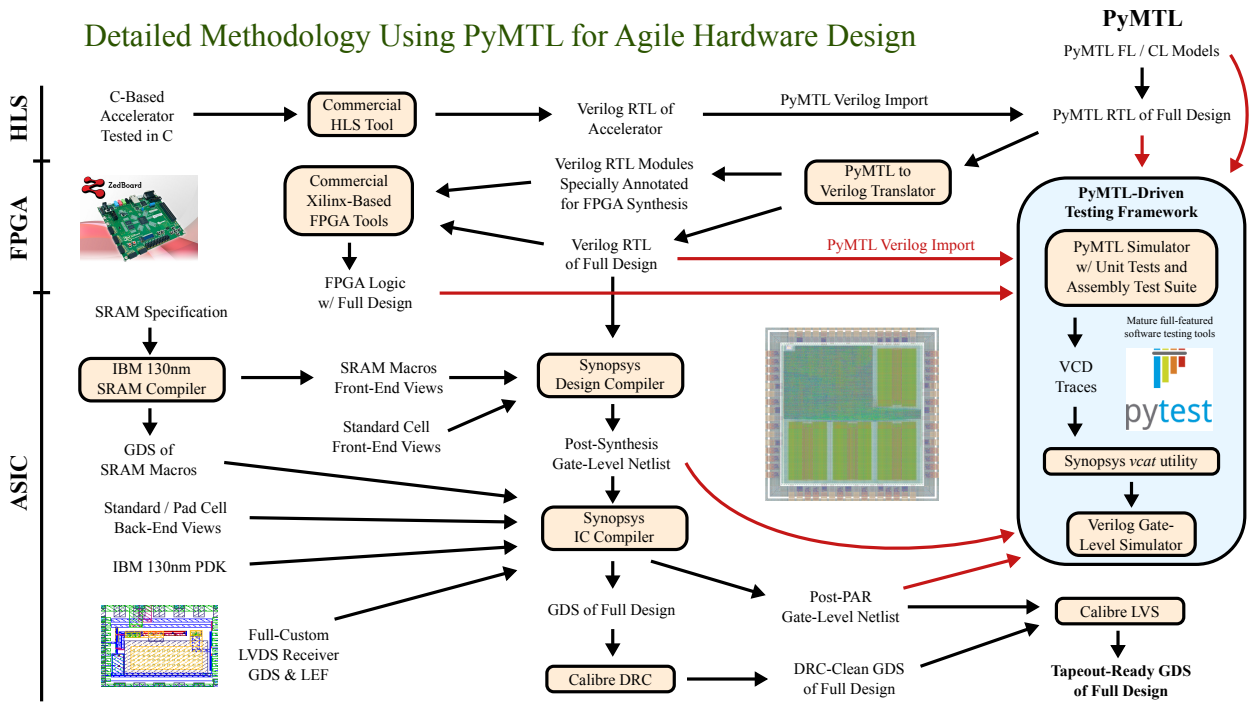


Figure 5.4: BRGTC1 Detailed Pre-Silicon Methodology

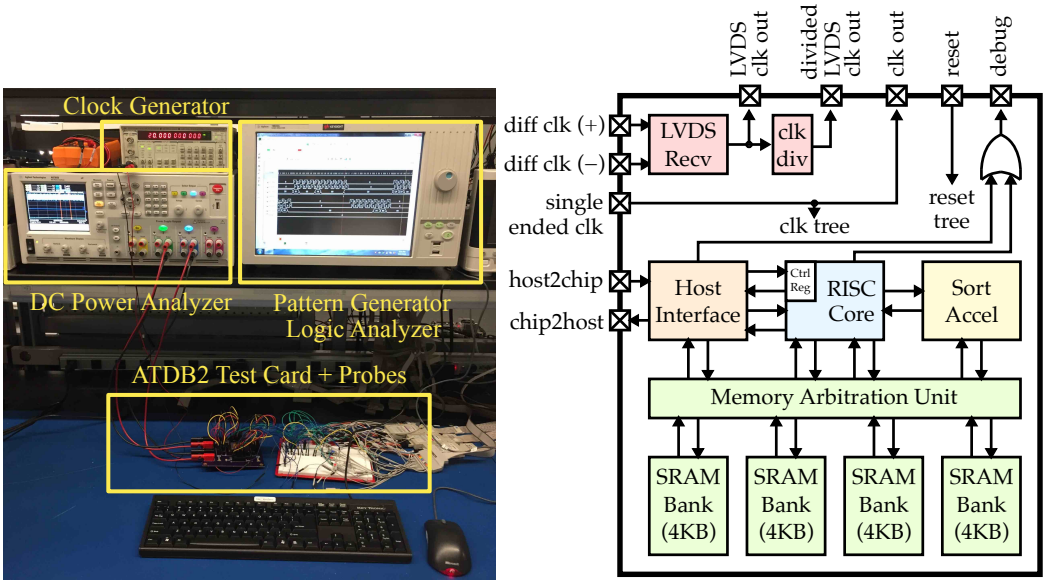
FL/CL/RTL designs, the Verilog RTL, the programmed FPGA logic, the post-synthesis gate-level netlist, the post-PAR gate-level netlist, and even the fabricated chip itself. These benefits worked to reduce the friction across different levels of verification and continued to be useful even during post-silicon testing. The PyMTL framework enabled our small team to iterate very rapidly, pushing the design from RTL to layout (and validated gate-level netlist) in less than a day.

The insights I gained from this silicon prototyping experience greatly contributed to my approach in addressing the VLSI-specific research challenges in Chapter 4. As a project lead, I was involved from architecture through to silicon and was also involved in the FPGA emulation, at the mixed-signal interface (i.e., for the full-custom clock receiver), and at the bench.

5.2.2 Chip Details

In this subsection, I detail our experimental setup and our post-silicon measurements for both performance and power.

Post-Silicon Experimental Setup – Figure 5.5(a) shows our bench setup with the equipment we used to run simple programs on BRGTC1 with and without hardware acceleration. We packaged BRGTC1 in a PGA package (CPG08462) and inserted the packaged die into our custom



(a) (b)

Figure 5.5: Experimental Setup – (a) The ATDB2 test card mounts a PGA-packaged BRGTC1 die and is externally powered, clocked, stimulated with test vectors from a pattern generator, and probed with a logic analyzer; (b) The BRGTC1 block diagram shows how messages are sent through the `host2chip` and `chip2host` ports (both are eight-bit asynchronous req/ack channels), are synchronized and deserialized by the host interface, and are then routed towards the core or SRAM banks.

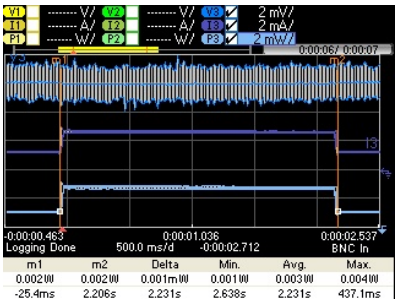
Parameter	Value
Die Dimensions	2×2 mm
Transistor Count	1.3 M
Technology	IBM 130 nm
Static Timing Freq	246 MHz

Parameter	Measurement
Core Voltage	1.2 V
IO Voltage	2.5 V
Static Power	0.97 mW
Idle Power	1.66 mW

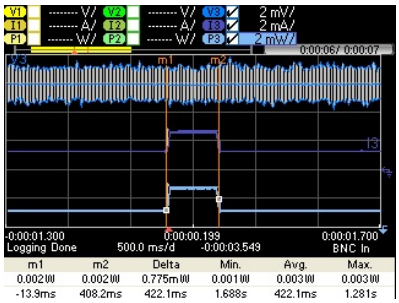
Table 5.1: System Parameters and Post-Silicon Resting Measurements

ATDB2 test card in a PGA zif socket (3M5067-ND). The ATDB2 test card is externally clocked with a Stanford Research Systems CG635 clock generator and is externally powered by an Agilent N6705B DC power analyzer. The power analyzer supplies both IO and core voltage levels. Finally, we program the Agilent 16822A pattern generator to provide test stimulus to the chip (e.g., writing programs to SRAM, communicating with the core), and we also use its logic analyzer capabilities to probe the requests and responses. All messages pass through the two eight-bit asynchronous `host2chip` and `chip2host` channels. The architectural block diagram in Figure 5.5(b) illustrates both channels and the internal chip connectivity.

Post-Silicon Resting Measurements – Table 5.1 lists the system parameters and resting measurements of the chip. The static and idle power are measured at the nominal IO and core voltages at room temperature. The static power was measured with all inputs (including the clock) grounded



(a)



(b)

Parameter	Measurement
Experiment Clock Frequency	20 MHz
Dataset Size	2000 integers
Dataset Distribution	`uniform random`
SW Sort Power	3.28 mW
SW Sort Performance	2.23 s
SW Sort Energy	7.32 J
HW-Accel Sort Power	3.13 mW
HW-Accel Sort Performance	0.42 s
HW-Accel Sort Energy	1.32 J

(c)

Figure 5.6: Case Study for Hardware-Accelerated Sorting – BRGTC1 tightly couples a general-purpose core with an HLS-generated bubble-sort accelerator. (a) Power measurement for *software-only sort*; (b) Power measurement for *hardware-accelerated sort*; (c) Measured case study parameters and calculations.

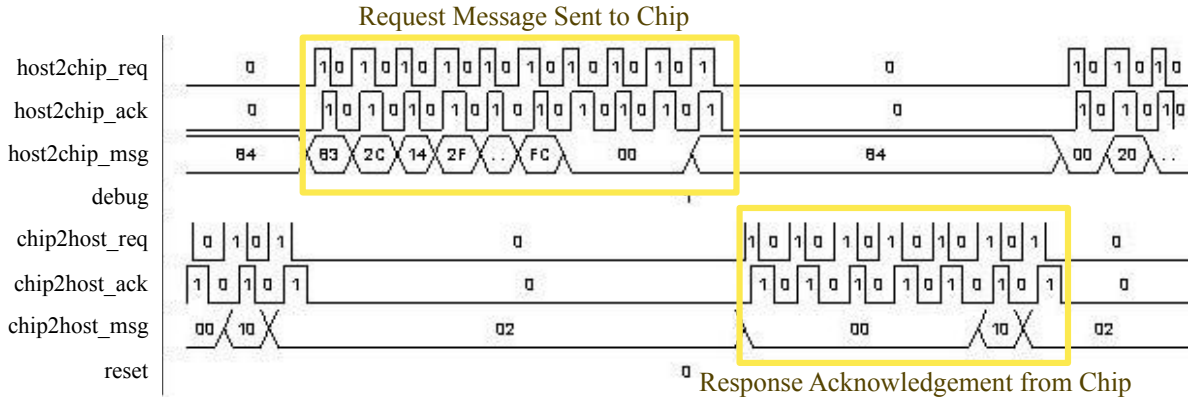


Figure 5.7: Messages Captured by Agilent 16822A Logic Analyzer – A serialized message is transmitted to the chip over the eight-bit asynchronous channel using four-phase req/ack handshakes, and the chip generates an acknowledgement message in response with additional handshakes. These specific messages correspond to an SRAM write of program code and a write acknowledgement from the chip.

and with reset asserted beforehand. The idle power was measured with all inputs grounded but while driving the clock at 20 MHz and after releasing reset. The core has a freeze bit which keeps the pipeline frozen during this measurement. The idle power therefore represents mostly clock tree power and logic that was not clock-gated.

Post-Silicon Accelerator Case Study – BRGTC1 tightly couples a general-purpose core with an HLS-generated bubble-sort accelerator (see Figure 5.5(b)). This presents the opportunity to measure the performance and energy of a sorting microbenchmark with and without hardware acceleration. The case study sorts a uniform-random distribution of 2000 integers filling roughly 8 kB of the on-chip SRAM.

The *software-only* sorting microbenchmark is hand-coded in assembly and programmed into the on-chip SRAM. The core is then released from freeze state to begin computation.

The *hardware-accelerated* sorting microbenchmark includes special accelerator instructions that the core uses to write two messages to the accelerator's internal register space. One message indicates the starting address of the data and the other is the number of integers to sort. Then the core blocks until it receives a done signal from the accelerator. The accelerator communicates directly with memory through the memory arbitration unit and raises its done signal after the sorting is complete.

The signal waveforms shown in Figure 5.7 are probed from the chip interface within the ATDB2 test card by the Agilent 16822A logic analyzer. The waveforms show a request and a response message on the eight-bit asynchronous `host2chip` and `chip2host` channels. On the request side, ten four-phase handshakes are required to send the full 80-bit request message, and on the response side, seven handshakes are required for the chip to acknowledge with a 56-bit response message. The logic analyzer allows very detailed observation of the activity at the chip interface. For example, these specific messages correspond to an SRAM write of program code (and a write acknowledgement from the chip) for the software-only sorting microbenchmark just before the core is released from its initial frozen state to begin execution. The logic analyzer has also been used to inspect many other messages to verify this case study.

We measured power for this case study with the Agilent N6705B DC power analyzer. The two power waveforms in Figure 5.6 show (a) the software-only bubble sort and (b) the hardware-accelerated bubble sort. Both scope views have the same scales at 2 mW/div and 500 ms/div, and it is clear that the hardware-accelerated sort completes more quickly. Figure 5.6(c) lists precise measurements for performance and power of the two sorts as well as the calculated energy. The hardware-accelerated sort achieves roughly 5.3× speedup and 5.5× lower energy compared to the software-only sort.

The insights I gained from this silicon prototyping experience greatly contributed to my approach in addressing the VLSI-specific research challenges in Chapter 4.

5.3 Batten Research Group Test Chip 2 (BRGTC2)

BRGTC2 is a small 1×1.25 mm 6.7 M-transistor RISC-V system in TSMC 28 nm. Like BRGTC1, BRGTC2 is also designed and implemented using PyMTL, our group's new Python-based hardware modeling framework developed by my colleagues [LZB14, JIB18]. Figure 5.8 shows an annotated chip plot and a die photo of the chip. A workshop paper about BRGTC2 was published at the MICRO 2018 RISC-V Day Workshop in Fukuoka, Japan [TJAH+18].

5.3.1 Research Purpose

The first research purpose of this chip was to silicon-validate PyMTL-generated RTL once again in a more advanced technology node. The second research purpose was to explore new preliminary ideas on hardware optimization for task-based parallel runtimes following in the spirit of AAWS runtimes in Chapter 3. Specifically, BRGTC2 integrates hardware for novel smart-sharing architectures that manage control flow to amortize expensive accesses to shared long-latency resources (e.g., caches, LLFUs) when executing task-based parallel workloads. The chip is designed to run a subset of our in-house work-stealing runtime and has been instrumented for detailed performance and energy characterization.

The ideas explored in BRGTC2 architecture are outside the scope of this thesis. However, the insights I gained from this silicon prototyping experience contributed to my interest in addressing the VLSI-specific research challenges in Chapter 4.

5.3.2 Chip Details

In this subsection, I provide an overview of various key aspects of our silicon prototyping experience with an emphasis on the open-source flow. Specifically, this includes our timeline and costs, open-source software toolchain and ISA, open-source cycle-level modeling, open-source RTL modeling, open-source modular VLSI build system, and synthesizable analog IP.

Timeline and Costs – A team of seven graduate students completed the 28 nm SoC in two months. This design period encompassed developing simple applications, porting an in-house work-stealing runtime to our RISC-V target, cycle-level design-space exploration of sharing architectures in gem5 [BBB+11], RTL development and testing of each component including SRAMs (see Figure 5.9), composition testing at RTL and gate level, SPICE-level modeling of the synthe-

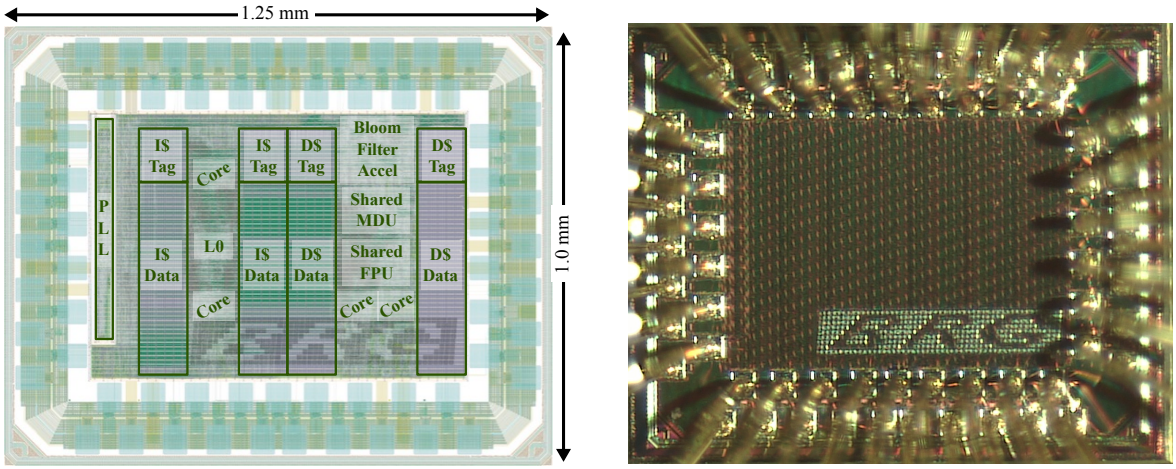


Figure 5.8: BRGTC2 Chip Plot and Die Photo – A 1×1.25 mm 6.7 M-transistor RISC-V system in TSMC 28 nm designed and implemented using our new PyMTL hardware modeling framework. The chip includes four RISC-V RV32IMAF cores which share a 32 kB instruction cache, 32 kB data cache, and a single-precision floating-point unit along with microarchitectural mechanisms to mitigate the performance impact of resource sharing. The chip also includes a fully synthesizable high-performance PLL written in SystemVerilog and ported from the Celerity SoC [DXT+18, AAHA+17, RZAH+19].

sizable PLL, IO floorplanning and physical design, post-place-and-route performance tuning, and final tapeout. About one person-month was required for me, a student with prior ASIC experience, to bring up the TSMC 28 nm design flow for the first time, including the process libraries, standard cell libraries, IO cell libraries, Synopsys DC, Cadence Innovus, and Calibre signoff tools, in order to pass DRC/LVS for dummy logic surrounded by staggered IO pads and no SRAM blocks. The entire chip RTL was designed in the final one-month period by seven graduate students using PyMTL for design, test, and composition. Multi-project wafer services have recently begun to support advanced technology nodes (e.g., 28 nm) with very small minimum sizes (e.g., 1×1 mm) at very reasonable pricing (e.g., $14K). We chose the Tiny2 program with MOSIS, selecting a 1×1.25 mm die size and one hundred parts for about $18K. Other services are also available for university researchers at similar pricing (e.g., Muse Semiconductor). Other costs included packaging (less than $2K for twenty parts), board costs (less than $1K for PCB and assembly), graduate student salaries, physical IP costs, and EDA tool licenses. The open-source RISC-V ecosystem helped us avoid any costs associated with the ISA and also helped avoid long communication delays with third parties, which can take months to resolve and can significantly delay a time-sensitive project. Many small benefits also made a difference (e.g., a very short but descriptive RISC-V ISA spec saving us from reading thousand-page specs, no time and effort required to bring up and

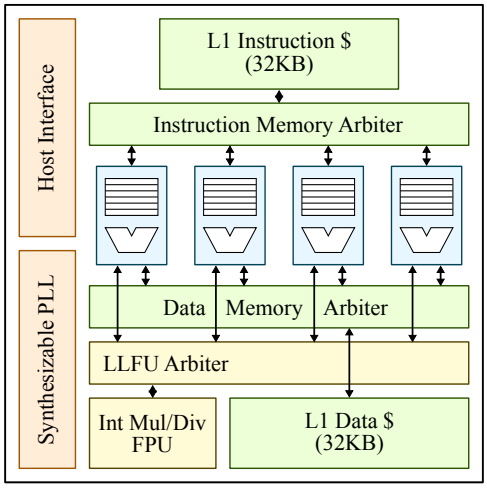


Figure 5.9: Block Diagram of BRGTC2 – The chip integrates four RISC-V RV32IMAF cores that support enough instructions to run a work-stealing runtime. The cores are arranged in a smart-sharing architecture [Sri18]. We paired the RISC-V ecosystem with productive open-source design tools to build this RISC-V system in TSMC 28 nm with seven graduate students in two months.

modify a software toolchain, open-source VLSI implementations of previously taped out RISC-V SoCs for reference including Rocket [AAB+16] and Celerity [DXT+18, AAHA+17, RZAH+19]).

Open-Source Software and ISA – The RISC-V software toolchain was tremendously useful as an out-of-the-box and standard solution for compiling applications for our system. In particular, we leveraged recent GCC support with options targeting RV32IMAF, and we were also able to use inline assembly in our in-house work-stealing runtime library to implement hints and to track stats. The RISC-V ISA itself was also a tremendous success. Because the ISA is designed as a small base set of instructions with modular extensions, we were able to apply an incremental design approach by writing RTL to first support the base set (i.e., RV32I), then add multiply/divide support (i.e., RV32IM), then add atomic support (i.e., RV32IMA), and finally add floating-point support (i.e., RV32IMAF). We also leveraged the control and status registers for many custom purposes including tracking stats.

Open-Source Cycle-Level Modeling – The gem5 simulator system [BBB+11] is a popular platform for simulator-based cycle-level modeling in the computer architecture research community. Multicore support has recently been added for RISC-V [TCB18], providing computer architects a critical tool for cycle-level design-space exploration of complex RISC-V systems. We leveraged RISC-V support on gem5 to explore our sharing architecture shown in Figure 5.9, which shares caching resources and long-latency functional units. We swept important parameters includ-

ing the latency to shared resources, the number of each resource to share, the impact of memory coalescing techniques, the size and capacity of caches and buffers, and the impact of various arbitration schemes.

Open-Source RTL Modeling – We paired the RISC-V ecosystem with our new Python-based hardware modeling framework, PyMTL [LZB14, JIB18], to build our RISC-V system. PyMTL leverages the Python programming language to create a highly productive and flexible environment for test, design, and composition in BRGTC2. *Testing* in PyMTL enables access to full-featured software testing frameworks built for Python (e.g., pytest [pyt14]), providing useful features including automatic test discovery, modular fixtures, and rich customizable plugins. We leveraged PyMTL support for two-state simulation and state initialization to one, zero, and random values. *Designing* in PyMTL was more accessible than in Verilog for students new to RTL design, avoiding many well-known quirks of the older language while also enabling a familiar style of debugging in Python. *Composition* in PyMTL enabled powerful multi-level co-simulation of functional-level, cycle-level, and RTL models. For example, to debug an issue with atomics, we swapped in a functional model of the cache to narrow the bug location down to other components. The PyMTL framework generates Verilog for our standard ASIC toolflow. Overall, we found the PyMTL framework to be a tremendous success for designing a RISC-V system from scratch with rigorous testing support.

Open-Source ASIC Flow Organization – The availability of high-quality, community-developed reference ASIC flows is a tremendously useful resource for both new and experienced chip designers. We designed our RISC-V silicon prototype using a modular VLSI build system[1], which we have open-sourced as a reference organization of the ASIC toolflow for architecture and VLSI researchers interested in silicon prototyping. One of the most challenging aspects of working with ASIC flows is managing the many moving pieces (e.g., PDK, physical IP libraries, ASIC-specific tools), which come from many different vendors and yet must still be made to work together coherently. Despite the great effort required to successfully assemble a working ASIC flow, teams typically end up with little reuse across projects as designers frequently tweak steps, target different technology nodes, or even use different vendors for physical IP. Furthermore, while architectural design-space exploration tends to require just a few stages (e.g., synthesis, simple floorplanning, no IO cells, need not be DRC or LVS clean), a full tapeout requires many more stages to guarantee

[1] Modular VLSI Build System: https://github.com/cornell-brg/alloy-asic

manufacturability. The key idea behind a modular VLSI build system is to avoid rigidly structured ASIC flows that cannot be repurposed, and to instead break the ASIC flow into modular steps that can be re-assembled into different flows. With a modular build system, architecture projects can omit detailed steps from a chip flow (e.g., use default floorplan, skip IO pads, skip DRC/LVS), while a VLSI project can include key steps for VLSI research (e.g., custom floorplanning for synchronizer research), while reusing the common steps in between. Our approach also includes the idea of an ASIC design kit, which is the specific set of physical backend files required to successfully build chips, as well as a unified and standard interface to those files. A well-defined interface enables swapping process and IP libraries without modification to the scripts that use them. Finally, this approach embraces plugins that hook into steps across the entire ASIC flow for design-specific customization. We have found that a modular ASIC flow enables productive reuse of steps across projects, and we have open-sourced our flow for reference.

Synthesizable Analog IP – Our RISC-V system is clocked by a synthesizable PLL that was first designed for use in the Celerity SoC [DXT+18, AAHA+17, RZAH+19], but has been adapted for use in a TSMC 28 nm process. Management of analog IP is traditionally a significant challenge in the design of a complex SoC. Mixed-signal crossings between analog and digital domains are well-known sources of costly design mistakes, and the communication between analog and digital design teams adds project management overhead that is nevertheless crucial to the overall success of the chip. Synthesizable analog IP is an approach that migrates blocks that are traditionally full-custom into the digital domain to mitigate these challenges. At a high level, the PLL relies on an array of internal ring oscillators with varying numbers of stages and configurable load capacitances (i.e., configurable NAND2 loads) controlled by a digital feedback loop. Our design experience using this PLL was a success, as we would not have been able to quickly design a PLL to generate programmable clocks from scratch with confidence. To test our design, we ported our PLL as a GDS from Cadence Innovus to Virtuoso, where we ran SPICE-level extracted simulations. The synthesizable PLL is planned to be open-sourced, which will provide architects an additional useful tool for silicon prototyping.

Final Thoughts – This chip follows in the spirit of AAWS in Chapter 3 as an investment in future exploration of hardware optimizations for task-based parallel runtimes. The insights I gained from this silicon prototyping experience also contributed to my interest in addressing the VLSI-specific research challenges in Chapter 4.

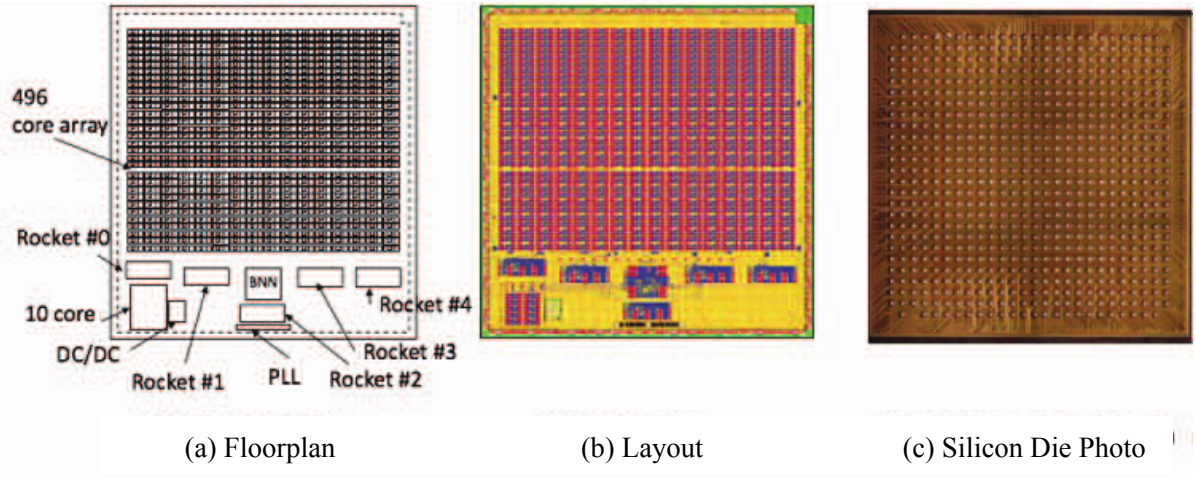
5.4 The Celerity Open-Source 511-Core RISC-V Tiered Accelerator Fabric

Celerity is a 5×5 mm 385 M-transistor SoC implemented in an advanced 16 nm technology. The tapeout was a multi-university effort funded through the DARPA Circuit Realization At Faster Timescales (CRAFT) program. Figure 5.10 shows an annotated chip plot and die photo of the chip. This chip appears in multiple venues, including Hot Chips: A Symposium on High Performance Chips [AAHA+17], the IEEE Symposium on VLSI Technology & Circuits [RZAH+19], and IEEE MICRO Top Picks from Hot Chips [DXT+18], for which I was a key contributor.

5.4.1 Research Purpose

The Celerity SoC is a DARPA-funded research vehicle for a range of productive hardware design and verification tools that are creating excitement in the hardware community including: synthesizable analog IP (synthesizable PLL, digital LDO), high-level synthesis (complex HLS-generated binarized neural network accelerator), Python-based hardware modeling (composition logic, BNN wrapper logic), techniques for area-efficient and high-bandwidth manycore networks (496-core RISC-V tiled manycore, remote-store programming model), and reuse from open-source libraries and generators (BaseJump IP, Chisel-generated RISC-V Rocket cores). A key achievement of the project was implementing the entire SoC in nine months with only twenty graduate students and faculty spread across four geographical locations and universities (Univerity of California at San Diego, University of Washington, University of Michigan, and Cornell University). As an architecture research chip, the Celerity SoC also represents a 16 nm 496-core RISC-V network-on-chip (NoC). The mesh achieves 1.4 GHz at 0.98 V, yielding a peak of 695 Giga RISC-V instructions/s (GRVIS) and a record 812,350 CoreMark benchmark score. The main feature is the NoC architecture, which uses only 1881 μm^2 per router node, enables highly scalable and dense compute, and provides up to 361 Tb/s of aggregate bandwidth.

In connection to my thesis, my experience working with the Celerity SoC was the greatest motivating factor for my work on ultra-elastic CGRAs in Chapter 4. Specifically, debugging asynchronous crossings is a challenging problem, and the majority of post-silicon bugs were related to these crossings. The Celerity SoC's large manycore array had the potential to be extremely energy-efficient, but the programmability challenges for utilizing the vast computational resources were



(a) Floorplan (b) Layout (c) Silicon Die Photo

Figure 5.10: Celerity SoC Chip Plot and Die Photo – A 5 × 5 mm 385 M-transistor SoC implemented in an advanced 16 nm technology.

significant. This pair of challenges motivated me to investigate (1) novel approaches to manage asynchronous crossings in the context of many small and configurable fine-grain workers, and (2) automatic hardware-managed scheduling as in elastic CGRAs.

Despite this, the architecture also motivates an interesting direction for future work. Chapter 4 focused on CGRA architecture, but many of the key research questions apply to massively parallel manycores, proposed both from industry (e.g. 64-core Tilera TILE64 [BEA+08], 48-core Intel SCC [int09], and 72-core Intel Knights Landing [SGC+16]), and academia (e.g., 25-core Piton [MFN+17], 511-core Celerity [DXT+18], 1000-core KiloCore [BSP+17] and 1024-core Adapteva Epiphany-V [Olo16]):

- **Key Question #1 –** How can we better deal with asynchronous crossings for very small workers which may be configured to exhibit fine-grain asymmetry (manycore tiles are more programmable than CGRA tiles)?
- **Key Question #2 –** How can we better utilize the computational resources in massively parallel manycore arrays for extreme energy-efficiency? The combination of tiny cores and small scratchpads is reminiscent of the CGRA context. However, manycores come with their own unique challenges (e.g., coherence, programmability).

A manycore array is very nearly an example of configured asymmetry with a general-purpose bent. “Configuration” in the manycore context corresponds to programming a sea of tiny cores.

In conclusion, the Celerity SoC closely relates to my thesis research. At the same time, the architecture motivates an interesting direction for future work. I now briefly describe my role in the project. As the Cornell University student lead, I was a key player in integrating all of the productive methodologies and tools into the final chip. I collaborated frequently with Prof. Dreslinski (Michigan) as well as with Prof. Taylor (Washington). In addition to leading the Rocket and BNN accelerator logical/physical design, I also contributed to project management and team building. I visited the University of Michigan as an academic exchange student for two months, where I worked with Prof. Dreslinski and made key contributions to physical design and verification. I developed our first power strategy as well as our first DRC/LVS-clean block layout. I developed our gate-level simulation infrastructure and contributed to the final top-level integration, LVS, DRC, power signoff, and timing closure.

CHAPTER 6
CONCLUSION

This thesis explored novel fine-grain power-control techniques to exploit fine-grain asymmetry in both the space and time dimensions to improve both performance and energy efficiency. Differentiating from previous work, I focused on a software, architecture, and VLSI co-design approach to provide control for these techniques using previously inaccessible information newly exposed across layers of abstraction. I explored the potential of specializing these fine-grain power-control techniques for productive and parallel software runtimes as well as for energy-efficient spatial architectures.

In this chapter, I summarize my thesis contributions and also my thoughts on their potential long-term impacts. Based on the insights in this thesis, I also describe potential future research directions.

6.1 Thesis Summary and Contributions

This thesis began with three high-level observations. With architects increasingly relying on parallelization and specialization, the first observation was that the combination of parallelism and specialization has steadily increased on-chip asymmetry in both the space and time dimensions. The second observation was that this on-chip asymmetry can result in widely varying utilization in space (i.e., across different components) and in time (i.e., used at different times across varying performance levels), motivating research on very fine-grain power-control techniques in order to power (or not power) different components to different levels at just the right times to significantly reduce waste. The final observation was that traditional walls of abstraction have broken down, allowing a cross-stack co-design approach across software, architecture, and VLSI to provide new, previously inaccessible information that can be used to control novel fine-grain power-control techniques. I then presented specific techniques and further investigated the contexts of productive task-based parallel runtimes as well as for coarse-grain reconfigurable arrays.

I first introduced reconfigurable power distribution networks, which realistically enable fine-grain voltage and frequency scaling techniques for homogeneous systems of little cores at microsecond timescales by leveraging recent work on fully integrated voltage regulation. Although on-chip regulators suffer from lower on-chip conversion efficiencies and on-die area overheads, I

demonstrated how careful software, architecture, and circuit co-design can mitigate those circuit-level challenges while also resolving architecture-level performance and energy bottlenecks in homogeneous multicores.

I then narrowed the scope to the domain of productive task-based parallel runtimes. I explored fine-grain voltage and frequency scaling for both big and little cores at microsecond timescales. I argued that work-stealing runtimes are a natural fit for managing asymmetry at the software level, and I showed how these software runtimes can be made aware of underyling asymmetry in the architecture and VLSI layers to create more efficient schedules and to dynamically tune processing elements.

Shifting focus to configured asymmetry in CGRAs, I explored fine-grain voltage and frequency scaling for each tile and memory subbank at reconfiguration timescales which vary from hundreds of nanoseconds to milliseconds. I built on previous research on elastic CGRAs (which use elasticity to reduce compiler complexity) and proposed ultra-elastic CGRAs, which capitalize on new opportunities in elastic CGRAs to enable support for configurable per-tile fine-grain power control and significantly improved dataflow efficiency.

Lastly, I described my work on four silicon prototypes to support various aspects of my thesis. Besides meeting many other research goals, these prototypes motivate future research on fine-grain power-control techniques in unexplored contexts. In particular, I showed how the Celerity SoC can have similar utilization challenges across its massively parallel manycore array as well as across its three-tier specialization fabric.

The primary contributions of this thesis are:

- A novel approach for fine-grain voltage and frequency scaling for homogeneous systems of little cores at microsecond timescales based on switched-capacitor-based integrated voltage regulators using a novel **dynamic capacitance sharing** technique.
- A novel approach for fine-grain power control for heterogeneous multicore systems at microsecond timescales specialized for **task-based parallel runtimes** using a set of three techniques based on balancing marginal utility.
- A novel proposal for ultra-elastic CGRAs which capitalize on new opportunities in elastic CGRAs, enabling support for configurable per-tile fine-grain power control and significantly improved dataflow efficiency.

- A deep design-space exploration of these ideas using a vertically integrated research methodology that in many cases extends from cycle-level modeling down to silicon prototyping.

6.2 Potential Long-Term Impacts: Reconfigurable Power-Distribution Networks

If I consider this work broadly, including beyond the scope of this thesis, then I believe one of the most important potential long-term impacts will be **encouraging the research community to pursue an architecture and analog circuit co-design approach to leverage the emerging trend towards integrated voltage regulation.** I also see both the specific FG-SYNC+ DVFS controller and the RPDN approach proposed in this work as having strong potential for long-term impact. In the rest of this section, I highlight what I feel are the three most important contributions of the work and the potential long-term impact of each contribution.

Contribution #1 – This work makes a strong case for architecture and analog circuit co-design to maximize the system-level benefit of the emerging trend towards integrated voltage regulation.

Potential Impact – Encouraging other researchers to think of integrated voltage regulation and the associated power distribution network as a critical subsystem that should no longer be ignored by architects, nor studied in isolation by analog circuit designers.

Recent technology trends suggest that we are entering a new era where it is now becoming feasible to reduce system cost by integrating switching regulators on-chip; however, it is difficult to design a system with integrated voltage regulation from solely an architecture or an analog circuit perspective. Architects tend to idealize concepts such as per-core voltage regulation without considering the intricacies that come with enabling it. Analog circuit designers tend to design regulators as stand-alone components in a one-size-fits-all approach, which provisions for the worst case and over-designs for the lightest loads. Architecture and analog circuit co-design enables control over the whole system and lays the groundwork of expertise necessary for maximizing the system-level benefit of a design. As an example, this work is based on the key insight that providing per-core voltage regulation with the traditional approach of multiple adjustable voltage regulators is significantly over-designed, since all cores can never be in the fastest operating mode

at once. This observation enables us to propose RPDNs for system-level benefits. I expect that there is much more research to be done on integrated voltage regulation and the associated power distribution network, and I encourage the architecture community to consider "architecting" this critical subsystem.

Contribution #2 – This work proposes a new DVFS controller called FG-SYNC+ that uses lightweight hints provided by software to improve performance and energy efficiency at similar average power.

Potential Impact – Motivate other researchers and/or industry to adopt the idea of hints to inform the hardware of fine-grain activity imbalance, and then to create new hint-based fine-grain DVFS controllers.

In this work, I instrument sychronization primitives in the threading library with (1) activity hints to inform the hardware which threads are doing useful work and (2) novel “work left” hints to convey the relative progress of each thread through the thread library’s `parallel_for` function. These hints form a lightweight abstraction between hardware and software that enables clean communication of application-level behavior to the DVFS controller while avoiding complicated hardware heuristics. The design space of hint-based fine-grain DVFS controllers is rich; in this work, I propose a novel DVFS controller called FG-SYNC+ that uses a lookup table to map dynamic, per-core activity information to per-core DVFS modes. Designing these lookup tables for multiple voltage levels and domains enables FG-SYNC+ to exploit fine-grain activity imbalance in multithreaded applications for improved performance and energy-efficiency at the same average power. Future hint-based fine-grain DVFS controllers may be uniquely positioned to flexibly balance the performance and energy efficiency requirements of arbitrary subsystems by leveraging information embedded in well-placed and well-designed hints.

Contribution #3 – This work proposes a novel approach to on-chip regulator design based on the idea of reconfigurable power distribution networks (RPDNs).

Potential Impact – Motivate other researchers and/or industry to explore new circuit techniques and designs that leverage (1) dynamically reconfigurable inter-core energy storage and (2) demand-based power delivery to reduce area overhead. voltage regulation viable.

RPDN is a novel approach to the design of integrated voltage regulators and the associated power distribution network. A shared energy storage of many "unit cells" is flexibly reconfigured through a switch fabric that, combined with per-core control circuitry, effectively enables multiple SC regulators to be created "on-demand". RPDN is the first example of integrated voltage regulator design using dynamically reconfigurable inter-core energy storage, and this work unfolds a potentially broad category of PDN design. There is significant room to explore the design space of integrated voltage regulators with dynamically reconfigurable inter-core energy storage, including (1) design with other types of regulators (e.g., buck converters, three-level converters), (2) alternative methods to partition shared energy storage, (3) design of new feedback control schemes for switching regulators specifically designed for reconfiguration, and (4) alternative methods for scaling the network to support larger numbers of cores. In this broad design space, RPDN is one example based on switched-capacitor regulators with equally partitioned energy storage; regulators have their own feedback control loops based on load profiles and configurations stored in lookup tables; and I chose a specific design approach for addressing scalability (i.e., partitioning the network into sub-RPDNs). I expect that there is more research to be done and I encourage other researchers to further explore this design space.

6.3 Potential Long-Term Impacts: Asymmetry-Aware Work-Stealing Runtimes

If I consider this work broadly, including beyond the scope of this thesis, then I believe one of the most important potential long-term impacts will be steering focus in the computer architecture research community away from thread-based parallel frameworks and toward task-based parallel frameworks such as work-stealing runtimes for multithreaded computer architecture research. I also see the specific methods proposed in this work that integrate the work-stealing runtime with novel hardware techniques as having strong potential for long-term impact. In the rest of this section, I highlight what I feel are the three most important contributions of the work and the potential long-term impact of each contribution.

Contribution #1 – This work identifies the potential for combining the areas of work-stealing runtimes, static asymmetry (i.e., single-ISA heterogeneous architectures), and dynamic asymmetry (i.e., DVFS).

Potential Impact – Encourage other researchers and/or industry to explore how these three open research areas interact.

This work lies at the intersection between three different areas, including work-stealing runtimes, static asymmetry in the form of single-ISA heterogeneous architectures, and dynamic asymmetry in the form of DVFS. Previous works have explored these areas pairwise. For example, Bender et al. [BR02] combines work-stealing runtimes and static asymmetry, studying the theoretical properties of work-stealing runtimes given a mix of workers with varying speeds. Ribic et al. [RL14] combines work-stealing runtimes and dynamic asymmetry, proposing to reduce the voltage/frequency of thieves and increase the voltage/frequency of workers with deep task queues. Azizi et al. [AML+10] studies the interplay between static and dynamic asymmetry by varying circuit and microarchitectural parameters and DVFS to perform an energy-performance cost-benefit analysis. To my knowledge, we are the first to explore and find benefit from the combination of these three areas, with the goal of exploring how to use asymmetry awareness to improve the performance and energy efficiency of a work-stealing runtime. I encourage other researchers and/or industry to further explore how these three open research areas interact.

The focus on task-based parallel runtimes is of particular importance. Previous researchers have spent significant effort addressing fine-grain thread imbalance for both symmetric and asymmetric systems using thread-based parallel programming models (e.g., pthreads). However, the *task-based programming model* has emerged as an elegant balance between programmer productivity and expression of parallelism while facilitating dynamic fine-grain load balancing, mitigating much of the prior challenges based on the thread-based programming model while opening new opportunities for optimization. In particular, work-stealing runtimes elegantly and naturally distribute work in systems with static asymmetry while preserving the balance in performance and energy efficiency across serial and parallel application regions. Computer architects should move away from thread-based programming models and focus instead on optimizing for productive task-based programming models.

Contribution #2 – This work presents three novel software/hardware techniques (*work-pacing*, *work-sprinting*, and *work-mugging*) that improve performance and energy efficiency of a work-stealing runtime in the presence of static and dynamic asymmetry.

Potential Impact – Motivate other researchers to explore further software/hardware co-design opportunities for work-stealing runtimes.

This work proposes building work-stealing runtimes that are aware of the underlying system's static and dynamic asymmetry. Awareness of this underlying asymmetry enables the work-stealing runtime to balance marginal utilities across big and little cores by applying three techniques: *work-pacing*, *work-sprinting*, and *work-mugging*. These techniques require only lightweight changes to the software and hardware, but would not be possible without a full-on software/hardware co-design approach. The experimental results are promising and suggest that work-stealing runtimes can provide a rich context for software/hardware co-design. For example, although this work focuses on DVFS, dynamic asymmetry is itself a rich space reaching beyond DVFS (e.g., hybrid big/little cores, reconfigurable core types, resizable intra-core structures). In addition, a wide range of potential optimization metrics (e.g., performance, power, real-time constraints) further widens the design space, increasing the potential for high impact in future multicore systems based on work-stealing runtimes.

Contribution #3 – This work proposes integrating the work-stealing runtime with the underlying hardware using lightweight software hints embedded in the runtime.

Potential Impact – Encourage other researchers to consider tighter integration between a work-stealing runtime and the underlying hardware.

Task-based parallel frameworks expose information that may be performance- or power-critical (e.g., which threads are in the steal loop, which threads have recently stolen work), but this information is often hidden from the hardware. More tightly integrating a work-stealing runtime and the underlying hardware can expose opportunities for performance or power optimization that are unique to the specific but widely used paradigm of work-stealing. For example, knowledge of which threads have recently stolen work can potentially enable locality optimizations within caches, which may be especially relevant for large systems with many cores. This work proposes

using lightweight software hints embedded in the runtime to inform the hardware when threads are in the steal loop and not executing useful work. This scheme provides the DVFS controller information to decide how to allocate power across cores. I encourage other researchers to explore the benefits of tightly integrating work-stealing runtimes with the underlying hardware.

6.4 Potential Long-Term Impacts: Ultra-Elastic Coarse-Grain Reconfigurable Arrays

The most evident potential impact of this work is a clear pull towards closer integration starting from the compiler, through architecture, and down to VLSI. Traditionally, compiler awareness extends only to the underlying architecture with general-purpose compilers targeting ISAs and spatial compilers targeting cycle-level architectural behavior. As this work demonstrates, breaking an additional layer of abstraction can provide significant benefits to dataflow efficiency, making this a timely work in the context of recently renewed interest in dataflow for reducing data-movement energy. I also see the specific methods proposed in this work that enable per-tile fine-grain DVFS for CGRAs as having strong potential for long-term impact. In the rest of this section, I highlight what I feel are the three most important contributions of the work and the potential long-term impact of each contribution.

Contribution #1 – This work makes a strong case for compilers aware of not only architecture-level but also VLSI-level opportunities and challenges.

Potential Impact – Encourage other researchers and/or industry to further explore how the compiler can more closely interact across abstraction layers to extract system-level benefit.

An increasing body of work focuses on reducing not only computational energy but also data-movement energy [CKES17, PSC+19]. Energy efficiency in both cases is intimately and non-trivially tied to VLSI, in particular with energy proportional to the square of the voltage and frequency similarly non-trivial in the presence of high leakage. While the compiler has traditionally been abstracted from VLSI (e.g., for general-purpose computing), architects focusing on specialization today must carefully orchestrate data to execute on the most energy-efficient compute units and to match the most frequent data accesses with the most energy-efficient storage [PSC+19].

This demands careful decision making at the compiler level (i.e., determines where data lives), at the architecture level (i.e., determines which units and which types of storage are available), and at the VLSI level (i.e., determines the actual efficiency of each unit and storage element). This work brings voltage and frequency to attention as first-class citizens in the compiler flow, representing a step towards closer integration across the compiler, the architecture, and VLSI. We encourage computer architects to closely investigate how compilers can make use of new information and new opportunities across abstractions.

Contribution #2 – This work presents ultra-elastic CGRAs, which enable more efficient dataflow by capitalizing on new opportunities available for elastic CGRAs.

Potential Impact – Motivate other researchers to explore new opportunities and new hardware techniques that can capitalize on latency-insensitive interfaces in elastic CGRAs.

Previous works introduced elastic CGRAs primarily to reduce compiler complexity. However, elastic CGRAs also open new opportunities to explore hardware techniques that can capitalize on their latency-insensitive interfaces. Latency-insensitive interfaces are well-known to enable very robust architectures that are "correct by construction" [KKV+18, CMSV01]. It is therefore a significant step to take traditional inelastic CGRAs (which are extremely dependent on latency for correctness) and to relax this requirement. This work capitalizes on this robustness to enable configurable voltage and frequency scaling, a technique which would have destroyed correctness on a traditional inelastic CGRA. We introduced the novel ultra-elastic CGRA computational model with innovation across the computing stack (analytical model, compiler, architecture, and VLSI) to enable configurable per-tile fine-grain DVFS with reasonable overheads. Although this work focuses primarily on fine-grain DVFS, the potential for latency insensitivity opens the CGRA context to many new opportunities that were previously infeasible but are now possible without violating correctness. We hope that other researchers will find inspiration from our approach.

Contribution #3 – This work acknowledges exponentially rising NREs and prioritizes design techniques that can be robustly verified.

Potential Impact – Encourage other researchers to recognize the impact of design and verification complexity on design cost for new hardware techniques.

Exponentially rising NRE costs are motivating a greater focus on verification challenges [KZVT17]. Many architecture works are not designed with these challenges in mind. In this work we specifically do *not* use asynchronous queues as is assumed in most other literature on fine-grain DVFS [SAD+02, YCZ12, JTH+13] for this reason. From the perspective of verification, constraining and verifying asynchronous clock-domain crossings is a tremendous challenge in a modern design flow and requires specialized expertise, methodologies, and verification tools [Cum08]. From the perspective of functionality, adding asynchronous queues at tile granularities fundamentally increases the risk of corrupted messages combinatorially (i.e., with the number of additional asynchronous crossings), even for the most rigorously verified implementations. Finally, from the perspective of performance, asynchronous queues add two-to-three-cycle synchronization latency penalties which can significantly reduce performance in the context of a CGRA. Note that synchronization latencies in asynchronous queues only appear when the queues are empty or full, but this is precisely the common case in the context of a CGRA. I encourage other researchers to explore the impact of new hardware techniques on design and verification complexity in addition to the traditional focus on performance and energy efficiency.

6.5 Future Work

The ideas in this thesis are just a few steps towards the vision of flexible and efficient fine-grained power-control mechanisms. There are a number of ways to take the ideas in this thesis forward.

Extending Information Across Abstractions – The asymmetry-aware work-stealing runtime chapter showed how application-level information can make a tremendous impact on power and performance, even if just a single bit of information is exposed (and only exploited by a single technique). Moving forward, it may be worth investigating systematic approaches to extend more information across layers in such a way that enables traditional execution (i.e., non-specialized architectures execute without penalty) while still allowing specialized techniques to easily access useful information. While it is not a focus of my thesis, my previous work on Loop-Task Accelerators [KJT+17] was an inspiring example of lowering the abstraction of loop-tasks from the software runtime down to the hardware, providing direct and unambiguous semantics for acceleration.

Critical-Path-Aware Ultra-Elastic CGRAs – The work presented in Chapter 4 made the most conservative assumptions about the critical path of the tile architecture. Specifically, it assumed that every configurable operation had a critical path of the same length (i.e., the worst case). However, different operations can have very different critical paths. For example, the critical path of a routing tile is very short compared to the critical path of a multiply tile. If the compiler were to be made aware of these opportunities, it may be possible to aggressively increase the frequency of tiles while running at (counterintuitively) *lower* voltages.

Exploiting Fine-Grain Asymmetry in Software-Coherent Manycores – As alluded to in Chapter 5, many of the key research questions in Chapter 4 also apply to massively parallel manycores, proposed both from industry (e.g. 64-core Tilera TILE64 [BEA+08], 48-core Intel SCC [int09], and 72-core Intel Knights Landing [SGC+16]), and academia (e.g., 25-core Piton [MFN+17], 511-core Celerity [DXT+18], 1000-core KiloCore [BSP+17] and 1024-core Adapteva Epiphany-V [Olo16]). These manycores exhibit configured asymmetry in which the dataflow across cores is not evenly balanced and in which all cores may not even be configured to do useful work. Fine-grain power control is applicable here, but there are key challenges in managing manycores that must be answered in parallel, including addressing how data is loaded, how coherence is managed, and how applications can efficiently be mapped to these substrates in the first place. In terms of physical design, one key question is how to combine the capability for fine-grain power control with a tile-based design approach, which in my experience with the Celerity SoC significantly reduces design spin time. This would have to be done in such a way as to not add additional latency between tiles due to synchronization, to avoid increasing leakage, to mitigate area overheads from margins between core islands, and other considerations.

Exploiting Fine-Grain Asymmetry in CNN accelerators – The work in Chapter 4 focused on CGRAs, which bear resemblance to many well-known spatial accelerators that target CNNs, ranging from well-known existing hardware like Eyeriss [CKES17] up to the wafer-scale deep learning chips built by Cerebras [ea19]. CGRAs are a compelling option for acceleration not only because of the traits of spatial architectures (e.g., reducing data movement energy, high reuse), but also because of the flexibility of the substrate. It is possible to map many more applications to a CGRA than to a CNN accelerator. Despite the flexibility penalty, CNNs have made an outsized impact on the computing landscape today, and it may be worthwhile to investigate how fine-grain power-control techniques can apply to CNN-specific spatial architectures.

Power-Amorphous Architectures – The majority of the literature on fine-grain power control adapts the power delivery *to* an architecture. For example, we may configure a power-hungry accelerator and then allocate power to meet its expected demands. This requires us to budget (with margin) and add up budgets until we hit the total power cap. It may be worthwhile to think about conducting this process in the reverse: first allocating power and then having the architecture naturally adapt to its new bounds. The primary reason why I find this interesting is that as the number of asymmetric on-chip components grows, the act of summing many small power budgets and estimating each of their margins quickly becomes a very complex task. If instead we were to distribute power from the available pool and have architectures naturally reconfigure, speed up, or slow down without dedicated attention, this challenge could become much more manageable.

BIBLIOGRAPHY

[AAB+16] K. Asanović, R. Avižienis, J. Bachrach, S. Beamer, D. Biancolin, C. Celio, H. Cook, P. Dabbelt, J. Hauser, A. Izraelevitz, S. Karandikar, B. Keller, D. Kim, J. Koenig, Y. Lee, E. L. andMartin Maas, A. Magyar, H. Mao, M. Moreto, A. Ou, D. Patterson, B. Richards, C. Schmidt, S. Twigg, H. Vo, and A. Waterman. The Rocket Chip Generator. Technical report, EECS Department, University of California, Berkeley, Apr 2016.

[AAHA+17] T. Ajayi, K. Al-Hawaj, A. Amarnath, S. Dai, S. Davidson, P. Gao, G. Liu, A. Lotfi, J. Puscar, A. Rao, A. Rovinski, L. Salem, N. Sun, C. Torng, L. Vega, B. Veluri, X. Wang, S. Xie, C. Zhao, R. Zhao, C. Batten, R. G. Dreslinski, I. Galton, R. K. Gupta, P. P. Mercier, M. Srivastava, M. B. Taylor, and Z. Zhang. Celerity: An Open Source RISC-V Tiered Accelerator Fabric. *Symp. on High Performance Chips (Hot Chips)*, Aug 2017.

[AGS05] M. Annavaram, E. Grochowski, and J. Shen. Mitigating Amdahl's Law through EPI Throttling. *Int'l Symp. on Computer Architecture (ISCA)*, Jun 2005.

[AKK+13] T. Andersen, F. Krismer, J. Kolar, T. Toifl, C. Menolfi, L. Kull, T. Morf, M. Kossel, M. Brandli, P. Buchmann, and P. Francese. A 4.6 W/mm^2 Power Density 86% Efficiency On-Chip Switched Capacitor DC-DC Converter in 32-nm SOI CMOS. *Applied Power Electronics Conf. and Exposition*, Mar 2013.

[AML+10] O. Azizi, A. Mahesri, B. C. Lee, S. J. Patel, and M. Horowitz. Energy-performance Tradeoffs in Processor Architecture and Circuit Design: A Marginal Cost Analysis. *Int'l Symp. on Computer Architecture (ISCA)*, Jun 2010.

[ARKK13] A. Annamalai, R. Rodrigues, I. Koren, and S. Kundu. An Opportunistic Prediction-Based Thread Scheduling to Maximize Throughput/Watt in AMPs. *Int'l Conf. on Parallel Architectures and Compilation Techniques (PACT)*, Sep 2013.

[ASL+09] M. Alimadadi, S. Sheikhaei, G. Lemieux, S. Mirabbasi, W. Dunford, and P. Palmer. A Fully Integrated 660 MHz Low-Swing Energy-Recycling DC-DC Converter. *IEEE Trans. on Power Electronics (TPEL)*, 24(6):1475–1485, 2009.

[ASSK11] R. A. Abdallah, P. S. Shenoy, N. R. Shanbhag, and P. T. Krein. System Energy Minimization via Joint Optimization of the DC-DC Converter and the Core. *Int'l Symp. on Low-Power Electronics and Design (ISLPED)*, Jun 2011.

[Bat10] C. Batten. *Simplified Vector-Thread Architectures for Flexible and Efficient Data-Parallel Accelerators*. Ph.D. Thesis, MIT, 2010.

[BBB+11] N. Binkert, B. M. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, and D. A. Wood. The gem5 Simulator. *SIGARCH Computer Architecture News (CAN)*, 39(2):1–7, Aug 2011.

[BEA+08] S. Bell, B. Edwards, J. Amann, R. Conlin, K. Joyce, V. Leung, J. MacKay, M. Reif, L. Bao, J. Brown, M. Mattina, C.-C. Miao, C. Ramey, D. Wentzlaff, W. Anderson, E. Berger, N. Fairbanks, D. Khan, F. Montenegro, J. Stickney, and J. Zook. TILE64 Processor: A 64-Core SoC with Mesh Interconnect. *Int'l Solid-State Circuits Conf. (ISSCC)*, Feb 2008.

[BHME18] I. Bae, B. Harris, H. Min, and B. Egger. Auto-Tuning CNNs for Coarse-Grained Reconfigurable Array-Based Accelerators. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, Jul 2018.

[BJK+96] R. D. Blumofe, C. F. Joerg, B. C. Kuszmaul, C. E. Leiserson, K. H. Randall, and Y. Zhou. Cilk: An Efficient Multithreaded Runtime System. *Journal of Parallel and Distributed Computing (JPDC)*, 37(1):55–69, Aug 1996.

[BKSL08] C. Bienia, S. Kumar, J. P. Singh, and K. Li. The PARSEC Benchmark Suite: Characterization and Architectural Implications. *Int'l Conf. on Parallel Architectures and Compilation Techniques (PACT)*, Oct 2008.

[BL99] R. D. Blumofe and C. E. Leiserson. Scheduling Multithreaded Computations by Work Stealing. *Journal of the ACM*, 46(5):720–748, Sep 1999.

[BM09] A. Bhattacharjee and M. Martonosi. Thread Criticality Predictors for Dynamic Performance, Power, and Resource Management in Chip Multiprocessors. *Int'l Symp. on Computer Architecture (ISCA)*, Jun 2009.

[BPSB00] T. Burd, T. Pering, A. Stratakos, and R. Brodersen. A Dynamic Voltage Scaled Microprocessor System. *IEEE Journal of Solid-State Circuits (JSSC)*, 35(11):1571–1580, Nov 2000.

[BR02] M. A. Bender and M. O. Rabin. Online Scheduling of Parallel Programs on Heterogeneous Systems with Applications to Cilk. *Theory of Computing Systems (TCS)*, 35(3):289–304, Jun 2002.

[BS84] F. W. Burton and M. R. Sleep. Executing Functional Programs on a Virtual Tree of Processors. *Conf. on Functional Programming Languages and Computer Architecture*, Aug 1984.

[BSP+17] B. Bohnenstiehl, A. Stillmaker, J. J. Pimentel, T. Andreas, B. Liu, A. T. Tran, E. Adeagbo, and B. M. Baas. KiloCore: A 32-nm 1000-Processor Computational Array. *IEEE Journal of Solid-State Circuits (JSSC)*, 52(4):891–902, Apr 2017.

[BTG+17] I. Bukreyev, C. Torng, W. Godycki, C. Batten, and A. Apsel. Four Monolithically Integrated Switched-Capacitor DC-DC Converters with Dynamic Capacitance Sharing in 65-nm CMOS. *IEEE Transactions on Circuits and Systems I (TCAS-I)*, Nov 2017.

[CC06] B. H. Calhoun and A. Chandrakasan. Ultra-Dynamic Voltage Scaling Using Sub-Threshold Operation and Local Voltage Dithering. *IEEE Journal of Solid-State Circuits (JSSC)*, 41(1):238–245, Jan 2006.

[CCHG12] Q. Chen, Y. Chen, Z. Huang, and M. Guo. WATS: Workload-Aware Task Scheduling in Asymmetric Multi-core Architectures. *Int'l Parallel and Distributed Processing Symp. (IPDPS)*, May 2012.

[CCK07] Y. Choi, N. Chang, and T. Kim. DC-DC Converter-Aware Power Management for Low-Power Embedded Systems. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, 26(8):1367–1381, Aug 2007.

[CGR+08] Q. Cai, J. González, R. Rakvic, G. Magklis, P. Chaparro, and A. González. Meeting Points: Using Thread Criticality to Adapt Multicore Hardware to Parallel Regions. *Int'l Conf. on Parallel Architectures and Compilation Techniques (PACT)*, Oct 2008.

[CH09] J.-M. Chabloz and A. Hemani. A flexible communication scheme for rationally-related clock frequencies. *Int'l Conf. on Computer Design (ICCD)*, Oct 2009.

[CH10a] J.-M. Chabloz and A. Hemani. Distributed DVFS Using Rationally-Related Frequencies and Discrete Voltage Levels. *Int'l Symp. on Low-Power Electronics and Design (ISLPED)*, Aug 2010.

[CH10b] J.-M. Chabloz and A. Hemani. Lowering the latency of interfaces for rationally-related frequencies. *Int'l Conf. on Computer Design (ICCD)*, Oct 2010.

[CH13] J.-M. Chabloz and A. Hemani. Low-Latency Maximal-Throughput Communication Interfaces for Rationally Related Clock Domains. *IEEE Trans. on Very Large-Scale Integration Systems (TVLSI)*, Apr 2013.

[Cha84] D. M. Chapiro. *Globally-Asynchronous Locally-Synchronous Systems*. Ph.D. Thesis, Computer Science Department, Stanford, 1984.

[Cha12] J.-M. Chabloz. *Globally-Ratiochronous, Locally-Synchronous Systems*. Ph.D. Thesis, Electronic and Computer Systems Department, KTH, 2012.

[CHM+01] L. T. Clark, E. J. Hoffman, J. Miller, M. Biyani, Y. Liao, S. Strazdus, M. Morrow, K. E. Velarde, and M. A. Yarch. An Embedded 32-b Microprocessor Core for Low-Power and High-Performance Applications. *IEEE Journal of Solid-State Circuits (JSSC)*, 36(11):1599–1608, Nov 2001.

[CIOQ15] L. Costero, F. D. Igual, K. Olcoz, and E. S. Quintana-Ortí. Revisiting Conventional Task Schedulers to Exploit Asymmetry in ARM big.LITTLE Architectures for Dense Linear Algebra. *CoRR arXiv:1509.02058*, Sep 2015.

[CKES17] Y.-H. Chen, T. Krishna, J. S. Emer, and V. Sze. Eyeriss - An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks. *IEEE Journal of Solid-State Circuits (JSSC)*, 52(1):127–138, Jan 2017.

[CL05] D. Chase and Y. Lev. Dynamic Circular Work-stealing Deque. *Symp. on Parallel Algorithms and Architectures (SPAA)*, Jun 2005.

[CM08] G. Contreras and M. Martonosi. Characterizing and Improving the Performance of Intel Threading Building Blocks. *Int'l Symp. on Workload Characterization (IISWC)*, Sep 2008.

[CMJ+10] L. Chang, R. K. Montoye, B. L. Ji, A. J. Weger, K. G. Stawiasz, and R. H. Dennard. Fully-Integrated Switched-Capacitor 2:1 Voltage Converter w/ Regulation Capability & 90% Efficiency at 2.3A/mm^2. *Symp. on Very Large-Scale Integration Circuits (VLSIC)*, Jun 2010.

[CMSV01] L. P. Carloni, K. L. McMillan, and A. L. Sangiovanni-Vincentelli. Theory of Latency-Insensitive Design. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, Sep 2001.

[CRB+15] K. Chronaki, A. Rico, R. M. Badia, E. Ayguadé, J. Labarta, and M. Valero. Criticality-Aware Dynamic Task Scheduling for Heterogeneous Architectures. *Int'l Symp. on Supercomputing (ICS)*, Jun 2015.

[Cum08] C. E. Cummings. Clock domain crossing (cdc) design & verification techniques using systemverilog. *Synopsys Users Group (SNUG)*, 2008.

[Dem13] M. Demler. MediaTek Steps Up to Tablets: MT8135 Brings Heterogeneous Multiprocessing to Big.Little. *Microprocessor Report (MPR)*, Aug 2013.

[DGP+12] R. G. Dreslinski, B. Giridhar, N. Pinckney, D. Blaauw, D. Sylvester, and T. Mudge. Reevaluating Fast Dual-Voltage Power Rail Switching Circuitry. *Workshop on Duplicating, Deconstructing, and Debunking (WDDD)*, Jun 2012.

[DGY+74] R. H. Dennard, F. H. Gaensslen, H.-N. Yu, V. L. Rideout, E. Bassous, and A. R. LeBlanc. Design of Ion-Implanted MOSFET's with Very Small Physical Dimensions. *IEEE Journal of Solid-State Circuits (JSSC)*, 9(5):256–268, Oct 1974.

[DLS+09] J. Dinan, D. B. Larkins, P. Sadayappan, S. Krishnamoorthy, and J. Nieplocha. Scalable Work Stealing. *Int'l Conf. on High Performance Networking and Computing (Supercomputing)*, Nov 2009.

[DM06] J. Donald and M. Martonosi. Techniques for Multicore Thermal Management: Classification and New Exploration. *Int'l Symp. on Computer Architecture (ISCA)*, Jun 2006.

[DMS+13] W. Deng, A. Musa, T. Siriburanon, M. Miyahara, K. Okada, and A. Matsuzawa. 0.022 mm^2 970 uW Dual-Loop Injection-Locked PLL with -243dB FOM Using Synthesizable All-Digital PVT Calibration Circuits. *Int'l Solid-State Circuits Conf. (ISSCC)*, Feb 2013.

[DR07] G. Dhiman and T. Rosing. Dynamic Voltage Frequency Scaling for Multi-Tasking Systems Using Online Learning. *Int'l Symp. on Low-Power Electronics and Design (ISLPED)*, Aug 2007.

[Dre11] R. G. Dreslinski. *Near-Threshold Computing: From Single-Core to Many-Core Energy-Efficient Architectures*. Ph.D. Thesis, EECS Department, University of Michigan, 2011.

[DWB+10] R. G. Dreslinski, M. Wieckowski, D. Blaauw, D. Sylvester, and T. Mudge. Near-Threshold Computing: Reclaiming Moore's Law Through Energy-Efficient Integrated Circuits. *Proc. of the IEEE*, 98(2):253–266, Feb 2010.

[DXT+18] S. Davidson, S. Xie, C. Torng, K. Al-Hawaj, A. Rovinski, T. Ajayi, L. Vega, C. Zhao, R. Zhao, S. Dai, A. Amarnath, B. Veluri, P. Gao, A. Rao, G. Liu, R. K. Gupta, Z. Zhang, R. G. Dreslinski, C. Batten, and M. B. Taylor. The Celerity Open-Source 511-Core RISC-V Tiered Accelerator Fabric: Fast Architectures and Design Methodologies for Fast Chips. *IEEE Micro*, Mar 2018.

[ea19] S. L. et al. Wafer-Scale Deep Learning. *Symp. on High Performance Chips (Hot Chips)*, Aug 2019.

[ECF96] C. Ebeling, D. C. Cronquist, and P. Franklin. RaPiD: Reconfigurable Pipelined Datapath. *Int'l Conf. on Field Programmable Logic (FPL)*, Sep 1996.

[EE11] S. Eyerman and L. Eechkout. Fine-Grained DVFS Using On-Chip Regulators. *ACM Trans. on Architecture and Code Optimization (TACO)*, 8(1):1:1–1:24, Apr 2011.

[FBB+15] E. Fluhr, S. Baumgartner, D. Boerstler, J. Bulzacchelli, T. Diemoz, D. Dreps, G. English, J. Friedrich, A. Gattiker, T. Gloekler, C. Gonzalez, J. Hibbeler, K. Jenkins, Y. Kim, P. Muench, R. Nett, J. Paredes, J. Pille, D. Plass, P. Restle, R. Robertazzi, D. Shan, D. Siljenberg, M. Sperling, K. Stawiasz, G. Still, Z. Toprak-Deniz, J. Warnock, G. Wiedemeier, and V. Zyuban. The 12-Core POWER8 Processor With 7.6 Tb/s IO Bandwidth, Integrated Voltage Regulation, and Resonant Clocking. *IEEE Journal of Solid-State Circuits (JSSC)*, 50(1):10–23, Jan 2015.

[FDD+06] T. Fischer, J. Desai, B. Doyle, S. Naffziger, and B. Patella. A 90-nm Variable Frequency Clock System for a Power-Managed Itanium Architecture Processor. *IEEE Journal of Solid-State Circuits (JSSC)*, 41(1):218–228, Jan 2006.

[FLR98] M. Frigo, C. E. Leiserson, and K. H. Randall. The Implementation of the Cilk-5 Multithreaded Language. *ACM SIGPLAN Conf. on Programming Language Design and Implementation (PLDI)*, Jun 1998.

[FWC+18] X. Fan, D. Wu, W. Cao, W. Luk, and L. Wang. Stream Processing Dual-Track CGRA for Object Inference. *IEEE Trans. on Very Large-Scale Integration Systems (TVLSI)*, Feb 2018.

[GC97] V. Gutnik and A. P. Chandrakasan. Embedded Power Supply for Low-Power DSP. *IEEE Trans. on Very Large-Scale Integration Systems (TVLSI)*, 5(4):425–435, Dec 1997.

[GHN+12] V. Govindaraju, C.-H. Ho, T. Nowatzki, J. Chhugani, N. Satish, K. Sankaralingam, and C. Kim. DySER: Unifying Functionality and Parallelism Specialization for Energy Efficient Computing. *IEEE Micro*, 33(5), Sep/Oct 2012.

[GHS11] V. Govindaraju, C.-H. Ho, and K. Sankaralingam. Dynamically Specialized Datapaths for Energy-Efficient Computing. *Int'l Symp. on High-Performance Computer Architecture (HPCA)*, Feb 2011.

[GK16] M. Gao and C. Kozyrakis. HRL - Efficient and flexible reconfigurable logic for near-data processing. *Int'l Symp. on High-Performance Computer Architecture (HPCA)*, Apr 2016.

[GOK+06] F. K. Gurkaynak, S. Oetiker, H. Kaeslin, N. Felber, and W. Fichtner. GALS at ETH Zurich: success or failure? *Int'l Symposium on Asynchronous Circuits and Systems (ASYNC)*, Mar 2006.

[Gre11] P. Greenhalgh. Big.LITTLE Processing with ARM Cortex-A15 & Cortex-A7. *EE Times*, Oct 2011.

[GRSW04] E. Grochowski, R. Ronen, J. Shen, and H. Wang. Best of Both Latency and Throughput. *Int'l Conf. on Computer Design (ICCD)*, Oct 2004.

[GSA14] W. Godycki, B. Sun, and A. Apsel. Part-Time Resonant Switching for Light Load Efficiency Improvement of a 3-Level Fully Integrated Buck Converter. *European Solid-State Circuits Conf. (ESSCIRC)*, Sep 2014.

[GSSK12] H. Ghasemi, A. Sinkar, M. Schulte, and N. S. Kim. Cost-Effective Power Delivery to Support Per-Core Voltage Domains for Power-Constrained Processors. *Design Automation Conf. (DAC)*, Jun 2012.

[GTB+14] W. Godycki, C. Torng, I. Bukreyev, A. Apsel, and C. Batten. Enabling Realistic Fine-Grain Voltage Scaling with Reconfigurable Power Distribution Networks. *Int'l Symp. on Microarchitecture (MICRO)*, Dec 2014.

[Gwe13] L. Gwennap. Renesas Mobile Goes Big (and Little). *Microprocessor Report (MPR)*, Feb 2013.

[Gwe14a] L. Gwennap. Qualcomm Tips Cortex-A57 Plans: Snapdragon 810 Combines Eight 64-Bit CPUs, LTE Baseband. *Microprocessor Report (MPR)*, Apr 2014.

[Gwe14b] L. Gwennap. Samsung First with 20 nm Processor. *Microprocessor Report (MPR)*, Sep 2014.

[Gwe19] L. Gwennap. An Inside Look at AI Accelerators. *Microprocessor Report (MPR)*, Aug 2019.

[HIT+13] Y. Huang, P. Ienne, O. Temam, Y. Chen, and C. Wu. Elastic CGRAs. *Int'l Symp. on Field Programmable Gate Arrays (FPGA)*, Feb 2013.

[HKB+05] P. Hazucha, T. Karnik, B. Bloechel, C. Parsons, D. Finan, and S. Borkar. Area-Efficient Linear Regulator With Ultra-Fast Load Regulation. *IEEE Journal of Solid-State Circuits (JSSC)*, 40(4):933–940, Apr 2005.

[HM07] S. Herbert and D. Marculescu. Analysis of dynamic voltage/frequency scaling in chip-multiprocessors. *Int'l Symp. on Low-Power Electronics and Design (ISLPED)*, Aug 2007.

[HM08] M. Hill and M. Marty. Amdahl's Law in the Multicore Era. *IEEE Computer*, 41(7):33–38, Jul 2008.

[Hor14] M. Horowitz. Computing's energy problem (and what we can do about it). *Int'l Solid-State Circuits Conf. (ISSCC)*, Feb 2014.

[HSH+05] P. Hazucha, G. Schrom, J. Hahn, B. A. Bloechel, P. Hack, G. E. Dermer, S. Narendra, D. Gardner, T. Karnik, V. De, and S. Borkar. A 233-MHz 80–87% Efficient Four-Phase DC-DC Converter Utilizing Air-Core inductors on Package. *IEEE Journal of Solid-State Circuits (JSSC)*, 40(4):838–845, Apr 2005.

[HSN+07] G. Huang, D. Sekar, A. Naeemi, K. Shakeri, and J. Meindl. Compact Physical Models for Power Supply Noise and Chip/Package Co-Design of Gigascale Integration. *Electronic Components and Technology Conf. (ECTC)*, May 2007.

[IBC+06] C. Isci, A. Buyuktosunoglu, C. Cher, P. Bose, and M. Martonosi. An Analysis of Efficient Multi-Core Global Power Management Policies: Maximizing Performance for a Given Power Budget. *Int'l Symp. on Microarchitecture (MICRO)*, Dec 2006.

[IM02] A. Iyer and D. Marculescu. Power and performance evaluation of globally asynchronous locally synchronous processors. *Int'l Symp. on Computer Architecture (ISCA)*, May 2002.

[IM06] C. Isci and M. Martonosi. Live, Runtime Phase Monitoring and Prediction on Real Systems with Application to Dynamic Power Management. *Int'l Symp. on Microarchitecture (MICRO)*, Dec 2006.

[int09] Intel Single-chip Cloud Computer. Online Webpage, 2009 (accessed Dec 4, 2009). `http://techresearch.intel.com/articles/tera-scale/1826.htm`.

[JB02] N. Jovanović and M. A. Bender. Task Scheduling in Distributed Systems by Work Stealing and Mugging – A Simulation Study. *Int'l Conf. on Information Technology Interfaces (ITI)*, Jun 2002.

[JBH+13] S. M. A. H. Jafri, O. Bag, A. Hemani, N. Farahini, K. Paul, J. Plosila, and H. Tenhunen. Energy-aware coarse-grained reconfigurable architectures using dynamically reconfigurable isolation cells. *Int'l Symp. on Quality Electronic Design (ISQED)*, Mar 2013.

[JIB18] S. Jiang, B. Ilbeyi, and C. Batten. Mamba: closing the performance gap in productive hardware development frameworks. *Design Automation Conf. (DAC)*, Jun 2018.

[JLB+15] R. Jevtic, H. Le, M. Blagojevic, S. Bailey, K. Asanovic, E. Alon, and B. Nikolic. Per-Core DVFS with Switched-Capacitor Converters for Energy Efficiency in Many-core Processors. *IEEE Trans. on Very Large-Scale Integration Systems (TVLSI)*, 23(4):723–730, Apr 2015.

[JSMP12] J. A. Joao, M. A. Suleman, O. Mutlu, and Y. N. Patt. Bottleneck Identification and Scheduling in Multi-Threaded Applications. *Int'l Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, Mar 2012.

[JSMP13] J. A. Joao, M. A. Syleman, O. Mutlu, and Y. N. Patt. Utility-Based Acceleration of Multithreaded Applications on Asymmetric CMPs. *Int'l Symp. on Computer Architecture (ISCA)*, Jun 2013.

[JTH+13] S. M. A. H. Jafri, M. A. Tajammul, A. Hemani, K. Paul, J. Plosila, and H. Tenhunen. Energy-aware-task-parallelism for efficient dynamic voltage, and frequency scaling, in CGRAs. *Int'l Conf. on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS)*, Jul 2013.

[Kan13] D. Kanter. Haswell's FIVR Extends Battery Life. *Microprocessor Report (MPR)*, Jun 2013.

[Kan17] D. Kanter. Ryzen Returns AMD to the Desktop. Microprocessor Report, The Linley Group, Mar 2017. `http://www.linleygroup.com/mpr/article.php?id=11775`.

[KBW12] W. Kim, D. Brooks, and G.-Y. Wei. A Fully-Integrated 3-Level DC-DC Converter for Nanosecond-Scale DVFS. *IEEE Journal of Solid-State Circuits (JSSC)*, 47(1):206–219, Jan 2012.

[KCC+12] C. Kim, M. Chung, Y. Cho, M. Konijnenburg, S. Ryu, and J. Kim. ULP-SRP: Ultra low power Samsung Reconfigurable Processor for biomedical applications. *International Conference on Field-Programmable Technology (FPT)*, Dec 2012.

[KCZ+17] B. Keller, M. Cochet, B. Zimmer, J. Kwak, A. Puggelli, Y. Lee, M. Blagojević, S. Bailey, P. Chiu, P. Dabbelt, C. Schmidt, E. Alon, K. Asanović, and B. Nikolić. A RISC-V Processor SoC With Integrated Power Management at Submicrosecond Timescales in 28 nm FD-SOI. *IEEE Journal of Solid-State Circuits (JSSC)*, 52(7):1863–1875, Jul 2017.

[KFJ+03] R. Kumar, K. Farkas, N. Jouppi, P. Ranganathan, and D. Tullsen. Single-ISA Heterogeneous Multi-Core Architectures: The Potential for Processor Power Reduction. *Int'l Symp. on Microarchitecture (MICRO)*, Dec 2003.

[KGWB08] W. Kim, M. Gupta, G.-Y. Wei, and D. Brooks. System-Level Analysis of Fast, Per-Core DVFS Using On-Chip Switching Regulators. *Int'l Symp. on High-Performance Computer Architecture (HPCA)*, Feb 2008.

[KH11] S. S. Kudva and R. Harjani. Fully-Integrated On-Chip DC-DC Converter With a 450X Output Range. *IEEE Journal of Solid-State Circuits (JSSC)*, 46(8):1940–1951, Aug 2011.

[KJT+17] J. Kim, S. Jiang, C. Torng, M. Wang, S. Srinath, B. Ilbeyi, K. Al-Hawaj, and C. Batten. Using Intra-Core Loop-Task Accelerators to Improve the Productivity and Performance of Task-Based Parallel Programs. *Int'l Symp. on Microarchitecture (MICRO)*, Oct 2017.

[KKV+18] B. Khailany, E. Krimer, R. Venkatesan, J. Clemons, J. Emer, M. Fojtik, A. K. and-Michael Pellauer, N. Pinckney, Y. S. Shao, S. Srinath, C. Torng, S. L. Xi, Y. Zhang, and B. Zimmer. A Modular Digital VLSI Flow for High-Productivity SoC Design. *Design Automation Conf. (DAC)*, Jun 2018.

[KMRM19] O. O. Kibar, P. Mohan, P. Rech, and K. Mai. Evaluating the Impact of Repetition, Redundancy, Scrubbing, and Partitioning on 28-nm FPGA Reliability Through Neutron Testing. *IEEE Transactions on Nuclear Science*, 66(1):248–254, Jan 2019.

[Kre11] K. Krewell. ARM Pairs Cortex-A7 With A15: Big.Little Combines A5-Like Efficiency With A15 Capability. *Microprocessor Report (MPR)*, Nov 2011.

[KTR+04] R. Kumar, D. M. Tullsen, P. Ranganathan, N. P. Jouppi, and K. I. Farkas. Single-ISA Heterogeneous Multi-Core Architectures for Multithreaded Workload Performance. *Int'l Symp. on Computer Architecture (ISCA)*, Jun 2004.

[KZVT17] M. Khazraee, L. Zhang, L. Vega, and M. B. Taylor. Moonwalk: NRE Optimization in ASIC Clouds. *Int'l Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, Apr 2017.

[LAS+13] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi. The McPAT Framework for Multicore and Manycore Architectures: Simultaneously Modeling Power, Area, and Timing. *ACM Trans. on Architecture and Code Optimization (TACO)*, 10(1):5:1–5:29, Apr 2013.

[LCSA13] H.-P. Le, J. Crossley, S. Sanders, and E. Alon. A Sub-ns Response Fully Integrated Battery-Connected Switched-Capacitor Voltage Regulator Delivering 0.19 W/mm^2 at 73% Efficiency. *Int'l Solid-State Circuits Conf. (ISSCC)*, Feb 2013.

[LCVR03] H. Li, C.-Y. Cher, T. Vijaykumar, and K. Roy. VSV: L2- Miss-Driven Variable Supply-Voltage Scaling For Low Power. *Int'l Symp. on Microarchitecture (MICRO)*, Jan 2003.

[LK09] J. Lee and N. S. Kim. Optimizing Throughput of Power- and Thermal-Constrained Multicore Processors using DVFS and Per-Core Power-Gating. *Design Automation Conf. (DAC)*, Jul 2009.

[LK14] D. Lo and C. Kozyrakis. Dynamic Management of TurboMode in Modern Multicore Chips. *Int'l Symp. on High-Performance Computer Architecture (HPCA)*, Feb 2014.

[LLK09] N. Lakshminarayana, J. Lee, and H. Kim. Age-Based Scheduling for Asymmetric Multiprocessors. *Int'l Conf. on High Performance Networking and Computing (Supercomputing)*, Nov 2009.

[LMH04] J. Li, J. F. Martinez, and M. C. Huang. The Thrifty Barrier: Energy-Aware Synchronization in Shared-Memory Multiprocessors. *Int'l Symp. on High-Performance Computer Architecture (HPCA)*, Feb 2004.

[LPD+14] A. Lukefahr, S. Padmanabha, R. Das, J. Ronald Dreslinski, T. F. Wenisch, and S. Mahlke. Heterogeneous Microarchitectures Trump Voltage Scaling for Low-Power Cores. *Int'l Conf. on Parallel Architectures and Compilation Techniques (PACT)*, Aug 2014.

[LSA11] H.-P. Le, S. R. Sanders, and E. Alon. Design Techniques for Fully Integrated Switched-Capacitor DC-DC Converters. *IEEE Journal of Solid-State Circuits (JSSC)*, 46(9):2120–2131, Sep 2011.

[LSL12] I.-T. A. Lee, A. Shafi, and C. E. Leiserson. Memory-Mapping Support for Reducer Hyperobjects. *Symp. on Parallel Algorithms and Architectures (SPAA)*, Jun 2012.

[LZB14] D. Lockhart, G. Zibrat, and C. Batten. PyMTL: A Unified Framework for Vertically Integrated Computer Architecture Research. *Int'l Symp. on Microarchitecture (MICRO)*, Dec 2014.

[LZW+15] Y. Lee, B. Zimmer, A. Waterman, A. Puggelli, J. Kwak, R. Jevtic, B. Keller, S. Bailey, M. Blagojevic, P.-F. Chiu, H. Cook, R. Avizienis, B. Richards, E. Alon, B. Nikolic, and K. Asanović. POWER7: IBM's Next Generation POWER Microprocessor. *Symp. on High Performance Chips (Hot Chips)*, Aug 2015.

[MBH+14] A. J. Martinez, A. A. Bajwa, D. L. Hill, E. Hallnor, H. Jiang, M. Dixon, M. Derr, M. Hunsaker, R. Kumar, R. B. Osborne, R. Rajwar, R. Singhal, R. D'Sa, R. Chappell, S. Kaushik, S. Chennupaty, S. Jourdan, S. Gunther, T. Piazza, and T. D. K. Burton. Haswell: The Fourth-Generation Intel Core Processor. *IEEE Micro*, 34(2):6–20, Mar/Apr 2014.

[MCGG06] G. Magklis, P. Chaparro, J. González, and A. González. Independent front-end and back-end dynamic voltage scaling for a GALS microarchitecture. *Int'l Symp. on Low-Power Electronics and Design (ISLPED)*, Oct 2006.

[MFN+17] M. McKeown, Y. Fu, T. Nguyen, Y. Zhou, J. Balkind, A. Lavrov, M. Shahrad, S. Payne, and D. Wentzlaff. Piton: A Manycore Processor for Multitenant Clouds. *IEEE Micro*, Mar 2017.

[Mif01] R. Miftakhutdinov. An Analytical Comparison of Alternative Control Techniques for Powering Next-Generation Microprocessors. TI Application Note, 2001.

[MPT+12] T. N. Miller, X. Pan, R. Thomas, N. Sedaghati, and R. Teodorescu. Booster: Reactive Core Acceleration For Mitigating the Effects of Process Variation and Application Imbalance in Low-Voltage Chips. *Int'l Symp. on High-Performance Computer Architecture (HPCA)*, Feb 2012.

[MVV+03a] B. Mei, S. Vernalde, D. Verkest, H. De Man, and R. Lauwereins. Exploiting loop-level parallelism on coarse-grained reconfigurable architectures using modulo scheduling. *IEEE Proceedings on Computers and Digital Techniques (IPCDT)*, Nov 2003.

[MVV+03b] B. Mei, S. Vernalde, D. Verkest, H. D. Man, and R. Lauwereins. ADRES: An Architecture with Tightly Coupled VLIW Processor and Coarse-Grained Reconfigurable Matrix. *Int'l Conf. on Field Programmable Logic (FPL)*, Sep 2003.

[MWK+06] T. Y. Morad, U. C. Weiser, A. Kolodny, M. Valero, and E. Ayguade. Performance, Power Efficiency and Scalability of Asymmetric Cluster Chip Multiprocessors. *Computer Architecture Letters (CAL)*, 5(1), Jan 2006.

[MYN+11] A. Muramatsu, T. Yasufuku, M. Nomura, M. Takamiya, H. Shinohara, and T. Sakurai. 12% Power reduction by within-functional-block fine-grained adaptive dual sup-

ply voltage control in logic circuits with 42 voltage domains. *European Solid-State Circuits Conf. (ESSCIRC)*, Sep 2011.

[OEPM09] T. Oh, B. Egger, H. Park, and S. Mahlke. Recurrence cycle aware modulo scheduling for coarse-grained reconfigurable architectures. *International Conference on Languages, Compilers, Tools, and Theory for Embedded Systems (LCTES)*, Jun 2009.

[OHL+06] S. Olivier, J. Huan, J. Liu, J. Prins, J. Dinan, P. Sadayappan, and C.-W. Tseng. UTS: An Unbalanced Tree Search Benchmark. *Int'l Workshop on Lanaguages and Compilers for Parallel Computing (LCPC)*, Nov 2006.

[Olo16] A. Olofsson. Epiphany-V: A 1024-processor 64-bit RISC System-On-Chip. *CoRR arXiv:1610.01832*, Aug 2016.

[PDS+13] N. Pinckney, R. Dreslinski, K. Sewell, D. Fick, T. Mudge, D. Sylvester, and D. Blaauw. Limits of Parallelism and Boosting in Dim Silicon. *IEEE Micro*, 33(5):30–37, Sep/Oct 2013.

[PPA+13] A. Parashar, M. Pellauer, M. Adler, B. Ahsan, N. Crago, D. Lusting, V. Pavlov, A. Zhai, M. Gambhir, A. Jaleel, R. Allmon, R. Rayess, S. Maresh, and J. Emer. Triggered Instructions: A Control Paradigm for Spatially-programmed Architectures. *Int'l Symp. on Computer Architecture (ISCA)*, Jun 2013.

[PPM09] Y. Park, H. Park, and S. Mahlke. CGRA express: accelerating execution using dynamic operation fusion. *Int'l Conf. on Compilers, Architecture, and Synthesis for Embedded Systems (CASES)*, Oct 2009.

[PSC+19] M. Pellauer, Y. S. Shao, J. Clemons, N. Crago, K. Hegde, R. Venkatesan, S. W. Keckler, C. W. Fletcher, and J. Emer. Buffets: An Efficient and Composable Storage Idiom for Explicit Decoupled Data Orchestration. *Int'l Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, Apr 2019.

[PSCP10] J. Park, D. Shin, N. Chang, and M. Pedram. Accurate Modeling and Calculation of Delay and Energy Overheads of Dynamic Voltage Scaling in Modern High-Performance Microprocessors. *Int'l Symp. on Low-Power Electronics and Design (ISLPED)*, Aug 2010.

[pyt14] PyTest. Online Webpage, 2014 (accessed Oct 1, 2014). `http://www.pytest.org`.

[PZK+17] R. Prabhakar, Y. Zhang, D. Koeplinger, M. Feldman, T. Zhao, S. Hadjis, A. Pedram, C. Kozyrakis, and K. Olukotun. Plasticine: A Reconfigurable Architecture For Parallel Paterns. *Int'l Symp. on Computer Architecture (ISCA)*, Jun 2017.

[QHS+13] W. Qadeer, R. Hameed, O. Shacham, P. Venkatesan, C. Kozyrakis, and M. Horowitz. Convolution engine: balancing efficiency and flexibility in specialized computing. *Int'l Symp. on Computer Architecture (ISCA)*, Jun 2013.

[RES+13] A. Raghavan, L. Emurian, L. Shao, M. Papaefthymiou, K. Pipe, T. Wenisch, and M. Martin. Utilizing Dark Silicon to Save Energy with Computational Sprinting. *IEEE Micro*, 33(5):20–28, Sep/Oct 2013.

[RGH+09] V. Reddi, M. Gupta, G. Holloway, G.-Y. Wei, M. Smith, and D. Brooks. Voltage Emergency Prediction: Using Signatures to Reduce Operating Margins. *Int'l Symp. on High-Performance Computer Architecture (HPCA)*, Feb 2009.

[RHH84] J. Robert H. Halstead. Implementation of Multilisp: Lisp on a Multiprocessor. *Symp. on Lisp and Functional Programming*, Oct 1984.

[RL14] H. Ribic and Y. D. Liu. Energy-Efficient Work-stealing Language Runtimes. *Int'l Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, Mar 2014.

[RWB09] K. K. Rangan, G.-Y. Wei, and D. Brooks. Thread Motion: Fine-Grained Power Management for Multi-Core Systems. *Int'l Symp. on Computer Architecture (ISCA)*, Jun 2009.

[RZAH+19] A. Rovinski, C. Zhao, K. Al-Hawaj, P. Gao, S. Xie, C. Torng, S. Davidson, A. Amarnath, L. Vega, B. Veluri, A. Rao, T. Ajayi, J. Puscar, S. Dai, R. Zhao, D. Richmond, Z. Zhang, I. Galton, C. Batten, M. B. Taylor, and R. G. Dreslinski. A 1.4 GHz 695 Giga RISC-V Inst/s 496-core Manycore Processor with Mesh On-Chip Network and an All-Digital Synthesized PLL in 16nm CMOS. *Symp. on Very Large-Scale Integration Circuits (VLSIC)*, Jun 2019.

[SAD+02] G. Semeraro, D. H. Albonesi, S. G. Dropsho, G. Magklis, S. Dwarkadas, and M. L. Scott. Dynamic frequency and voltage control for a multiple clock domain microarchitecture. *Int'l Symp. on Microarchitecture (MICRO)*, Nov 2002.

[sam19] Samsung Exynos 5 Octa (5422). Online Webpage, 2019 (accessed Sep 24, 2019). `https://www.samsung.com/semiconductor/minisite/exynos/products/mobileprocessor/exynos-5-octa-5422`.

[SBF+12] J. Shun, G. E. Blelloch, J. T. Fineman, P. B. Gibbons, A. Kyrola, and H. V. Simhadri. Brief Announcement: The Problem Based Benchmark Suite. *Symp. on Parallel Algorithms and Architectures (SPAA)*, Jun 2012.

[See09] M. Seeman. *A Design Methodology for Switched-Capacitor DC-DC Converters*. Ph.D. Thesis, EECS Department, University of California, Berkeley, May 2009.

[SGC+16] A. Sodani, R. Gramunt, J. Corbal, H.-S. Kim, K. Vinod, S. Chinthamani, S. Hutsell, R. Agarwal, and Y.-C. Liu. Knights Landing: Second-Generation Intel Xeon Phi Product. *IEEE Micro*, 36:34–46, Mar/Apr 2016.

[SGS+14] A. A. Sinkar, H. R. Ghasemi, M. J. Schulte, U. R. Karpuzcu, and N. S. Kim. Low-Cost Per-Core Voltage Domain Support for Power-Constrained High-Performance Processors. *IEEE Trans. on Very Large-Scale Integration Systems (TVLSI)*, 22(4):747–758, Apr 2014.

[SLL+00] H. Singh, M.-H. Lee, G. Lu, N. Bagherzadeh, F. J. Kurdahi, and E. M. C. Filho. MorphoSys: An Integrated Reconfigurable System for Data-Parallel and Computation-Intensive Applications. *IEEE Transactions on Computers*, May 2000.

[SMQP09] M. A. Suleman, O. Mutlu, M. K. Qureshi, and Y. N. Patt. Accelerating Critical Section Execution with Asymmetric Multi-Core Architectures. *Int'l Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, Mar 2009.

[SNL+10] M. Seeman, V. Ng, H.-P. Le, M. John, E. Alon, and S. Sanders. A Comparative Analysis of Switched-Capacitor and Inductor-Based DC-DC Conversion Technologies. *Workshop on Control and Modeling for Power Electronics*, Jun 2010.

[SPW95] L. F. G. Sarmenta, G. A. Pratt, and S. A. Ward. Rational clocking. *Int'l Conf. on Computer Design (ICCD)*, Oct 1995.

[SPW+11] N. Sturcken, M. Petracca, S. Warren, L. Carloni, A. Peterchev, and K. Shepard. An Integrated Four-Phase Buck Converter Delivering 1 A/mm^2 with 700 ps Controller Delay and Network-On-Chip Load in 45-nm SOI. *Custom Integrated Circuits Conf. (CICC)*, Sep 2011.

[Sri18] S. Srinath. *Lane-Based Hardware Specialization for Loop- and Fork-Join-Centric Parallelization and Scheduling Strategies*. Ph.D. Thesis, ECE Department, Cornell University, 2018.

[SYK10] D. Sanchez, R. M. Yoo, and C. Kozyrakis. Flexible Architectural Support for Fine-Grain Scheduling. *Int'l Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, Mar 2010.

[Tay13] M. B. Taylor. A Landscape of the New Dark Silicon Design Regime. *IEEE Micro*, 33(5):8–19, Sep/Oct 2013.

[TCB18] T. Ta, L. Cheng, and C. Batten. Simulating Multi-Core RISC-V Systems in gem5. *Workshop on Computer Architecture Research with RISC-V (CARRV)*, Jun 2018.

[TCM+09] D. Truong, W. Cheng, T. Mohsenin, Z. Yu, A. Jacobson, G. Landge, M. Meeuwsen, C. Watnik, A. Tran, Z. Xiao, E. Work, J. Webb, P. Mejia, and B. Baas. A 167-Processor Computational Platform in 65-nm CMOS. *IEEE Journal of Solid-State Circuits (JSSC)*, 44(4):1130–1144, Apr 2009.

[TGL07] P. Teehan, M. Greenstreet, and G. Lemieux. A Survey and Taxonomy of GALS Design Styles. *IEEE Design and Test of Computers*, Oct 2007.

[TJAH+18] C. Torng, S. Jiang, K. Al-Hawaj, I. Bukreyev, B. Ilbeyi, T. Ta, L. Cheng, J. Puscar, I. Galton, and C. Batten. A New Era of Silicon Prototyping in Computer Architecture Research. *RISC-V Day Workshop*, Oct 2018.

[TWB16] C. Torng, M. Wang, and C. Batten. Asymmetry-Aware Work-Stealing Runtimes. *Int'l Symp. on Computer Architecture (ISCA)*, Jun 2016.

[TWS+16] C. Torng, M. Wang, B. Sudheendra, N. Murali, S. Jayasuriya, S. Srinath, T. Pritchard, R. Ying, and C. Batten. Experiences Using A Novel Python-Based Hardware Modeling Framework For Computer Architecture Test Chips. *Poster at the Symp. on High Performance Chips (Hot Chips)*, Aug 2016.

[VBP+16] A. Vasilyev, N. Bhagdikar, A. Pedram, S. Richardson, S. Kvatinsky, and M. Horowitz. Evaluating programmable architectures for imaging and vision applications. *Int'l Symp. on Microarchitecture (MICRO)*, Oct 2016.

[VCJE+12] K. Van Craeynest, A. Jaleel, L. Eeckhout, P. Narvaez, and J. Emer. Scheduling Heterogeneous Multi-cores Through Performance Impact Estimation (PIE). *Int'l Symp. on Computer Architecture (ISCA)*, Jun 2012.

[vECGS92] T. von Eicken, D. Culler, S. Goldstein, and K. Schauser. Active Messages: A Mechanism for Integrated Communication and Computation. *Int'l Symp. on Computer Architecture (ISCA)*, May 1992.

[VHR+07] S. Vangali, J. Howard, G. Ruhi, S. Dighe, H. Wilson, J. Tschanz, D. Finan, P. Iyerl, A. Singh, T. Jacob, S. Jain, S. Venkataraman, Y. Hoskotel, and N. Borkarl. 80-Tile 1.28 TFlops Network-on-Chip in 65 nm CMOS. *Int'l Solid-State Circuits Conf. (ISSCC)*, Feb 2007.

[wav18] A Coarse Grain Reconfigurable Array (CGRA) for Statically Scheduled Data Flow Computing. WAVE Computing White Paper, 2018. `https://wavecomp.ai/wp-content/uploads/2018/12/WP_CGRA.pdf`.

[WH08] J. Wibben and R. Harjani. A High-Efficiency DC-DC Converter Using 2 nH Integrated Inductors. *IEEE Journal of Solid-State Circuits (JSSC)*, 43(4):844–854, Apr 2008.

[Whe18] B. Wheeler. RISC-V Enables IoT Edge Processor. *Microprocessor Report (MPR)*, Feb 2018.

[WJMC04] Q. Wu, P. Juang, M. Martonosi, and D. W. Clark. Formal online methods for voltage/frequency control in multiple clock domain microprocessors. *Int'l Conf. on Ar-*

chitectural Support for Programming Languages and Operating Systems (ASPLOS), Oct 2004.

[WJMC05] Q. Wu, P. Juang, M. Martonosi, and D. W. Clark. Voltage and Frequency Control With Adaptive Reaction Time in Multiple-Clock-Domain Processors. *Int'l Symp. on High-Performance Computer Architecture (HPCA)*, Feb 2005.

[WL08] D. H. Woo and H.-H. S. Lee. Extending Amdahl's Law for Energy-Efficient Computing in the Many-Core Era. *IEEE Computer*, 41(12):24–31, Dec 2008.

[WM15] X. Wang and J. F. Martinez. XChange: A Market-Based Approach to Scalable Dynamic Multi-Resource Allocation in Multicore Architectures. *Int'l Symp. on High-Performance Computer Architecture (HPCA)*, Feb 2015.

[WS11] M. Wens and M. Steyaert. Fully-Integrated CMOS 800-mW Four-Phase Semi-Constant On/Off-time Step-Down Converter. *IEEE Trans. on Power Electronics (TPEL)*, 26(2):326–333, Feb 2011.

[WWFY11] R. Wang, H. Wang, B. Fan, and L. Yang. RIRI scheme: A robust instant-responding ratiochronous interface with zero-latency penalty. *Int'l Conf. on Circuits and Systems (ISCAS)*, May 2011.

[YCZ12] M. K. Yadav, M. R. Casu, and M. Zamboni. DVFS Based on Voltage Dithering and Clock Scheduling for GALS Systems. *Int'l Symposium on Asynchronous Circuits and Systems (ASYNC)*, May 2012.

[YLH+12] G. Yan, Y. Li, Y. Han, X. Li, M. Guo, and X. Liang. AgileRegulator: A Hybrid Voltage Regulator Scheme Redeeming Dark Silicon for Power Efficiency in a Multicore Architecture. *Int'l Symp. on High-Performance Computer Architecture (HPCA)*, Feb 2012.

[ZJKS11] P. Zhou, D. Jiao, C. Kim, and S. Sapatnekar. Exploration of On-Chip Switched-Capacitor DC-DC Converter for Multicore Processors Using a Distributed Power Delivery Network. *Custom Integrated Circuits Conf. (CICC)*, Sep 2011.

[ZYFL10] Z. Zeng, X. Ye, Z. Feng, and P. Li. Tradeoff Analysis and Optimization of Power Delivery Networks with On-Chip Voltage Regulation. *Design Automation Conf. (DAC)*, Jun 2010.