# UC San Diego

## UC San Diego Electronic Theses and Dissertations

**Title**

Adaptive architectures for peak power management

**Permalink**

https://escholarship.org/uc/item/5gn6b57g

**Author**

Kontorinis, Vasileios

**Publication Date**

2013

Peer reviewed|Thesis/dissertation

eScholarship.org Powered by the California Digital Library
University of California

UNIVERSITY OF CALIFORNIA, SAN DIEGO

**Adaptive Architectures for Peak Power Management**

A dissertation submitted in partial satisfaction of the
requirements for the degree
Doctor of Philosophy

in

Computer Science (Computer Engineering)

by

Vasileios Kontorinis

Committee in charge:

Professor Dean Tullsen, Chair
Professor Tajana Rosing, Co-Chair
Professor Andrew B. Kahng
Professor Timothy Sherwood
Professor Steven Swanson

2013

Copyright
Vasileios Kontorinis, 2013
All rights reserved.

The dissertation of Vasileios Kontorinis is approved, and it is acceptable in quality and form for publication on microfilm and electronically:

_____

_____

_____

_____
Co-Chair

_____
Chair

University of California, San Diego

2013

DEDICATION

To my dear wife Banu – my life's companion and critic.

# EPIGRAPH

*"Logic will get you from A to Z; imagination will get you everywhere."*
– Albert Einstein

*"Don't cry because it's over, smile because it happened."*
– Dr. Seuss

TABLE OF CONTENTS

Signature Page . . . iii

Dedication . . . iv

Epigraph . . . v

Table of Contents . . . vi

List of Figures . . . viii

List of Tables . . . ix

Acknowledgements . . . x

Vita and Publications . . . xii

Abstract of the Dissertation . . . xiii

Chapter 1 Introduction . . . 1
1.1 Why enforce power limits . . . 2
1.2 Managing peak power with adaptation . . . 4
1.2.1 Power capping at the core level . . . 4
1.2.2 Removing over-provisioning in 3D stacked chips . . . 5
1.2.3 Power capping in the data center . . . 5

Chapter 2 Core peak power management . . . 7
2.1 Related work . . . 9
2.2 The power of adaptation . . . 10
2.3 Adaptive architecture to limit peak power . . . 15
2.3.1 Filling the config ROM . . . 17
2.3.2 Configuration mechanisms . . . 19
2.3.3 Implementation overhead . . . 22
2.4 Methodology . . . 24
2.4.1 Power modeling . . . 24
2.4.2 Benchmarks . . . 26
2.5 Results . . . 26
2.5.1 Dynamic adaptation vs static tuning . . . 27
2.5.2 Realistic adaptive techniques . . . 32
2.5.3 Reducing ROM configurations . . . 37
2.5.4 Delay sensitivity analysis . . . 38
2.5.5 Quantifying the benefits of peak power reduction 39
2.6 Conclusion . . . 40

Chapter 3 Removing over-provisioning in 3D stacked chips . . . . . 42
3.1 Baseline architecture . . . . . 45
3.1.1 Processor model . . . . . 45
3.1.2 3D floorplans . . . . . 46
3.2 Resource pooling in the third dimension . . . . . 47
3.3 Stackable structures for resource pooling . . . . . 50
3.3.1 Reorder buffer and register file . . . . . 50
3.3.2 Instruction queue and ld/st queue . . . . . 52
3.4 Adaptive mechanism for resource pooling . . . . . 55
3.5 Methodology . . . . . 56
3.5.1 Modeling interconnect for resource pooling . . . . 57
3.5.2 Benchmarks and metrics . . . . . 60
3.6 Results . . . . . 61
3.6.1 Fine vs. coarse partitioning . . . . . 64
3.6.2 Power, temperature, and energy . . . . . 65
3.7 Related work . . . . . 66
3.8 Conclusion . . . . . 68

Chapter 4 Managing peak power for data centers . . . . . 70
4.1 Background . . . . . 73
4.2 Total cost of ownership analysis . . . . . 74
4.3 The benefits of power oversubscription . . . . . 78
4.4 Characterizing distributed UPS batteries . . . . . 80
4.5 Policies . . . . . 88
4.6 Methodology . . . . . 92
4.7 Results . . . . . 95
4.8 Related work . . . . . 102
4.9 Conclusions . . . . . 104

Chapter 5 Summary . . . . . 106
5.1 Core peak power management . . . . . 106
5.2 Resource pooling for power efficiency . . . . . 108
5.3 Managing peak power for data centers . . . . . 109
5.4 Concluding remarks . . . . . 111

Bibliography . . . . . 112

LIST OF FIGURES


Figure 1.1: Power density throughout the history of electronics . . . . . 2
Figure 1.2: Organization of thesis body in chapters. . . . . 6

Figure 2.1: Performance when core bottlenecks gradually removed . . . . . 15
Figure 2.2: The adaptive core architecture . . . . . 16
Figure 2.3: Peak and average power breakdown . . . . . 26
Figure 2.4: Performance when core peak power constraint at 70% . . . . . 28
Figure 2.5: Performance when core peak power constraint at 75% . . . . . 29
Figure 2.6: Performance when core peak power constraint at 80% . . . . . 31
Figure 2.7: Average over peak power ratio . . . . . 34
Figure 2.8: Performance of adaptive policies . . . . . 36
Figure 2.9: Performance for different peak power constraints . . . . . 37
Figure 2.10: Sensitivity analysis to adaptation delay . . . . . 39
Figure 2.11: Distributed power distribution network model. . . . . 40

Figure 3.1: Baseline and resource pooled CMP configuration . . . . . 46
Figure 3.2: Motivation graph for back-end resource pooling . . . . . 48
Figure 3.3: Partitioned reorder buffer and register file design . . . . . 51
Figure 3.4: Partitioned instruction queue design . . . . . 53
Figure 3.5: Weighted speedup and fairness with varying number of threads 62
Figure 3.6: 3D sharing performance for medium-end and high-end cores . . 63
Figure 3.7: Weighed speedup for different partition sizes . . . . . 64
Figure 3.8: Power consumption per core with and without sharing . . . . . 65
Figure 3.9: Max core temperature with and without sharing . . . . . 66
Figure 3.10: Energy efficiency in $MIPS^2$ per Watt . . . . . 67

Figure 4.1: UPS topologies in data centers . . . . . 73
Figure 4.2: Data center Total Cost of Ownership breakdown . . . . . 77
Figure 4.3: Oversubscription benefits at different levels of power hierarchy . 79
Figure 4.4: Sensitivity of TCO savings to server cost and server peak power 81
Figure 4.5: Comparison of battery technologies . . . . . 82
Figure 4.6: Battery capacity and TCO savings for varying load values . . . 86
Figure 4.7: TCO savings with future battery cost . . . . . 88
Figure 4.8: The relation between targeted depth of discharge and the reduction in TCO. . . . . 88
Figure 4.9: Algorithms for localized and coordinated policies . . . . . 90
Figure 4.10: Data center yearly variation in energy required . . . . . 94
Figure 4.11: Server, grid and battery power over three days of operation . . 96
Figure 4.12: Load partitioning into separate clusters . . . . . 97
Figure 4.13: Sensitivity of power capping to battery failures . . . . . 101

LIST OF TABLES


Table 2.1: Important core parameters for Media, Olden, NAS applications . 13
Table 2.2: Important core parameters for spec2000 applications . . . . . 14
Table 2.3: Design Space . . . . . 18
Table 2.4: Distribution of configurations over peak power groups . . . . . 19
Table 2.5: Assumed delays to powergate components . . . . . 21
Table 2.6: Delays to disable components holding state . . . . . 22
Table 2.7: Absolute Power Values for the modeled processor . . . . . 25
Table 2.8: Architectural Specification . . . . . 25
Table 2.9: Example of configuration score estimation . . . . . 35
Table 2.10: Distribution of reduced set configurations over peak power groups 38
Table 2.11: Peak power impact on voltage variation and on-chip decoupling capacitor area . . . . . 38

Table 3.1: Architectural specification. . . . . 58
Table 3.2: Tier to tier delay via TSV path. . . . . 58
Table 3.3: TSV area utilization. . . . . 59
Table 3.4: Temperature estimation related parameters. . . . . 60
Table 3.5: Workload Mix . . . . . 61

Table 4.1: TCO model assumptions . . . . . 78
Table 4.2: Input values for battery cost estimation. . . . . 83
Table 4.3: Recharge cycles as a function of depth of discharge (DoD) . . . . 83
Table 4.4: Workloads . . . . . 93
Table 4.5: Relative workload weights . . . . . 93

ACKNOWLEDGEMENTS

I want to thank my advisor, Dean Tullsen, for his guidance and support. He has been an inspirational example of personal and professional integrity. Always setting the bar high, he urged me to improve and dream big, radical ideas. He demonstrated confidence in my abilities to solve tough problems and supported me every time I would hit dead-end in my research. Dean, this work would never be possible without you.

I thank my co-advisor, Tajana Rosing, and the other members of my thesis committee – Steven Swanson, Andrew B. Kahng, Tim Sherwood – for taking the time to review my work, and oversee the completion of this dissertation. Special thanks to professor Rosing for broadening my area of expertise by allowing me to work beyond microarchitecture, at the systems level, for teaching me the importance of exciting people with my work and for encouraging me to form collaborations and mentor junior students.

I want to express my appreciation to my family. My parents, Litsa and Palaiologos for motivating me to reach the highest level of studies, instilling high morals, patience and perseverance in my personality and teaching me how to treat everyday life with humor. My oldest brother, Nick, for being my role model, the guy I have always looked up to. My middle brother, George, for reminding me that life is about moments and I should never forget to live it. I love you guys.

Life at UCSD would be dull without all the great people I met during this phase of my life. I first want to thank, my seniors, Jeff Brown, Leo Porter, Matt Devuyst for our discussions and all the knowledge and experience I obtained through them. My juniors Michael Wei and Alex Eisner, for making the lab a fun and creative place to be by transforming it into the Amazon and following a Nerf-gun war zone. My collaborators, Houman Homayoun, Jack Sampson, Amirali Shayan and Nikos Strikos. A big thanks to you guys, your input was invaluable.

At last, I want to thank the most important person in my life, my wife Banu, for being my companion and my critic. The person who shared with me every good and bad moment, whose unconditional love was a constant source of inspiration and energy. Banu, thank you for being always there for me.

Chapter 2 contains material from "Reducing peak power with a table-driven adaptive processor core", by Vasileios Kontorinis, Amirali Shayan, Rakesh Kumar, and Dean Tullsen, which appears in *Proceedings of the 42nd annual International Symposium on Microarchitecture (MICRO)*. The dissertation author was the primary investigator and author of this paper. The material in this chapter is copyright ©2009 IEEE. Personal use of this material is permitted. However, permission to reprint/republish this material for advertising or promotional purposes or for creating new collective works for resale or redistribution to servers or lists, or to reuse any copyrighted component of this work in other works must be obtained from the IEEE.

Chapter 3 contains material from "Dynamically heterogeneous cores through 3D resource pooling", by Houman Homayoun, Vasileios Kontorinis, Amirali Shayan, Ta-Wei Lin, and Dean Tullsen, which appears in *Proceedings of the The 18th International Symposium on High Performance Computer Architecture (HPCA)*. The dissertation author contributed equally with the main author of the work, Houman Homayoun. The material in this chapter is copyright ©2012 IEEE. Personal use of this material is permitted. However, permission to reprint/republish this material for advertising or promotional purposes or for creating new collective works for resale or redistribution to servers or lists, or to reuse any copyrighted component of this work in other works must be obtained from the IEEE.

Chapter 4 contains material from "Managing Distributed UPS Energy for Effective Power Capping in Data Centers", by Vasileios Kontorinis, Liuyi Zhang, Baris Aksanli, Jack Sampson, Houman Homayoun, Eddie Pettis, Dean Tullsen and Tajana Rosing, which appears in *Proceedings of the 39th International Symposium on Computer Architecture (ISCA)*. The dissertation author was the primary investigator and author of this paper. The material in this chapter is copyright ©2012 IEEE. Personal use of this material is permitted. However, permission to reprint/republish this material for advertising or promotional purposes or for creating new collective works for resale or redistribution to servers or lists, or to reuse any copyrighted component of this work in other works must be obtained from the IEEE.

## VITA AND PUBLICATIONS

| | |
|---|---|
| 2006 | B. S. in Electrical Engineering and Computer Science<br>University of Patras, Greece. |
| 2007 | Internship<br>Conexant Inc.<br>San Diego |
| 2008 | Internship<br>Syn Microsystems Inc.<br>Menlo Park |
| 2008 | M. S. in Computer Science & Engineering.<br>University of California, San Diego |
| 2013 | Ph. D. in Computer Science (Computer Engineering)<br>University of California, San Diego |

## PUBLICATIONS

Vasileios Kontorinis, Amirali Shayan, Rakesh Kumar, Dean Tullsen, "Reducing peak power with a table-driven adaptive processor core", *International Symposium on Microarchitecture (MICRO), December, 2009.*

Gaurav Dhiman, Vasileios Kontorinis, Dean Tullsen, Tajana Rosing, Eric Saxe, Jonathan Chew, "Dynamic workload characterization for power efficient scheduling on CMP systems", *International Symposium in Low-Power Electronics and Design (ISLPED), Aug 2010.*

Houman Homayoun, Vasileios Kontorinis, Amirali Shayan, Ta-Wei Lin, Dean Tullsen, "Dynamically heterogeneous cores through 3D resource pooling", *International Symposium on High Performance Computer Architecture (HPCA), February 2012.*

Houman Homayoun, Mehryar Rahmatian, Vasileios Kontorinis, Shahin Golshan, Dean Tullsen, "Hot Peripheral Thermal Management to Mitigate Cache Temperature Variation", *Proceeding of International Symposium on Quality of Electronic Design, (ISQED), Mar 2012.*

Vasileios Kontorinis, Liuyi Zhang, Baris Aksanli, Jack Sampson, Houman Homayoun, Eddie Pettis, Dean Tullsen, Tajana Rosing, "Managing Distributed UPS Energy for Effective Power Capping in Data Centers", *39th Annual International Symposium on Computer Architecture (ISCA), June 2012.*

ABSTRACT OF THE DISSERTATION

**Adaptive Architectures for Peak Power Management**

by

Vasileios Kontorinis

Doctor of Philosophy in Computer Science (Computer Engineering)

University of California, San Diego, 2013

Professor Dean Tullsen, Chair
Professor Tajana Rosing, Co-Chair

Power budgeting – allocating power among computing components to maximize performance – is a necessity for both modern CPUs and systems in general. Our industry expects every new chip generation to double performance. However, non-ideal transistor scaling makes it hard to provide the expected performance at the same power envelope. The available power budget dictates microprocessor performance and functionality. At the system level, power provisioning affects infrastructure and energy costs. Power associated costs like wall-power cost, cooling, and power delivery have already surpassed the server costs in state-of-the-art data centers. Every component of the computing stack is associated with a power budget, and its respective marginal increase in performance must justify the respective power increase.

Power budgeting is a difficult issue to address. Different workloads stress different computing resources and even a single application may stress different resources at different points of time. Thus, a single universal solution is hard to achieve. Currently, most systems fix the decision of what resources consume more

power at design time. Instead, in this thesis we propose to dynamically customize resources to boost performance while respecting power budgets.

This thesis presents techniques that manage power in three domains: the processor core, 3D-die stacked chips, and the data center. First, we describe a microarchitectural solution that always keeps a portion of the core powered-off, but what gets powered off can change for each application. By intelligently providing the most critical resources, we achieve a particular peak power goal while minimizing performance impact. Second, we dynamically reconfigure the resources of 3D-stacked cores. Adaptation in this context allocates unused back-end resources of one core to an adjacent core in the z-axis that needs them. Hence, we can use lower-power cores and boost their performance. Finally, we present a solution to reduce data center costs by using the energy stored in UPSs (Uninterruptible Power Supply). By discharging UPS batteries during high utilization and recharging them during low utilization, we adapt available power capacity. This approach permits more functionality under the provisioned power and can translate to cost savings of millions of dollars.

# Chapter 1

# Introduction

In the past 5 years, we have witnessed an abrupt slowdown of the rate at which we reduce chip nominal voltage and as a result we can no longer double chip performance at the same power envelope. With increasing power dissipation and fixed power budgets, allocating power to maximize performance becomes critical.

Classical CMOS scaling, as described by Dennard in [DGR+74], dictates that nominal voltage and threshold voltage should be reduced across technology nodes so that power density remains the same. However, lower threshold voltage results in exponential increase of leakage power. Because of this increase we can no longer reduce voltage at the same pace; otherwise, leakage will dominate power dissipation. As the degree of integration in the computing stack increases and available power budgets do not change significantly, the power budget per component becomes tighter. Even in the high performance domain, we can no longer afford to think of performance in isolation. We need to allocate the available power budget in such a way that maximizes performance returns.

The first step towards achieving this goal is to remove system overprovisioning. Systems are commonly provisioned for the worst case. We want to either disable unused system capacity or re-purpose it to actively contribute to performance. This thesis investigates how to manage peak power by removing overprovisioning and identifies the associated benefits across three different domains: the microprocessor core, the chip and the data center.

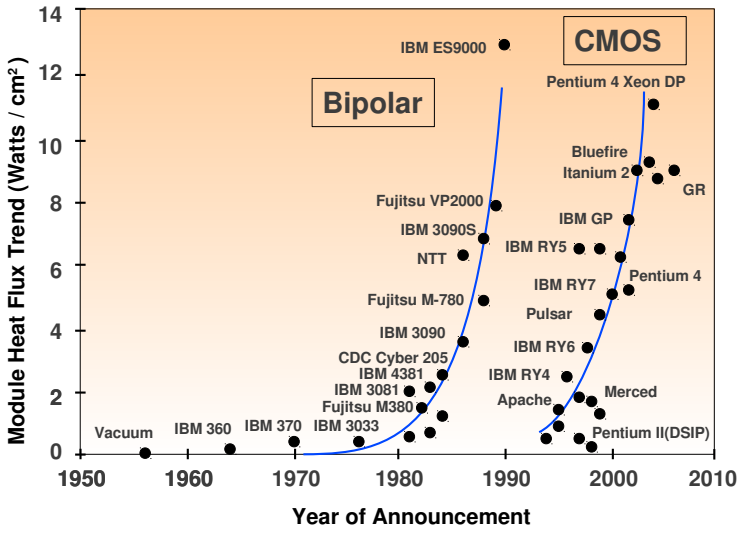



Figure 1.1: Technological shifts such as the transition from NMOS to CMOS and the introduction of multicore allowed significant improvements in power density. However, today most of the magic bullets have been spent and the slowdown of nominal voltage reduction due to leakage results in ever tighter power budgets. Reproduced from [Shm05]

## 1.1 Why enforce power limits

Before we discuss power budgeting – how to manage power of a given budget to maximize performance, we should discuss why to constrain the power budget in the first place. Power capping – limiting peak power – is necessary because it reduces the cost of the system, in some cases even makes the system financially worthwhile.

How limiting power affects cost is a function of the type of power we limit. "Average" power is the aggregate energy spent over a time period. "Peak sustained" power is the max power that the system can effectively cool over long time periods. "Theoretical peak" is the highest possible power, estimated as the sum of the power of all system components once each component is stressed with maximum load. Note than the theoretical peak power of a system may not be always achievable, because it is not always possible to stress all system components with

maximum load at the same time. Reducing the "average" power assuming we do not hurt performance, translates to reduced wall power costs, that is lower electricity bills. It also improves component reliability because indirectly reduces temperature and hence component failure rate. Limiting the "peak sustained" power translates to cheaper cooling costs and limiting the "theoretical peak" power lowers power delivery costs, reduces package cost, and removes constraints on processor design. In this thesis, we are more concerned with reducing the peak power of the chip, sustained and theoretical.

Peak sustained power defines the cost of the cooling mechanism. Chip packaging, the heat sink and fan is designed with the sustained power in mind, often called Thermal Design Power (TDP). Exotic cooling solutions, other than air-cooling, increase manufacturing costs and are not financially feasible. Thus, chip manufacturers either select under-performing components that require less power at design time to ensure air cooling can effectively remove heat, and/or introduce mechanisms to dynamically cap power at runtime. If CPU power exceeds the TDP for a given time period – milliseconds to tens of milliseconds – the microprocessor throttles performance to avoid permanent failure because of overheating [RNA+12]. Similarly, in modern data centers, ambient temperature conditions are strictly controlled and power capping mechanisms are enforced to ensure the data center remains within the provisioned cooling capabilities.

We design power delivery based on the theoretical peak power of the system because at design time it is hard to anticipate system configuration and load. The delivery network must be able to provide the worst case current. Exceeding the max current for power delivery will result in immediate operation halt. We design every component of the computing stack for a specific current rating. Higher current rating components such as voltage regulators, package pins, power supplies, and wires are bulkier and cost more. By limiting the theoretical peak power we can save on the cost on those components. Furthermore, significant difference between the theoretical peak power and the average power of the chip indicates large potential current swings which can lead to voltage drops and erroneous behavior ($di/dt$ issues). To deal with $di/dt$ issues processor manufacturers introduce

large timing guard-bands which leads to lower operating frequencies, and also incorporate an hierarchy of decoupling capacitors. The former negatively affects market value while the latter directly increases costs. In data centers, theoretical power directly translates to costly power distribution units, diesel generators, uninterruptible power supplies and other supporting machinery, necessary to ensure uninterrupted operation of computing infrastructure.

In order to decrease system cost, we want to design for a power value that is much lower than the theoretical peak, and in fact closer to the average power. However, we desire mechanisms that enforce power budgets without negatively affecting performance.

## 1.2 Managing peak power with adaptation

Power capping is important today and will become even more important as the degree of integration increases while available power budgets remain the same. Therefore, power budgeting is of paramount importance. Towards this goal we employ adaptation. Adaptation is the dynamic tailoring of resources so that we can decrease over-provisioning. We achieve that by powering off or reallocating to different domains unused and underutilized resources. We can also eliminate over-provisioning by adapting the capability to provide power over time. By charging and discharging batteries we can flatten the system power profile and provision power infrastructure for a point that is much lower than the theoretical peak.

### 1.2.1 Power capping at the core level

In chapter 2 we use adaptation to reduce over-provisioning by disabling core components. By guaranteeing that some portion of resources are always disabled, we can significantly reduce the peak power of the core. We find that in modern out-of-order cores all resources are not equally important for all applications. In fact, by intelligently providing the right components we can keep significant portion of the core powered off and give up little performance. The components we consider in this context are first level caches, integer and floating point units, register files,

instruction queues and the reorder buffer. We devise a low-overhead methodology that coordinates the adaptation of core components and ensure by construction that we do not violate arbitrarily-set power budgets. We describe algorithms to reduce the power-constraint design space and efficiently navigate it at run-time. We show that core level power capping can decrease voltage variation and improve the efficiency of on-chip power delivery.

### 1.2.2 Removing over-provisioning in 3D stacked chips

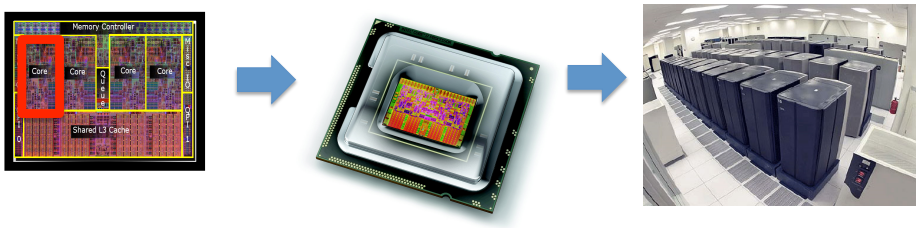
In chapter 2 we remove over-provisioning by dynamically tailoring resources to provide good performance on a tighter budget, while in chapter 3 we design the system with fewer resources to begin with and dynamically pool them to improve performance.

In modern multicores significant over-provisioning exists due to resource fragmentation. Resources needed by an application executing in one core maybe idle on a neighboring core. The resources may be idle because no application is scheduled in the neighboring core or because the application scheduled on the specific core simply does not use them. In chapter 3 we identify this type of fragmentation for the core back-end resources and use adaptation to remedy it. We rely on the proximity of vertically stacked cores in the z-axis to pool resources. 3D integration dramatically reduces wire delays and permits the access of resources on neighboring cores within a single cycle. The proposed architecture introduces a new alternative in 3D micro-architecture, which addresses many of the shortcomings of fusing resources in the 2D plane, and reduces over-provisioning.

### 1.2.3 Power capping in the data center

In chapter 4 we focus on the data center and present a technique to lower the overall provisioned power and the associated costs without affecting overall server performance. We use the energy stored in data center Uninterruptible Power Supplies (UPS) batteries to provide higher levels of power during high utilization hours throughout the day and recharge the batteries at night when server utilization is

low. With this adaptive technique we reshape the power profile of the data center and we perform the same work with significantly less power over-provisioning. We argue that investing in UPS batteries enables more aggressive power capping. As a result, we can over-subscribe power related infrastructure by adding more servers, and decrease the overall cost per server.




**chapter 2:** Core level **chapter 3:** Chip level **chapter 4:** Data center level

Figure 1.2: Organization of thesis body in chapters.

Figure 1.2 illustrates the organization of the body of this thesis. In chapter 2 we discuss power capping for a single core. Chapter 3 presents a technique to reduce over-provisioning one level higher, at the chip level. Finally, chapter 4 elaborates on power capping several levels higher, at the data center level.

# Chapter 2

# Core peak power management

The power dissipation of processors has become a first-class concern for both mobile devices as well as high-performance processors. This issue only becomes more challenging as we continue to pack more cores and other devices onto the processor die. Recent research has presented a number of techniques that enable a reduction in the average power of a processor or processor core and reduce the adverse effects of high power on cost and reliability [IBC+06, YDT+05, ZHC02, BKAB03]. Much less attention has been devoted to peak power; however, the peak power dissipation of a processor is also a first-class concern because it drives the design of the processor, thermal budgeting for the processor and system, the packaging and cooling costs, and the power supply costs and efficiency.

Designing robust power delivery for high performance processors requires expensive power supply and packaging solutions, including additional layers and decoupling capacitors embedded in the packaging [MES+07]. Placing on-chip decoupling capacitors on heavily congested silicon areas to prevent power overshoots is not always possible [Uni04]. Due to the random and unpredictable nature of the operational power demand, power distribution networks are over-designed for a worst case scenario which rarely occurs in normal operation. Peak power reduction will reduce on chip voltage variation along with the associated timing uncertainty and potential signal failures, or it will allow designers to achieve the same voltage variation with less silicon real estate for decoupling capacitance.

Also, power supply efficiency is proportional to the peak load current de-

mand of the processor. A power supply that is designed for the peak power requirement will be less efficient when supplying loads that are substantially under the peak power [ZWX+00]. That is, reducing peak power will result in both cheaper power supply solutions, but also power supplies that consume less power (even if the delivered average power has not changed) due to the increase in power supply efficiency.

This chapter presents a peak power management technique for current and future processors that attempts to guarantee that the peak power consumption of a processor is far lower than the sum of all core blocks. We do this via a highly adaptive processor and a table-driven approach to configuring the processor. In prior approaches [Alb99, BAS+01, MBB01, DBB+02], adaptive elements such as caches, queues, functional units, etc. make local decisions to decrease size based on local activity. This reduces average power, but does nothing for peak power. The same worst-case scenario which could theoretically incur a power drain close to peak power would also cause all of those elements to be maximally configured at once.

In our table-driven approach, we configure resources centrally. This central control allows us to guarantee that we do not give to all resources their maximum value at the same time. We can choose an arbitrary peak power design point, and only allow those configurations that do not exceed that point. Performance loss is minimal because we still allow any single resource to get its maximum value, just not all at once. We show that typical applications get close to full performance as long as their primary bottleneck resources are fully configured. We find that it is important to be able to adapt dynamically, because not all applications have the same bottleneck resources.

Thus, the discussed table-driven adaptive architecture requires configurable components, a table of possible configurations, and a mechanism for selecting the configuration to run in the next epoch. This research explores all of these design issues. We find that we are able to reduce peak power by 25% with only a 5% loss in performance.

This chapter is organized as follows. Section 2.1 discusses related work.

Motivation and mechanisms for reducing peak power dissipation are discussed in section 2.2. In section 2.3, the adaptive architecture for reducing peak power is presented. The methodology is provided in section 2.4. Results are shown in section 2.5 and section 2.6 concludes.

## 2.1 Related work

Albonesi, et al.[ABD+03] present a comprehensive study on how to tune processor resources in order to achieve better energy efficiency with several hardware and software techniques. We tune similar resources and in some cases even use the same hardware mechanisms for hardware adaptation. However, the policies for triggering and evaluating the potential configurations are completely different when targeting average power.

Most related work in adaptive micro-architectures considers a small number of configurable parameters. However, Lee et al. [LB08] explore the trends and limits of adaptivity in micro-processor design. They conduct a thorough study of a 240 billion point design space employing statistical inference techniques and genetic algorithms and provide intuition regarding the benefits of dynamically adapting processor resources in terms of performance, power, and efficiency. However, they do not provide run time heuristics that could be employed in order to dynamically adapt the processor and they target average power.

Dynamic Thermal Management [BM01, Int00] is a reactive technique that does not control peak power but does ensure that the power dissipation does not cause the temperature to exceed a threshold. It does this by reacting to thermal sensors and decreasing processor activity when temperature thresholds are exceeded. These techniques can reduce packaging costs related to heat dissipation and reduce temperature-related failures.

The research of Isci, et al.[IBC+06] and Meng, et al.[MJDS08] address the problem of meeting a global power budget while minimizing performance loss. The former employs dynamic voltage and frequency scaling (DVFS) and uses a trial-and-error method for system performance optimization. The latter uses DVFS and

cache adaptation in combination with analytic models for performance and power prediction. However, both works treat peak power budget as a soft limitation that can temporarily be exceeded and rely on power sensor measurements for reactively tuning the power consumption. Our architecture is designed to provide peak power guarantees and in that context does not rely on power sensor measurements.

Sartori and Kumar [SK09a, SK09b] target peak power on multicores. They employ DVFS to limit the peak power consumption of each core in a pro-active way that prevents global power overshoots. Our work is orthogonal and complementary to Sartori's work. Assuming different levels of peak power per core, decentralized peak power management can be used on top of our core design. This chapter focuses on reducing the peak power of each core, providing a building block for global multi-core power budgeting solutions.

In this chapter, we exploit a number of previously proposed, local adaptive techniques originally designed to reduce average power in particular components. These references are given in section 2.3.2. The key contribution of this work is the table-driven mechanism that coordinates the local adaptive techniques to limit power. In contrast to prior research focusing on energy reduction, thermals, or reliability, this work focuses on power delivery. The state-of-the-art practice to address signal integrity issues in modern chips is to introduce timing guard-bands and add an hierarchy of decoupling capacitors. In section 2.5, we show that our approach can reduce voltage variation for a given on-chip decap or require less on-chip decap to achieve a specific voltage variation goal.

## 2.2 The power of adaptation

Bounding peak power has several key advantages. It can reduce packaging and cooling costs. It will reduce the cost and increase the efficiency of the power supply. It may eliminate the need for thermal sensors in cost-sensitive designs. It allows more aggressive design in other parts of the processor or system. It reduces the need for large decoupling capacitors on chip. Power delivery circuitry does not have to be over-designed.

Reactive dynamic thermal management techniques only provide some of these advantages. It should also be noted that these techniques are not mutually exclusive, but actually complementary. Our adaptive processor with peak power guarantees can be a highly effective part of the response to thermal emergencies, because it allows us to change the peak power reactively. Also, while the reactive mechanisms do nothing for average power in the absence of thermal events, our architecture also reduces average power, because it always has part of the core turned off.

A good general-purpose processor will typically be configured such that any program that runs on the processor will find it's most critical resources sufficiently supplied to enable high throughput — this is what makes it a good general-purpose processor. However, it is rare that a *single* application needs all of those resources. We find that most applications have only a few bottlenecks, and as long as those resources are sufficiently supplied, the program can get nearly full performance.

Figure 2.1 motivates this research. It shows the result of experiments that provide a fully configured processor (similar to the MAX_CONF design point described later in this chapter), a minimally configured processor where most resources are dramatically reduced (cut in half), and a series of intermediate processors where a subset of the resources are at full capacity. The resources that change include instruction cache, data cache, integer and floating point instruction queues, reorder-buffer, load-store execution units, integer and floating point execution units, and renaming registers.

The stacked bar chart in the specific graph depicts (1) the average speedup over the minimally configured core when one resource is maximized for each benchmark (2) the average speedup when two resources are maximized, (3) the average speedup when three resources are maximized, and (4) the fully configured result. When results are shown for a subset of resources at maximum, we select those resources that give the highest performance for that particular benchmark. Two observations jump out of this graph. The first is that the total difference between the fully configured performance and the minimally configured (the baseline for the graph) is quite significant, since the former is approximately 50% faster than

the latter.

Even more significantly, we see that we can cover 65% of the gap between the two by only providing two out of ten resources at full capacity, as long as we choose those two resources carefully. This number rises to more than 85% when we provide three resources at full capacity. This is a powerful result — we can heavily under-configure 70% of the processor's components (very roughly speaking) and give up little performance. The bottleneck resources vary between applications, so we cannot achieve this same result with a single static configuration. This is confirmed in Tables 2.1 and 2.2, where the most important resources per application are presented. Lee and Brooks [LB08] note similar variance in bottlenecks, although the actual resources are different. These results indicate that we should be able to achieve close to full performance for any single application with a configuration whose cost (in peak power, in particular) is closer to the minimally configured core than it is to the maximally configured core.

There are two issues that might limit our ability to provide power guarantees, static leakage and process variation. Static leakage varies significantly with temperature. So, coupled with thermal sensors and a mechanism for dealing with thermal events (again, possibly using this architecture to deal with those events), our design can still provide a cap for both static and dynamic power. Process variation [TRfS03] is expected to result in cores that vary from the expected power behavior, even across a single chip. Again, this is an opportunity for this architecture, rather than a problem. Assuming we can detect this variation in testing/verification, we can either provide the same list of configurations to each core and thus provide a different guaranteed peak power for each core, or provide a different list of configurations to each core and thus provide the same peak power for each core. In each case, we still provide reduced peak power at the level of the processor.

Table 2.1: Important parameters for selected Media, Olden and Nas benchmarks. The resource that gives the best performance when maximized is marked with 1, the resource that gives the best performance when maximized in combination with the first one is marked with 2 and the resource that gives the best results with the first two is marked with 3.

| **Media** | iq | fq | ialu | falu | ldst | ic | dc | ipr | fpr | rob |
|---|---|---|---|---|---|---|---|---|---|---|
| g721d | | | 2 | | | 1 | 3 | | | |
| mesa-texgen | | | 2 | | | 1 | 3 | | | |
| epic | 3 | | 1 | | | | 2 | | | |
| jpege | | | 1 | | | 3 | 2 | | | |
| **Olden** | iq | fq | ialu | falu | ldst | ic | dc | ipr | fpr | rob |
| perim_big | 3 | | 1 | | | | 2 | | | |
| mst | | | 3 | | | 1 | | 2 | | |
| treeadd | 1 | | | | | | | 2 | | 3 |
| health | 3 | | | | | 1 | | 2 | | |
| bisort_med | 3 | | 2 | | | | 1 | | | |
| em3d_med | | | | | | 1 | | 2 | | 3 |
| **Nas** | iq | fq | ialu | falu | ldst | ic | dc | ipr | fpr | rob |
| mg.big | | 3 | | | | | | | 1 | 2 |
| ft.big | | 2 | | 3 | | | 1 | | | |
| sp.big | | 2 | | | | | | | 1 | 3 |
| cg.big | 3 | 1 | | | | | 2 | | | |
| bt.big | | 1 | | | | | | | 2 | 3 |
| ep.big | | 1 | | 3 | | 2 | | | | |

| | |
|---|---|
| *iq:* integer instruction queue | *fq:* floating point instruction queue |
| *ialu:* integer arithm. logic unit | *falu:* floating point arithm. logic unit |
| *ldst:* load/store unit | *rob:* reorder buffer |
| *ic:* instruction L1 cache | *dc:* data L1 cache |
| *ipr:* integer physical registers | *fpr:* floating point physical registers |

Table 2.2: Important parameters for the spec2000 suite. The resource that gives the best performance when maximized is marked with 1, the resource that gives the best performance when maximized in combination with the first one is marked with 2 and the resource that gives the best results with the first two is marked with 3.

| **Spec-int** | iq | fq | ialu | falu | ldst | ic | dc | ipr | fpr | rob |
|---|---|---|---|---|---|---|---|---|---|---|
| gzip-source | | | 2 | | | 3 | 1 | | | |
| vpr-route | | | | | | | 1 | 2 | | 3 |
| gcc-scilab | | | 3 | | | 1 | 2 | | | |
| mcf | 1 | | 2 | | | | | 3 | | |
| crafty | | | 3 | | | 1 | 2 | | | |
| parser | | | 3 | | | 1 | 2 | | | |
| eon-cook | | | 3 | | | 1 | 2 | | | |
| perlbmk-makerand | | | 3 | | | 1 | 2 | | | |
| gap | | | 2 | | | 1 | 3 | | | |
| vortex-3 | | | 3 | | | 1 | 2 | | | |
| bzip2-graphic | | | 3 | | | | 2 | 1 | | |
| twolf | | | 3 | | | 1 | 2 | | | |
| **Spec-fp** | iq | fq | ialu | falu | ldst | ic | dc | ipr | fpr | rob |
| wupwise | | 1 | 3 | | | 2 | | | | |
| swim | | 1 | | | | | | | 2 | 3 |
| mgrid | | | | | | | 1 | | 2 | 3 |
| applu | | 1 | | | 2 | | | | 3 | |
| mesa | | | 3 | | | 1 | 2 | | | |
| galgel | | | | | | | 1 | 3 | 2 | |
| art-110 | | | | | | | 3 | 1 | | 2 |
| equake | | 1 | | | | | | 3 | 2 | |
| facerec | | | 2 | | | 1 | 3 | | | |
| ammp | | 3 | | 2 | | | | 1 | | |
| lucas | | 2 | | 1 | | | | 3 | | |
| fma3d | | | | 3 | | 1 | 2 | | | |
| sixtrack | | 2 | | 1 | | 3 | | | | |
| apsi | | 3 | | | | 1 | 2 | | | |

| | |
|---|---|
| *iq:* integer instruction queue | *fq:* floating point instruction queue |
| *ialu:* integer arithm. logic unit | *falu:* floating point arithm. logic unit |
| *ldst:* load/store unit | *rob:* reorder buffer |
| *ic:* instruction L1 cache | *dc:* data L1 cache |
| *ipr:* integer physical registers | *fpr:* floating point physical registers |

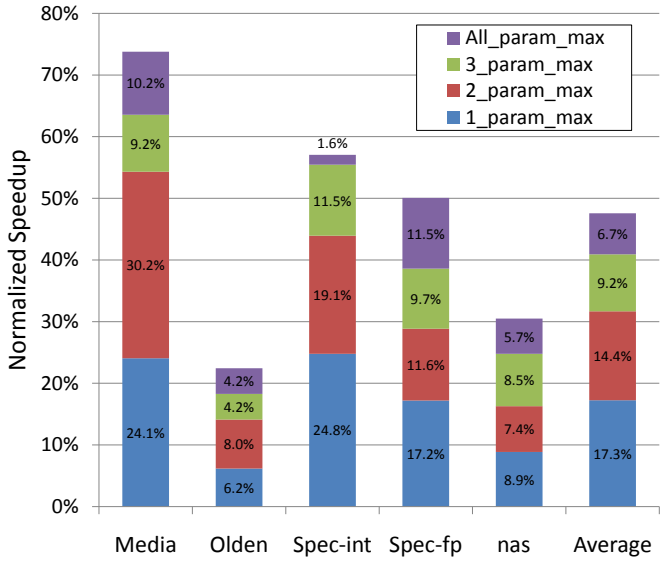



Figure 2.1: Shows the results of several experiments, where potential bottlenecks, or performance constraints, are selectively removed per experiment, for a variety of benchmarks. The stacked bar shows the performance gain when the most important constraint is removed, when two most important constraints are removed, when three most important constraints are removed and when all constraints are removed. Performance gain is relative to a minimally configured processor core.

## 2.3 Adaptive architecture to limit peak power

Current high-performance processors rely on knowledge of expected, or average, application behavior to design packaging and thermal sensors to detect when behavior exceeds thermal bounds. For example, Intel's current power strategy can be seen in [Int00]. For each processor, a set of typical applications are run on the processor, and a power point is identified that is higher than the average power dissipation of most applications. This is the Thermal Design Power (TDP). System designers are expected to create systems that can thermally handle this level of power. When power exceeds this threshold for more than 30 secs, thermal control circuitry is activated. The Intel Pentium 4 processor datasheet [Int00] shows in

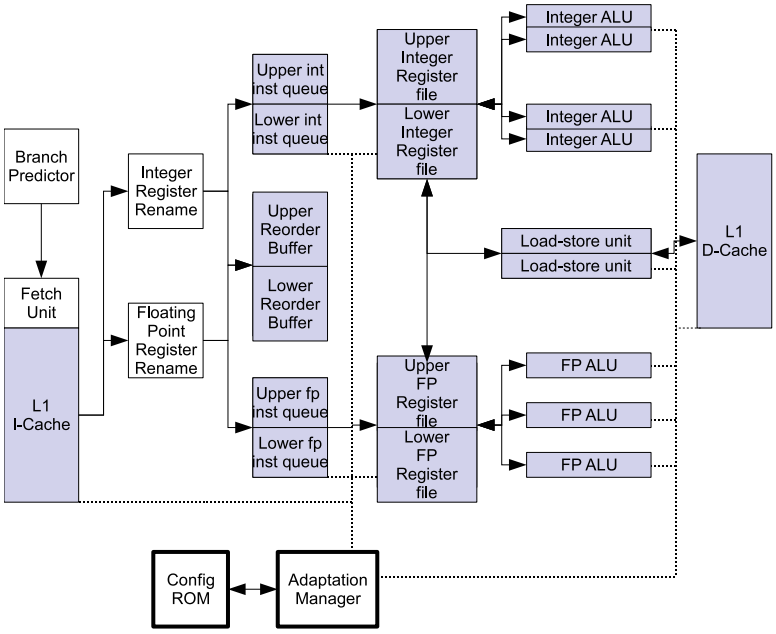



Figure 2.2: The adaptive core architecture. The shaded parts are dynamically configured for power savings and max performance while peak power limits are guaranteed.

figure 13 the performance loss as observed by the user once the thermal control circuit is activated. Such a design does not provide any peak power guarantees (only peak temperature), hence it is still susceptible to a power virus [NKH+07] – an application that maximizes processor power dissipation. Moreover, we cannot use TDP for the power supply design, because unlike temperature, power demand can go to its maximum value in any given cycle. Further, the processor itself, including clock distribution, power delivery circuitry, decoupling capacitors, and other wiring, must be designed for the true peak power, not the TDP.

In contrast to reactive approaches, the architecture described here guarantees that power will never exceed certain thresholds, by ensuring that in any given cycle a certain portion of the processor core is not active. It is impossible for the processor to get into a state where too many resources are being used or in a high power state, even in the presence of a power virus. The same could be achieved by just designing a smaller core; however, some applications will always degrade because of the resources excluded. With an adaptive architecture, we give up little

performance relative to the fully configured core, while achieving the peak power of a much smaller core.

Figure 2.2 shows our architecture. Most of the architecture is configurable. Control of all configurable pieces is managed centrally. Reconfigurations happen at very coarse intervals, so there is no latency issue related to central control. The Config ROM contains a set of configurations that are known not to exceed our peak power threshold. The Adaptation Manager collects performance counter information over an epoch, and based on that data possibly selects a new configuration from the Config ROM for the next epoch.

We assume our CPU is a core of a CMP in 65nm technology. We focus on a single core (including L1 caches) in this study, assuming each core limits peak power individually, resulting in a cumulative reduction in total processor power. The L2 cache could easily be incorporated, as well. For private L2 caches, it would be part of the core power management, and would only increase the gains shown in this chapter. For a shared cache, it would need to be part of a global peak power solution.

Table 2.3 shows the explored architectural design space, including all components we consider configurable. The total number of configurations grows exponentially with the number of adaptable components, making prohibitive any hardware implementation that would need to evaluate a significant fraction of them dynamically.

### 2.3.1 Filling the config ROM

We assume that the Config ROM should contain all the reasonable resource configurations that the core can be in. The first challenge we encounter is what configurations to put in the ROM. This section describes our process.

We determine the peak power for every possible configuration in the design space of Table 2.3. The total number of configurations sums up to 6144, which corresponds to the strict product of the number of all resource configurations. After determining the peak power threshold, we then eliminate all configurations that exceed that threshold. In most cases (depending on the threshold), we are

Table 2.3: Design Space

| | |
|---|---|
| INT instruction queue | 16,32 entries |
| INT registers | 64,128 |
| FP instruction queue | 16,32 entries |
| FP registers | 64,128 |
| INT alus | 2,4 |
| D cache | 1,2,4,8 ways of 4k each |
| FP alus | 1,2,3 |
| I cache | 1,2,4,8 ways of 4k each |
| Load/Store Units | 1,2 |
| Reorder Buffer | 128,256 entries |

still left with a large number of configurations; however, many are redundant and unnecessary. We therefore also eliminate all redundant configurations – those that are a strict subset of another in our list. For example, if configuration B is identical to A, except that A has 8 cache ways and B has 4, B is eliminated. This step provides us with a reasonable set of configurations for all threshold levels. The remaining configurations then go in the Config ROM.

This heuristic makes the implicit assumption that a larger configuration always outperforms a smaller one. We found this to be *almost* always true. The most notable exception was *crafty* which runs up to 11% faster with configurations that are not the largest. With more resources, *crafty* executes further down the wrong path and as a result, wrong path loads and stores generate useless data traffic that occupy elements of the memory hierarchy once the program returns to the correct path. An architecture that wanted to also exploit this phenomenon (able to find an optimal architecture even when it is a strict subset of other available configurations) would need to have more configurations and likely a more complex algorithm for selecting the next configuration — and the additional gains would be small.

We consider several possible power thresholds in this research, although we assume the processor core is configured for a single threshold (section 5 discusses other assumptions). Specifically we consider thresholds corresponding to 70%,

Table 2.4: Distribution of configurations over peak power groups

| Relative power threshold | Number of configurations |
|---|---|
| 0-70 % | 132 |
| 0-75 % | 279 |
| 0-80 % | 360 |
| 0-85 % | 285 |
| 0-inf % | 1 |

75%, 80%, 85% of the core's original peak power, as well as *no limit* which corresponds to the maximally configured core. It is difficult to go much lower than 70%, because much of the core power goes to things that we don't consider configurable. And even the configurable components are never disabled completely.

Table 2.4 shows how many configurations would appear in the table for each power threshold. The storage cost of the table is quite low – we require 13 bits per configuration (2 bits for FP ALUs, Icache and Dcache, 1 bit for the others), so the 75% list would require 454 bytes.

### 2.3.2 Configuration mechanisms

Each epoch, a new configuration is potentially chosen. Section 2.5.2 describes the actual decision mechanisms used to select the next configuration. This section is concerned with the mechanics of switching configurations and adapting individual components once that selection is made.

Reconfiguration is done carefully to ensure no resources are ever added until other resources are reduced. For example, if in the next epoch the Adaptation Manager chooses a configuration with a larger Icache and a smaller integer queue, we must first wait until the instruction queue drains out sufficiently and we have successfully power gated it, and then we can power up the new Icache ways.

We attempt to model these delays accurately and conservatively, and include an additional power-gating delay in order to ensure that powering down and especially powering up is done in a phased way, so that induction noise, ground

bounce, and rush current [GAT03, PPWT00, CGB97] will not be an issue. The adaptable components of the processor can be placed in two categories. Components such as integer and load store units maintain no state and can be partially power-gated immediately after they are signaled by the centralized adaptation mechanism. On the other hand, instruction queues, register files, and data caches must drain state before partial power-gating is allowed.

The literature contains several approaches for L1 cache power reduction. Way-prediction [PAV+01] will reduce dynamic power dissipation on successful predicts, while Drowsy caches [FKM+02] will reduce leakage power by periodically setting the bitcells in a drowsy, low-power mode and cache decay [KHM01] will reduce leakage by selectively power-gating cache lines. Unfortunately, none of the above provide an upper bound on power. We adopt a method similar to the selective cache-ways work [Alb99] with the important difference that the contents of a way are flushed and invalidated before the way is deactivated. Albonesi, et al. keep the contents of the memory cells valid and use or invalidate those on demand; we flush the dirty entries to the higher cache level and then invalidate the whole way. Consequently we can always cap the power activity of the specific cache since deactivated parts of the cache array can not dissipate any leakage or dynamic power. Our approach negatively impacts performance in the short term by increasing the stress on the memory subsystem. However, when the interval between adaptations is sufficiently large, the performance drop is insignificant.

For instruction queue partitioning, we assume a circuit mechanism similar to [BAS+01]. Integer and floating point instruction queues are both clustered in two 16-entry parts. Before disabling a cluster we ensure that all the instructions have issued by using a simple NOR of all active bits of the entries in the cluster. No compaction is assumed. Once the adaptation mechanism decides to shut down an instruction queue cluster, register renaming is throttled. Once the cluster issues all remaining instructions in that partition, we can begin power gating and resume renaming.

The register files and the reorder buffer of our architecture follows the resizing principles described in [DBB+02]. We segment the bitlines of the RAM

Table 2.5: Assumed delays to powergate components

| Component | Power-up Delay (cyc) | Power-down Delay (cyc) |
|---|---|---|
| Dcache | 651 | 163 |
| Icache | 335 | 84 |
| Int inst queue | 127 | 32 |
| FP inst queue | 127 | 32 |
| Int Alus | 198 | 50 |
| FP Alus | 375 | 94 |
| Int reg file | 277 | 69 |
| FP reg file | 277 | 69 |
| rob | 42 | 11 |

components of the various buffers but at a much coarser granularity. Therefore we have a lower and upper cluster of the register file, and a lower and upper cluster of the reorder buffer. When we partially deactivate the structure we always disable the higher part and only after ensuring that the specific entries are not active. For that we again throttle register renaming during the adaptation period until the desired partition becomes idle. Unlike the queues, which always empty out eventually, register file parts may never empty without some intervention – in this case, by injecting some additional instructions to be renamed [DBB+02].

In [HBS+04] the authors provide details on ALU power gating together with several techniques to minimize performance loss once power saving techniques are applied. We assume similar mechanisms but instead use our centralized control.

During the adaptation process handshake signals are exchanged between the Adaptation Manager that decides the next configuration of the processor from those stored in the Config ROM, and different adaptable components. Initially the Adaptation Manager will notify one or more components with a request for power down. The component will then take any necessary action (e.g., flush cache lines, etc.) to ensure safe partial deactivation as described above. After that completes, the unit signals the Adaptation Manager. At that point, the manager can initiate the power gating of the component, which will take some known number of cycles. Only at the end of that period can the Adaptation Manager initiate the power-up

Table 2.6: Delays to disable components holding state

| Component | Min(cyc) | Average(cyc) | Max(cyc) |
|---|---|---|---|
| Dcache | 163 | 384 | 1085 |
| Int inst. queue | 32 | 93 | 1725 |
| FP inst queue | 32 | 65 | 1741 |
| Int reg file | 69 | 101 | 3557 |
| FP reg file | 69 | 106 | 1108 |
| ROB | 11 | 114 | 2254 |

of those resources that will grow. Communication costs are minimal and happen very infrequently.

In Table 2.5 the delays to turn on and off each component are shown. According to [RMD+05], 200 ns suffice to power-up 1.2 million gates in 90nm technology. Ignoring the speed increase that should come with 65nm technology, and assuming linear scaling based on gate counts, our largest adaptive structure should be able to power up in a conservative 350 ns. Hence, powering-up the Dcache ways in the modeled core takes 350ns, and the delay for other components is faster, depending on each component's area. A 4:1 ratio was assumed between the delay to power up and power down a component, since powering down causes practically no inductive noise effect and should be much faster.

Table 2.6 summarizes the minimum, average, and maximum delays observed for powering down structures that hold state. This delay includes the assumed powergating delay and the time to free all occupied entries, but does not include communication costs with the Adaptation Manager.

While we have modeled all of these delays carefully, we show in section 2.5.4 that performance is highly insensitive to these delays.

### 2.3.3 Implementation overhead

The configuration selection mechanisms we describe in section 2.5.2 vary in their complexity – the computation is not complex, but in some cases the set of alternative configurations considered is large. Fortunately, the config ROM

and the adaptation manager – the main additions of our technique – are accessed infrequently and are not part of the critical path.

Leveraging this fact, we can evaluate each configuration in the Config ROM consecutively, minimizing the hardware resources required for the Adaptation Manager. Assuming the most complex heuristic proposed in the results section (see section 2.5), we need to estimate the "closeness vector" and the "weight vector". For the former we need to perform 10 2-bit subtractions (as many as the parameters we consider in each configuration) and for the latter we need to perform 10 23-bit comparisons between measured and threshold values (23 is log of the max interval used between adaptations – 8M cycles). Assuming that these subtractions and comparisons can be done consecutively as well, all we need is a small ALU capable of doing narrow width subtractions, a few additional state bits, and muxes. Our circuit analysis with stardard cells in 65nm technology for this additional computational hardware indicates that it should add well under 1% to the total peak power and less than 0.5% of area. The Config ROM was conservatively modeled as RAM, using CACTI, and introduced less than 0.1% peak power overhead and 0.75% area overhead. In terms of average power the mechanism cost can be ignored since this small specialized engine is activated infrequently.

Given the tolerance of delays associated with this computation, and the frequency at which we adapt, we could even relegate the decision of selecting the best configuration to software. Even if the decision is made in software, we would propose still hardwiring the configurations in the hardware table to ensure that malicious software could never force an illegal configuration.

Design verification and testing will increase for the considered adaptive core compared to a core with fixed-size resources. However, since we only permit a subset of all the possible core configurations testing will be simplified compared to an adaptive processor where configuration decisions are decoupled – such as the proposed architectures for average power reduction.

## 2.4 Methodology

In order to evaluate different adaptation policies that optimize performance for given power budgets, we added support for dynamic adaptation to the SMTSIM simulator [Tul96], integrated with the Wattch power models[BTM00]. Wattch was modified so that the underlying CACTI [TMAP08] models have more updated circuit parameters and a reorder buffer is added to the power modeling.

### 2.4.1 Power modeling

Wattch was developed for relative, activity-factor based power measurements and was originally validated against processors built in early 2000. Hence, it does not capture absolute power consumption for modern architectures but approximates well the relative power trends when the on-chip resources dynamically change. Wattch lacks a leakage power model and does not consider different circuit design styles or power reduction techniques. Our methodology is similar to the one in [KFJ+03] and addresses the above issues. In that work, they apply linear scaling to the output of Wattch for a particular technology, so that it matches published values for a real processor at two points – peak power and average power. If we assume Wattch results reflect relative power changes accurately, this methodology should produce accurate absolute power results for the calibrated processor. Below we present the data used for the scaling. For this methodology, we need real processor peak power and average power, and the peak and average power from Wattch.

We get the Wattch peak power by maximizing all activity factors. We get Wattch typical power simply by running a number of benchmarks through it. For the target processor, we obtained the TDP value from the datasheets of Intel Core Solo at 65nm and assumed the latter as 75% of the peak power [Int00]. We also get the average power from [Int00] – in fact, this paper gives us a range of power values over a workload set which overlaps heavily with ours, allowing us to validate our model even more strongly.

This is only for dynamic power. We also need to add leakage power esti-

Table 2.7: Absolute Power Values for the modeled processor

| | Processor | Core |
|---|---|---|
| Peak Power Value(W) | 36 | 28.57 |
| Average Value(W) | 20.87 | 15.90 |

Table 2.8: Architectural Specification

| | | | |
|---|---|---|---|
| Cores | 1 | I cache | 32k, 8 way |
| Fetch width | 4 | I cache miss penalty | 8 cyc |
| INT instruction queue | 32 entries | D cache | 32k, 8 way |
| FP instruction queue | 32 entries | D cache miss penalty | 8 cyc |
| Reorder Buffer entries | 256 | shared L2 cache | 2 MB, 4 way |
| FP registers | 128 | L2 miss penalty | 40 cyc |
| INT registers | 128 | L3 | 4 MB, 4 way |
| Cache line size | 64 bytes | L3 miss penalty | 315 cyc |
| Frequency | 1.83GHz | Vdd | 1.2V |

mates. Specifically, we assume leakage power to account for 40% of overall typical power and approximately 28% of the peak power [AG] for our 65 nm design. Since leakage scales with area, we break down the assumed value according to area ratios given by [KTJ06]. The L2 area is estimated as 70% of the core area from an Intel Core Solo die photo. The leakage values are additionally scaled linearly when a component is reconfigured dynamically, assuming that we always powergate deconfigured resources.

Table 2.7 gives the absolute peak and average power for the modeled processor as well as the corresponding core (processor excluding L2). Figure 2.3 presents the breakdown of peak and average power to different components. For the estimation of the average power we averaged the power consumption across our set of benchmarks when running on the maximally configured processor. We find that 50% of the total peak power and more than 40% of the average power is being used by our configurable resources, and the remaining power is used by non-configurable components. Because none of our resources are configurable to a size of zero, of course, only a portion of that power can be eliminated.

Table 2.8 gives the characteristics of our baseline architecture.

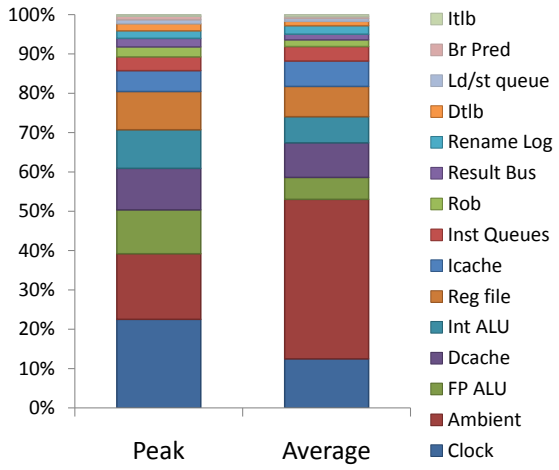



Figure 2.3: The breakdown to core components for the peak and average power. The power reported is comprised of dynamic and leakage power.

### 2.4.2 Benchmarks

In order to explore the benefits of adaptability we use the whole SPEC2000 benchmark suite and a selection (picked randomly) of Media, NAS and Olden benchmarks. Adaptive processors inherently are more effective in matching the varying needs of different programs. Hence, we use 42 benchmarks in total to expose all sorts of different execution behaviors. We use the Simpoint tool [SPHC02] to estimate the proper fast forward distance for up to 4 representative simpoints per benchmark and then we simulate 50M instructions at each simpoint. Multi-simpoint simulations were used to better capture intra-thread phase changes.

## 2.5 Results

In this section, we first explore the potential for exploiting adaptivity for peak power reduction by modeling ideal static and dynamic configuration selection. We then examine real heuristics for navigating the configuration space dynamically.

### 2.5.1 Dynamic adaptation vs static tuning

This section explores the potential of adaptivity to reduce peak power, while still limiting the performance hit introduced from the resource restrictions. Figures 2.4 through 2.6 present results for several oracle configuration policies (at each peak power threshold).

The policies simulated include the static worst per simpoint (WORST_PER_SIMPOINT), the static best per benchmark (BEST_PER_BENCH), and the static best per simpoint (BEST_PER_SIMPOINT). For those, the best (or worst) static configuration is identified for each application individually over the entire simulation interval (recall that our configuration trimming algorithm significantly restricts how bad "worst" can be). The fourth column (BEST_STATIC) corresponds to the best core configuration in the specific peak power chunk across all benchmarks. The fifth column (IDEAL_ADAPT) demonstrates the performance of a core than changes configurations dynamically and always chooses the best configuration for each interval of 1M instructions. Thus, the fourth bar represents potential performance from just designing the best possible non-configurable core within the peak power constraint. The second bar represents the potential from adaptivity to exploit inter-application diversity. The difference between the second and fourth bars represents the potential due to intra-application diversity. All the results were normalized to the BEST_STATIC, because this represents the best possible non-adaptive processor you could build within this peak power constraint, and constitutes our baseline. The last column (MAX_CONF) represents the fully configured core, which does not share the same peak power constraint as the rest of the core options in this graph.

Several conclusions come out of these figures. As expected the more limited the peak power budget the bigger the gains from adaptivity. For the lowest peak power budget an ideal adaptive core would perform on average 16% better than a core with fixed-size resources and tuned for best results (BEST_STATIC), while for the highest peak power budget the benefits are much smaller.

We also see that even with a tight peak power budget, we are able to get

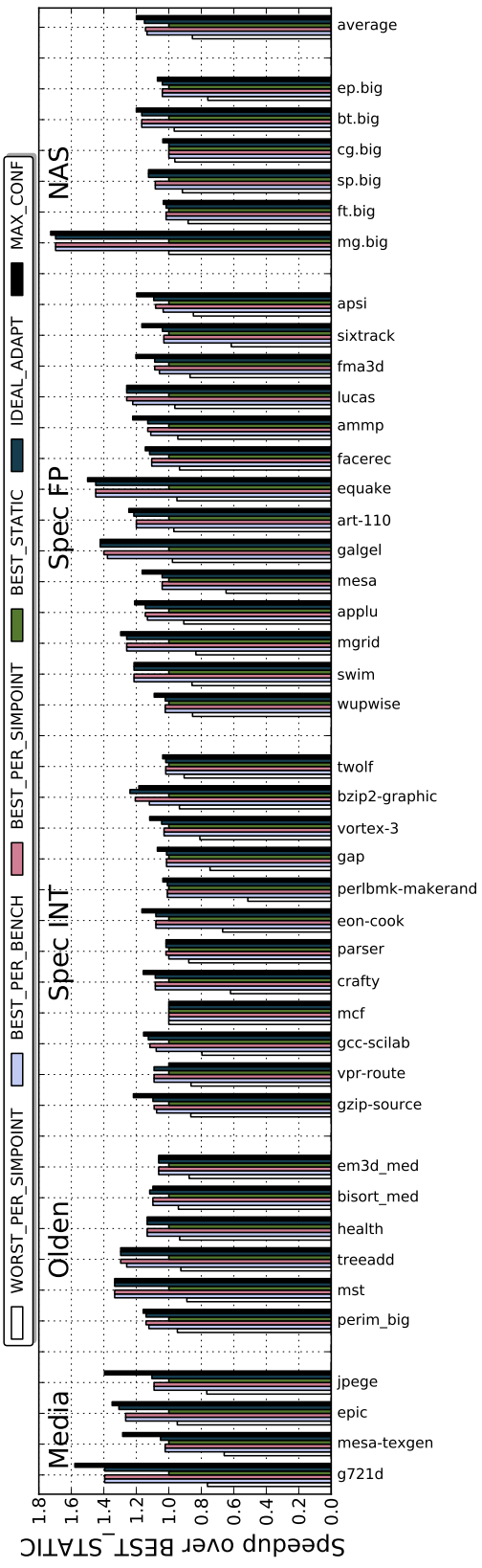



Figure 2.4: The performance of six configuration policies with a peak power constraint of 70% of the maximum configuration. The best static configuration is: iqs:32 fqs:32 ialu:2 falu:1 ldst:1 ics:16 dcs:16 ipr:64 fpr:64 rob:256

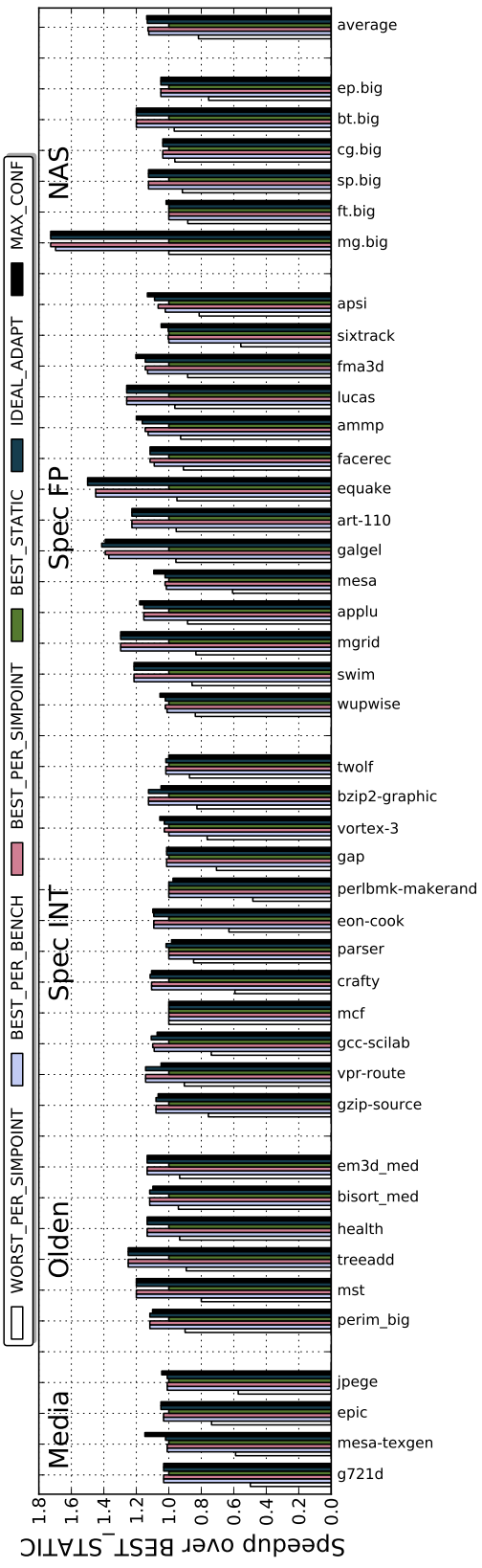



Figure 2.5: The performance of six configuration policies with a peak power constraint of 75% of the maximum configuration. The best static configuration is: iqs:32 fqs:32 ialu:4 falu:1 ldst:1 ics:16 dcs:16 ipr:64 fpr:64 rob:128

very close to the same performance as the full core. This confirms that most applications require just a few key resources. Examples include the g721 Media benchmark which demands high number of integer ALUs, galgel which needs a high number of integer and floating pointing physical registers, equake which needs a high number of floating point queue entries, and mg which requires floating point registers and reorder buffer entries.

But the bottleneck resources vary, and the best static core cannot give each of these applications what it needs most, while the adaptive core can. The configuration of the best static core will depend on the selection of benchmarks, of course, and how they might be prioritized. But this is another advantage of the adaptive architecture. Whereas the static architecture (the way we've always designed processors) will always be sensitive to the particular benchmarks used to optimize the design, the adaptive architecture is not.

Also, a larger selection of benchmarks, such as are used in real processor design, would almost certainly result in a larger gap between the compromise (best static) architecture and the adaptive architecture.

As we relax the peak power budget (for example, at 75%), we see a somewhat smaller gain from adaptivity over the static configuration, but we are able to nearly replicate the performance of the full core. At 80% of peak power the differences are smaller, but still noticeable. For an 85% threshold (results not shown), the best static core is pretty competitive with our adaptive core.

The use of multiple simpoints and the between-simpoint adaptation makes a significant difference in only gcc and bzip2. This indicates that overall we gain much more from inter-thread diversity than from intra-thread diversity.

Also notice in these results that we see the aforementioned behavior for *crafty*, and to a lesser extent for *vpr* — that they achieve higher results with a configuration smaller than the largest. Again, this is a result of being able to more aggressively pursue wrong branch paths with more resources, and is in general an anomalous result. Interestingly, this effect enables, at the higher peak thresholds (80% and 85%), the ideal adaptive technique to actually outperform the maximally configured core (by a small amount).

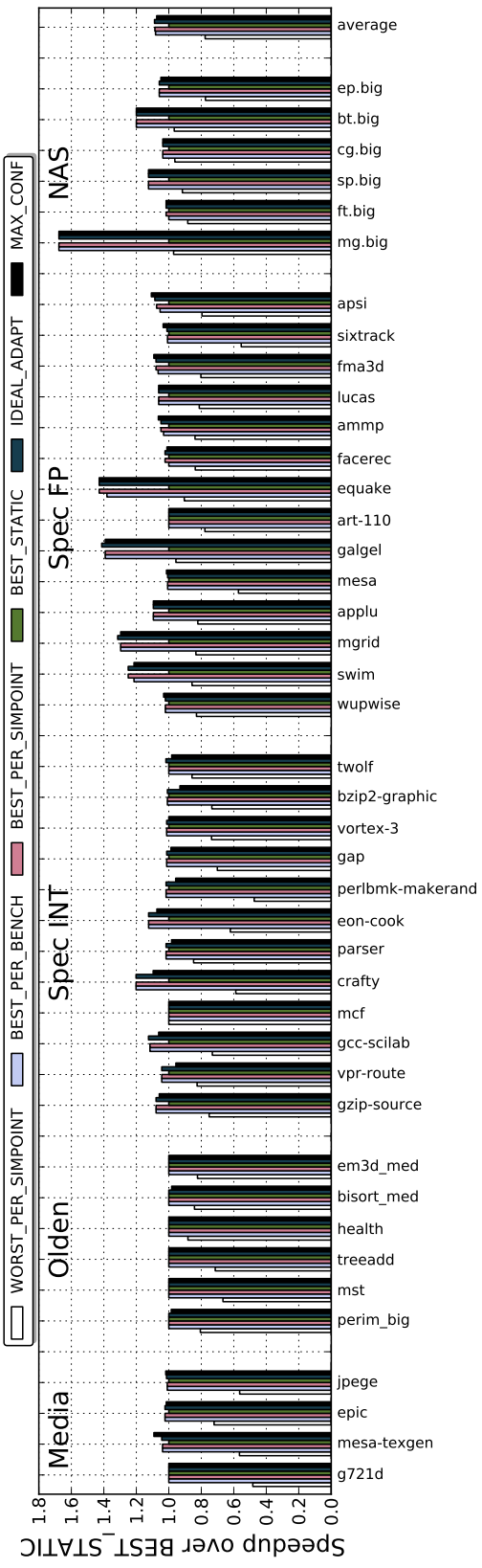



Figure 2.6: The performance of six configuration policies with a peak power constraint of 80% of the maximum configuration. The best static configuration is: iqs:32 fqs:32 ialu:4 falu:1 ldst:1 ics:16 dcs:16 ipr:128 fpr:64 rob:256

One measure of core design "goodness" is the ratio of average power to peak power. A low ratio implies an architecture that requires the system (including the processor design, the cooling, packaging, the power supply, etc.) be ever-provisioned for the delivered performance. A high ratio, then, implies a more efficient design. Figure 2.7 presents the average power and peak power for the IDEAL_ADAPT architecture.

This architecture, then, shows two very positive trends, it reduces both average power and the ratio of peak to average power at the same time. It should be noted, in fact, that we have moved the ratio of peak to average power to a point that is more typical of an in-order processor core than the out-of-order core we are simulating [KTJR05]. Again, this enables the use of a more efficient power supply (better targeted at the average power), saving wall power.

### 2.5.2 Realistic adaptive techniques

In order for the adaptation manager to approach the potential we have seen for our adaptive processor, it needs to constantly navigate the list of potential configurations in the config ROM, hopefully staying close to the optimal configuration at all times.

Specifically, the configuration manager has to address three issues: *When should we should trigger a reconfiguration?* We should be attempting to reconfigure at least often enough to capture major program phase changes, as well as transitions between applications (context switches). *Which configuration from the ROM is the most appropriate to run next?* The ROM stores more than 100 configurations for each power chunk. Being able to distinguish the configurations that are more likely to perform better becomes critical. Finally, *how do we evaluate the configuration chosen?* For the latter we need a notion of whether performance has improved or declined as a result of a reconfiguration decision.

We examine several policies for effectively navigating the configuration space. Each is described as a 3-tuple, corresponding to the questions in the previous paragraph. Thus, the naming convention is < adaptation triggering mechanism >_< configuration selection method >_< evaluation method >. We show results

for the following policies:

*INTV_RANDOM_NONE*: After a predetermined interval of cycles a different configuration is chosen randomly. There is no feedback mechanism or evaluation of the configurations. This is a simple policy and it is not expected to perform well. But it is a useful result in that it essentially gives us the "expected" behavior of the set of all the potentially good static architectures.

*INTV_SCORE_NONE*: After a predetermined interval of cycles, a different configuration is chosen according to a score-based technique. We keep statistics for conflict events of each resource that is dynamically configured. A conflict occurs when there is contention for a resource. Specifically, we maintain instruction queue conflicts (an instruction cannot enter the instruction queue because it is full), floating point queue conflicts, integer register conflicts (an instruction cannot be renamed because there are no available renaming registers), floating point register conflicts, integer ALU conflicts, floating point ALU conflicts, reorder buffer conflicts, Icache misses, and Dcache misses. To have a notion of which resource is most required we maintain the ratio of conflicts per cycle (we also tried conflicts per instruction, with no significant performance difference observed). We form a vector indicating the components that exceeded thresholds we set based on experimentation (weight_vector). Then, we form another vector which describes how closely a configuration under consideration relates to the current configuration (closeness_vector) – this vector can have both positive and negative values. The closeness vector is simply the result of subtracting the two entries (the configuration under consideration from the current configuration) from the config ROM. For example if we consider 1,2,4,8 ways caches and the current configuration has 2 way-enabled cache while the configuration under consideration has 4, the element of the closeness vector for that cache would be 1. If instead of 4 there were 8 ways the closeness element would be 2. For a configuration with a 1-way cache, the closeness element would be -1. The total score for each configuration is found by multiplying the two vectors and summing the elements, as illustrated in Table 2.9.

Once all the scores are estimated, the configuration with the highest score is selected. There are frequent ties, and so the selection is then made randomly.

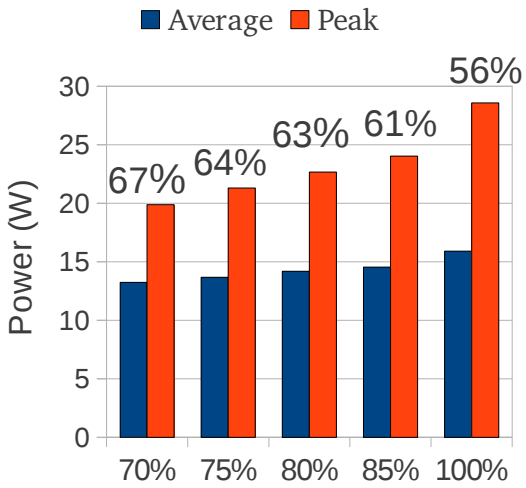



Figure 2.7: The average and peak power of our idealized adaptive core. Above the bars is shown the ratio of peak to average power.

This policy makes a more educated configuration selection by favoring configurations that increase the resources with conflicts. However, it still lacks a feedback mechanism to evaluate the configuration selected.

*INTV_SCORE_SAMPLE*: This is the same policy as the previous one with the addition of sampling as a feedback mechanism. Every interval of cycles a series of potential configurations are chosen based on the scoring system and the weights given by the execution of the previous configuration. The top n are chosen for sampling, again with random selection breaking ties. Experimentally, we found n=5 to work well. After all configurations are run for a sampling interval, the best IPC wins (the previous configuration's execution IPC is also considered) and that configuration runs until the next reconfiguration interval. The reconfiguration interval is set to keep a 1:10 ratio between the sampling time and the execution time. To prevent any transitional or cold start effects there is a warm up period before data is collected during a sampling interval.

*EVDRIV_SCORE_SAMPLE*: This policy does not consider adaptation periodically. Instead adaptations are considered only when there is evidence of a

Table 2.9: Example of configuration score estimation

| | iq size | fq size | ialus | falus | ldst units | iregs | fregs | IC ways | DC ways | rob |
|---|---|---|---|---|---|---|---|---|---|---|
| Current config | 32 | 16 | 2 | 1 | 2 | 128 | 128 | 4 | 1 | 128 |
| Considered config | 16 | 16 | 4 | 3 | 1 | 64 | 128 | 1 | 2 | 128 |
| Closeness vector | -1 | 0 | 1 | 2 | -1 | -1 | 0 | -2 | 1 | 0 |
| Weight vector | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 1 |
| Results | -1 | 0 | 0 | 2 | 0 | -1 | 0 | 0 | 1 | 0 |
| Total score | 1 | | | | | | | | | |

change in application behavior (event-driven). In particular, if we detect that over the previous interval, the measured IPC, or the cache behavior (misses per cycle), changed by a relative margin of 30%, then we initiate a reconfiguration evaluation. The frequency of adaptations is limited by an upper (8M cycles) and lower bound (2M cycles).

*INTVAD_SCORE_SAMPLE*: With this policy we adapt the interval between reconfiguration sampling based on how useful reconfiguration is. When we sample and find a better configuration than the previous, we cut the time to the next reconfiguration in half. When we fail to find a better configuration, we double the interval (adaptive interval). In this way, we search quickly when we are far from optimal, and minimize sampling overhead when we are near it. Again there is a minimum bound (2M cycles) and a maximum bound (8M cycles) for the interval.

In figure 2.8 we present the speedup over the best static configuration achieved with the different adaptive policies applied to the lowest peak power bound. We experimented with interval values ranging from 0.5M to 10M. The policies with fixed interval produced the best results with an interval of 2M cycles, while the policies with variable interval perform best with lower bound 2M and upper 8M cycles. We observe that randomly selecting a configuration from the ROM performs significantly worse than the best static configuration. This is the difference between the average static configuration and the optimal static configuration. With the addition of *score* to make more educated selections the performance is on average equivalent to the best static configuration. The evaluation that comes with sampling gives a major boost to performance since bad

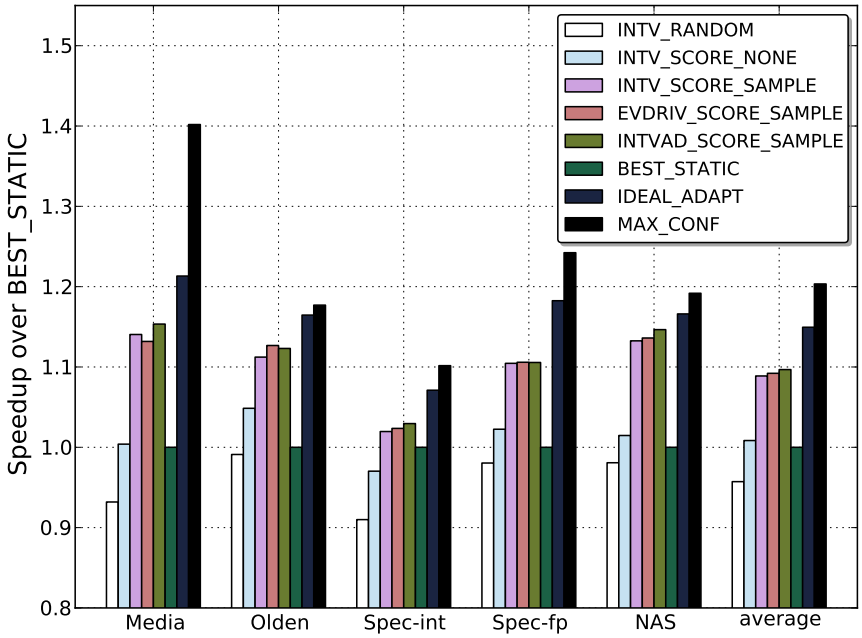



Figure 2.8: The performance of different adaptive policies with respect to the best static configuration

configurations are avoided. Finally, the dynamic interval techniques provide some additional gains – the adaptive interval doing better than the event-driven (phase detection) mechanism. The event-driven mechanism will periodically monitor instruction throughput and level one cache miss rates and will trigger an adaptation when a relative change of more than 30% is observed. The problem with the phase detection heuristic is that in its current form, when it encounters an abrupt phase change, it gets just one chance to find a good configuration before it settles in for a long interval, and adapts very slowly after that.

We see that with our best dynamic technique, we cover almost 10% out of the potential 15% speedup of the oracle adaptive technique over the best static. Overall with the best realistic adaptive approach we perform 10% worse than the maximally configured processor. Thus we have shown that we can reduce the peak power of a processor core by 30%, only sacrificing 10% in performance. The best static configuration with the same peak power would have a 20% performance hit. At 75% peak power, the performance loss is less than 5%.

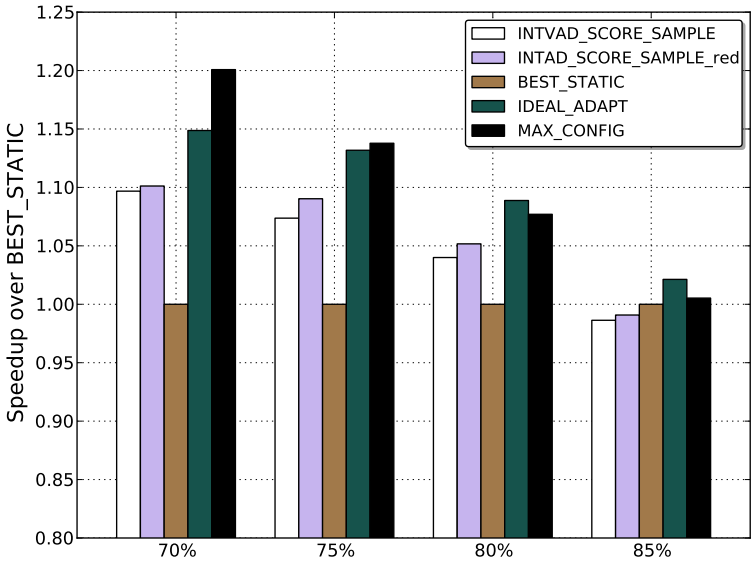



Figure 2.9: Comparison of full and reduced ROM for different peak power constraints

### 2.5.3 Reducing ROM configurations

There are two costs to having a large number of configurations in the config ROM. One is storage, although that cost is relatively low. The second cost is the difficulty in finding the optimal configuration. We can further pare down the configurations experimentally, separating those often found to be worthwhile from those rarely selected. For this experiment, we do this using the oracle adaptive experiments presented in section 2.5.1. We eliminate from the config ROM only those configurations *never* selected as the best configuration for even a single interval for any benchmark. This is a fairly conservative filter. Even still, we cut the number of configurations by more than half (see Table 2.10). Note that there is some danger to this optimization, if the actual workload is not well represented by the benchmark set used to pare down the configurations.

Figure 2.9 presents the best adaptive policy (INTVAD_SCORE_ SAMPLE) with the full ROM and the reduced ROM for different peak power thresholds. As expected, reducing the number of configurations improves our performance, inching us even closer to the ideal result. At 75% with the inclusive set of configurations we

Table 2.10: Distribution of reduced set configurations over peak power groups

| Relative power threshold | Number of configurations |
|---|---|
| 0-70 % | 71 |
| 0-75 % | 97 |
| 0-80 % | 119 |
| 0-85 % | 116 |
| 0-inf % | 1 |

Table 2.11: Peak power impact on voltage variation and on-chip decoupling capacitor area

| Power threshold | Experiment 1<br>On-chip decap (% Core Area) | Experiment 2<br>Max. Volt. Variation (% $V_{DD}$) |
|---|---|---|
| 0-70 % | 9% | 4.48% |
| 0-75 % | 9.7% | 4.80% |
| 0-80 % | 10.5% | 5.12% |
| 0-85 % | 11.5% | 5.44% |
| 0-inf % | 15% | 6.48% |

perform 6.4% worse than the maximally configured core, while with the reduced set we do 4.8% worse. At 80%, our realistic adaptation policy with the reduced set of configurations is only 2.5% worse than MAX_CONF. In many scenarios, trading 5% performance for a 25% reduction in peak power, or 2.5% for a 20% reduction, and the cost and real power savings that go with it, represents a huge win.

It is worth noting that DVFS, proposed for peak power budgeting, sees more than double the performance hit when achieving similar peak power savings (Table 1 in [MJDS08]).

### 2.5.4 Delay sensitivity analysis

This section examines the impact of the assumed reconfiguration latency on our results. The adaptive policy used was INTVAD_SCORE_SAMPLE. To change the duration of adaptation we multiplied all the parameters of Table 2.5 with different factors as demonstrated in figure 2.10. The “No delay” bar corresponds to the unrealistic scenario where component powergating happens instantly – note

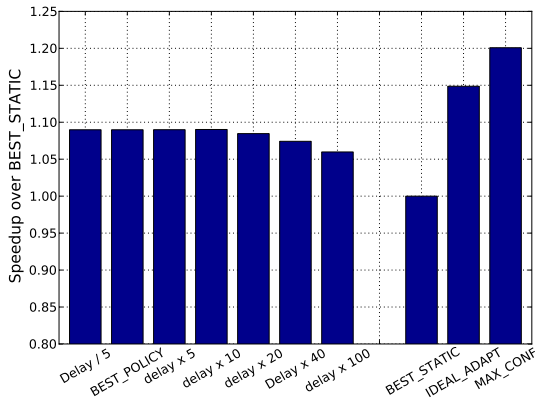



Figure 2.10: Sensitivity analysis of INTVAD_SCORE_SAMPLE to the delay of adaptation

that adaptation still is not instant since we wait for instruction queues to drain or dirty L1 entries to be flushed to the L2. For any reasonable assumption about these delays, the impact on performance is negligible. Undoubtedly, the adaptive interval optimization contributes to this, since on average the interval lengths were larger than the rest of the techniques.

### 2.5.5 Quantifying the benefits of peak power reduction

In this section, we analyze the impact of the peak power reduction on silicon area required for on-chip decoupling capacitors and the voltage variation. The voltage variation in sub-65nm technologies should be kept below 5% of the nominal supply voltage $V_{DD}$ [AP07]. We model a distributed power distribution network with power and ground network RLCG parasitics based on [SHP+09, PR08] for the total processor die area of $26.25mm^2$ as shown in figure 2.11. The core current demand is modeled based on the peak current demand – peak power divided by the supply voltage – and the clock frequency and is distributed as time-variant current sources over the power and ground mesh. The distributed current sources are modeled as triangular waveforms with the clock frequency period ($\frac{1}{f_{clk}=1.83GHz}$) and the rise and fall time of $10 \times f_{clk}$ [Uni04, YCH07].

To calculate the amount of necessary decoupling capacitances required to

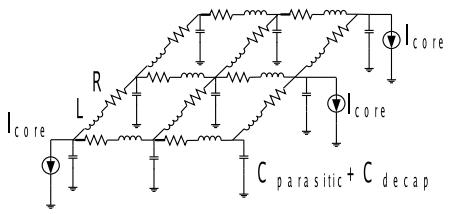



Figure 2.11: Distributed power distribution network model.

maintain a voltage fluctuation within 5%, we use the following first order approximation: $P = C_T \cdot {V_{DD}}^2 \cdot f \cdot p_{0-1}$ where $P$ is the total chip peak power consumption, $V_{DD}$ denotes the supply voltage, $f$ is the clock frequency, $C_T$ is the on-chip decoupling capacitance, and $p_{0-1}$ is the probability that a $0-1$ transition occurs [SHP+09]. $C_T$ includes the intrinsic decoupling capacitance which is usually small and the intentional decoupling capacitance which we add and for which we report area numbers [PMF08].

We find that peak power reduction saves significant die area for the intentional on-chip decoupling capacitors and minimizes the voltage drop at the same time. We perform the transient analysis based on [SHP+09] to find the minimum on-chip decoupling capacitors which are required based on 65-nm thin oxide technology from the initial estimation. Table 2.11 illustrates the experimental data for the on-chip decoupling capacitor estimation. We ran two different analyses. Experiment 1 illustrates that to maintain voltage variation below 5% of the nominal $V_{DD}$, the reduction of peak power to 75% of the original value reduces the amount of required on-chip decoupling capacitor area by 5.3% (of total core area). Experiment 2 demonstrates that if we instead maintain the same amount of on-chip decoupling capacitors (in this test case $150nF$) the voltage variation is suppressed significantly when we reduce the peak power. For the 75% threshold, the variation is reduced by 26%.

## 2.6 Conclusion

This chapter describes an adaptive processor that uses table-driven reconfiguration to place a cap on peak core power dissipation. By only allowing config-

urations specified in this table, the core is ensured to always operate below a given power threshold. While it is impossible to configure all resources at full capacity, each application can find a configuration that provides the resources it most needs at full capacity. In this way, we minimize the performance loss while providing significant peak power savings. This technique is shown to enable a 25% reduction in peak power with less than a 5% performance cost. Additionally, it can outperform the best static design at the same peak power threshold by 9%. The Peak power reduction translates to 5.3% less total silicon for decoupling capacitance or a 26% reduction in voltage variation for the same decoupling capacitance. Furthermore, the design of the chip voltage supply becomes cheaper and more efficient.

# Acknowledgments

Chapter 2 contains material from Reducing peak power with a table-driven adaptive processor core, by Vasileios Kontorinis, Amirali Shayan, Rakesh Kumar, and Dean Tullsen, which appears in *Proceedings of the 42nd annual International Symposium on Microarchitecture (MICRO)*. The dissertation author was the primary investigator and author of this paper. The material in this chapter is copyright ©2009 IEEE. Personal use of this material is permitted. However, permission to reprint/republish this material for advertising or promotional purposes or for creating new collective works for resale or redistribution to servers or lists, or to reuse any copyrighted component of this work in other works must be obtained from the IEEE.

# Chapter 3

# Removing over-provisioning in 3D stacked chips

In the previous chapter we have discussed a core architecture that dynamically tailors core resources to reduce over-provisioning. This core design can be used in the context of multi-cores. An alternative approach to reduce over-provisioning is to design a system with fewer resource to begin with. Then share those resources to boost performance. In this chapter we will discuss how chip integration using 3D technology can help towards reducing over-provisioning.

Prior research [KFJ+03, KTR+04] has shown that heterogeneous multicore architectures provide significant advantages in enabling energy-efficient or area-efficient computing. It allows each thread to run on a core that matches its resource needs more closely than a single one-size-fits-all core. However, that approach still constrains the ability to optimally map executing threads to cores because it relies on static heterogeneity, fixed at design time.

Other research attempts to provide dynamic heterogeneity, but each face a fundamental problem. Either the pipeline is tightly constructed and the resources we might want to share are too far away to be effectively shared, or the shared resources are clustered and the pipeline is inefficient. As a result, most provide resource sharing or aggregation at a very coarse granularity – Core Fusion [IKKM07] and TFlex [KSG+07] allow architects to double or quadruple the size of cores, for example, but do not allow a core to borrow renaming registers from another core

if that is all that is needed to accelerate execution. Thus, the heterogeneity is constrained to narrow cores or wide cores, and does not allow customization to the specific needs of the running thread. The WiDGET architecture [WDW10] can only share execution units, and thus enables only modest pipeline inflation. The conjoined core architecture [KJT04] shares resources between adjacent cores, but sharing is limited by the topology of the core design to only those structures around the periphery of the pipeline.

This work demonstrates that 3D stacked processor architectures eliminate the fundamental barrier to dynamic heterogeneity. Because of the extra design dimension, we can design a tight, optimized pipeline, yet still cluster, or pool, resources we might like to share between multiple cores.

3D die stacking makes it possible to create chip multiprocessors using multiple layers of active silicon bonded with low-latency, high-bandwidth, and very dense vertical interconnects. 3D die stacking technology provides very fast communication, as low as a few picoseconds [LXB07], between processing elements residing on different layers of the chip. Tightly integrating dies in the third dimension has already been shown to have several advantages. First, it enables the integration of heterogeneous components such as logic and DRAM memory [LXB07], or analog and digital circuits [LXB07], fabricated in different technologies (for instance integration of a 65nm and a 130nm design). Second, it increases the routability [SAT+08]. Third, it substantially reduces wire length, which translates to lowered communication latency and reduced power consumption [SAT+08, MZM+09, LXB07].

The dynamically heterogeneous 3D processors we propose in this chapter provide several key benefits.

First, they enable software to run on hardware optimized for the execution characteristics of the running code, even for software the original processor designers did not envision. Second, they enable us to design the processor with compact, lightweight cores without significantly sacrificing general-purpose performance. Modern cores are typically highly over-provisioned [KJT04] to guarantee good general-purpose performance – if we have the ability to borrow the specific

resources a thread needs, the basic core need not be over-provisioned in any dimension. Third, the processor provides true general-purpose performance, not only adapting to the needs of a variety of applications, but also to both high thread-level parallelism (enabling many area-efficient cores) and low thread-level parallelism (enabling one or a few heavyweight cores).

With a 3D architecture, we can dynamically pool resources that are potential performance bottlenecks for possible sharing with neighboring cores. The StageNet architecture [GFA+08] attempts to pool pipeline stage resources for reliability advantages. In that case, the limits of 2D layout mean that by pooling resources, the pipeline must be laid out inefficiently, resulting in very large increases in pipeline depth. Even Core Fusion experiences significant increases in pipeline depth due to communication delays in the front of the pipeline. With 3D integration, we can design the pipeline traditionally in the 2D plane, yet have poolable resources (registers, instruction queue, reorder buffer, cache space, load and store queues, etc.) connected along the third dimension on other layers. In this way, one core can borrow resources from another core or cores, possibly also giving up non-bottleneck resources the other cores need. This work focuses on the sharing of instruction window resources.

This architecture raises a number of performance, energy, thermal, design, and resource allocation issues. This chapter represents a first attempt to begin to understand the various options and trade-offs.

This chapter is organized as follows. Section 3.1 describes our 3D architecture assumptions, both for the baseline multicore and our dynamically heterogeneous architecture. Section 3.2 shows that both medium-end and high-end cores have applications that benefit from increased resources, motivating the architecture. Section 3.3 details the specific circuits that enable resource pooling. Section 3.4 describes our runtime hardware reallocation policies. Section 3.5 describes our experimental methodology, including our 3D models. Section 3.6 gives our performance, fairness, temperature, and energy results. Section 3.7 describes related work.

## 3.1 Baseline architecture

In this section, we discuss the baseline chip multiprocessor architecture and derive a reasonable floorplan for the 3D CMP. This floorplan is the basis for our power/temperature/area and performance modeling of various on-chip structures and the processor as a whole.

3D technology, and its implications on processor architecture, is still in the early stages of development. A number of design approaches are possible and many have been proposed, from alternating cores and memory/cache [LNR+06, MZM+09], to folding a single pipeline across layers [PL06].

In this research, we provide a new alternative to the 3D design space. A principal advantage of the dynamically heterogeneous 3D architecture is that it does not change the fundamental pipeline design of 2D architectures, yet still exploits the 3D technology to provide greater energy proportionality and core customization. In fact, the same single design could be used in 1-, 2-, and 4-layer configurations, for example, providing different total core counts and different levels of customization and resource pooling. For comparison purposes, we will compare against a commonly proposed approach which preserves the 2D pipeline design, but where core layers enable more extensive cache and memory.

### 3.1.1 Processor model

We study the impact of resource pooling in a quad-core CMP architecture. This does not reflect the limit of cores we expect on future multicore architectures, but a reasonable limit on 3D integration. For example, a design with eight cores per layer and four layers of cores would provide 32 cores, but only clusters of four cores would be tightly integrated vertically. Our focus is only on the tightly integrated vertical cores.

For the choice of core we study two types of architecture, a high-end architecture which is an aggressive superscalar core with issue width of 4, and a medium-end architecture which is an out-of-order core with issue width of 2. For the high-end architecture we model a core similar to the Alpha 21264 (similar in

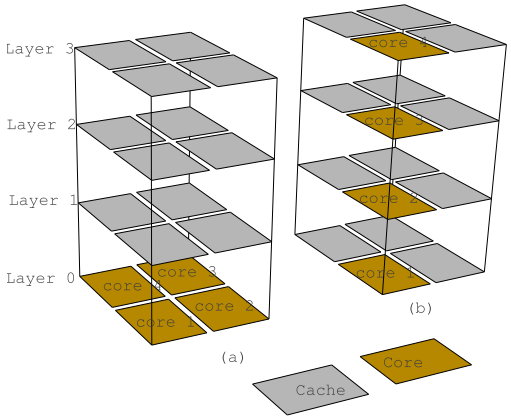



Figure 3.1: CMP configurations: (a) baseline and (b) resource pooling.

functionality to the Intel Nehalem Core, but we have more data available for validation on the 21264). For the medium-end architecture we configure core resources similar to the IBM PowerPC-750 FX processor [IBM03].

### 3.1.2 3D floorplans

The high-level floorplan of our 3D quad-core CMP is shown in figure 3.1. For our high-end processor we assume the same floorplan and same area as the Alpha 21264 [KMW98] but scaled down to 45nm technology. For the medium-end architecture we scale down the Alpha 21264 floorplan (in 45nm) based on smaller components in many dimensions, with area scaling models similar to those described by Burns and Gaudiot [BG01].

Moving from 2D to 3D increases power density due to the proximity of the active layers. As a result, temperature is always a concern for 3D designs. Temperature-aware floorplanning has been an active topic of research in the literature. There have been a number of 3D CMP temperature-aware floorplans proposed [HVE+07, PL07, CAH+10]. Early work in 3D architectures assumed that the best designs sought to alternate hot active logic layers with cooler cache/memory layers. More recent work contradicts that assumption – it is more important to put the active logic layers as close as possible to the heat sink [ZXD+08]. Therefore, an architecture that clusters active processor core layers tightly is consistent with

this approach. Other research has also exploited this principle. Loh, et al. [LXB07] and Intel [BNWS04] have shown how stacking logic on logic in a 3D integration could improve the area footprint of the chip, while minimizing the clock network delay and eliminating many pipeline stages.

For the rest of this work we focus on the two types of floorplan shown in figure 3.1(a) and figure 3.1(b). Both preserve the traditional 2D pipeline, but each provides a different performance, flexibility, and temperature tradeoff.

The thermal-aware architecture in figure 3.1(a) keeps the pipeline logic closest to the heat-sink and does not stack pipeline logic on top of pipeline logic. Conversely, the 3D dynamically heterogeneous configuration in figure 3.1(b) stacks pipeline logic on top of pipeline logic, as in other performance-aware designs, gaining increased processor flexibility through resource pooling. Notice that this comparison puts our architecture in the worst possible light – for example, a many-core architecture that already had multiple layers of cores would have very similar thermal characteristics to our architecture without the benefits of pooling. By comparing with a single layer of cores, the baseline has the dual advantages of not having logic on top of logic, but also putting all cores next to the heat sink.

## 3.2 Resource pooling in the third dimension

Dynamically scheduled processors provide various buffering structures that allow instructions to bypass older instructions stalled due to operand dependences. These include the instruction queue, reorder buffer, load-store queue, and renaming registers. Collectively, these resources define the instruction scheduling window. Larger windows allow the processor to more aggressively search for instruction level parallelism.

The focus of this work, then, is on resource adaptation in four major delay and performance-critical units – the reorder buffer, register file, load/store queue, and instruction queue. By pooling just these resources, we create an architecture where an application's scheduling window can grow to meet its runtime demands, potentially benefiting from other applications that do not need large windows.

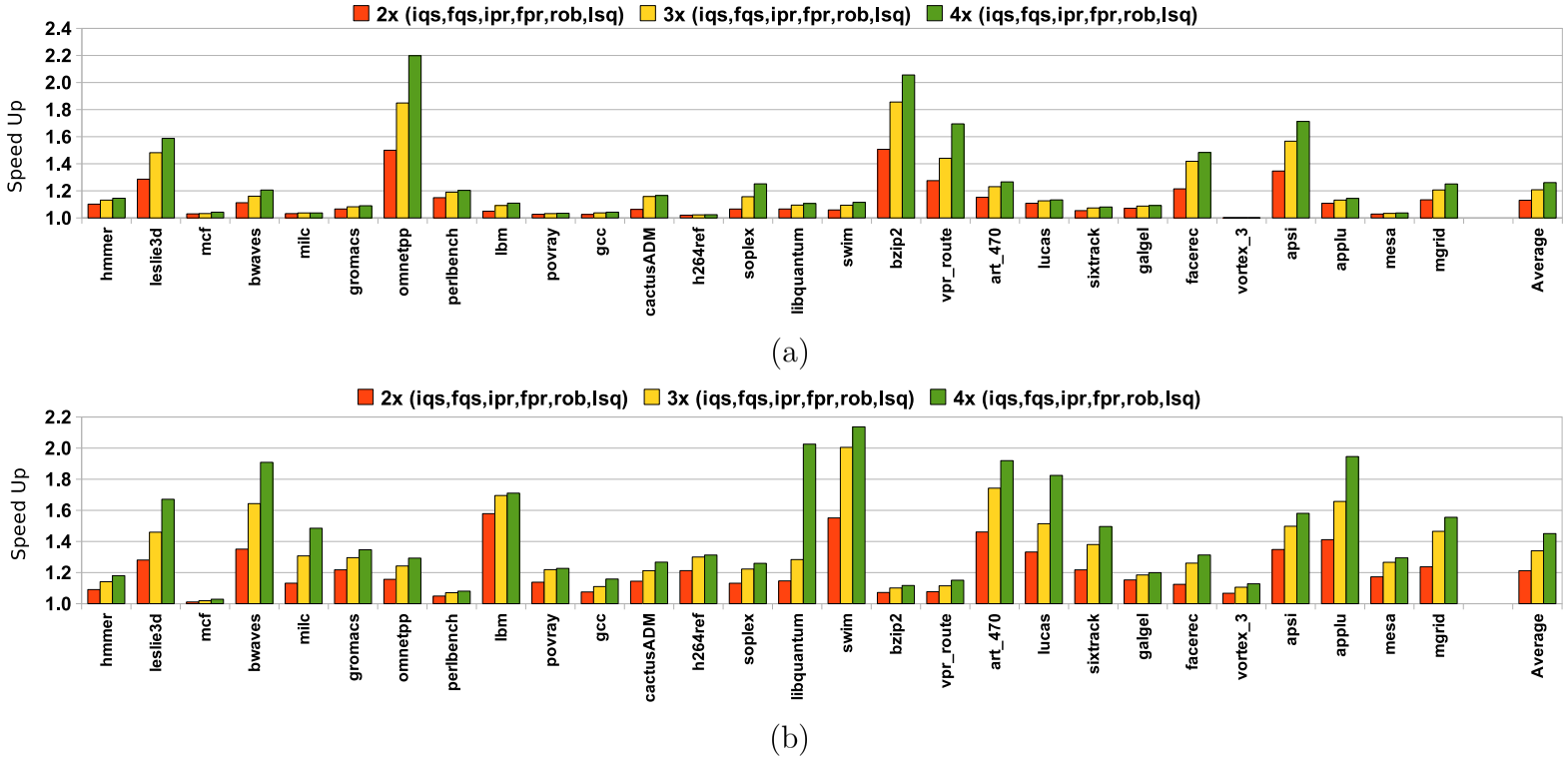



Figure 3.2: Speedup from increasing resource size in the 3D stacked CMP with (a) medium-end and (b) high-end cores.

While there are a variety of resources that could be pooled and traded between cores (including execution units, cache banks, etc.), we focus in this first study of dynamically heterogeneous 3D architectures on specific circuit techniques that enable us to pool these structures, and dynamically grow and shrink the allocation to specific cores.

In this section, we study the impact on performance of increasing the size of selected resources in a 3D design. We assume 4 cores are stacked on top of each other. The maximum gains will be achieved when one, two, or three cores in our 4-core CMP are idle, freeing all of their poolable resources for possible use by running cores. The one-thread case represents a limit study for how much can be gained by pooling, but also represents a very important scenario – the ability to automatically configure a more powerful core when thread level parallelism is low. This does not represent an unrealistic case for this architecture – in a 2D architecture, the cost of quadrupling, say, the register file is high, lengthening wires significantly and moving other key function blocks further away from each other. In this architecture, we are exploiting resources that are already there, the additional wire lengths are much smaller than in the 2D case, and we do not perturb the 2D pipeline layout.

We examine two baseline architectures (details given in section 3.5) — a 4-issue high-end core and a 2-issue medium-end core. In figure 3.2 we report the speedup for each of these core types when selected resources are doubled, tripled, and quadrupled (when 1, 2, and 3 cores are idle). Across most of the benchmarks a noticeable performance gain is observed with pooling. Omnetpp shows the largest performance benefit in medium-end cores. The largest performance is observed in swim and libquantum for high-end cores.

Performance gains are seen with increased resources, but the marginal gains do drop off with larger structures. Further experiments (not shown) indicate that pooling beyond four cores provides little gain. The more scheduling resources we provide, the more likely it is that some other resource (e.g., the functional units, issue rate, cache) that we are not increasing becomes the bottleneck. In fact, this is true for some benchmarks right away, such as mcf and perlbench, where

no significant gains are achieved, implying some other bottleneck (e.g., memory latency) restricts throughput. On average, 13 to 26% performance improvement can be achieved for the medium-end processor, and 21 to 45% for the high end, by increasing selected window resources. Most importantly, the effect of increased window size varies dramatically by application. This motivates resource pooling, where we can hope to achieve high overall speedup by allocating window resources where they are most beneficial.

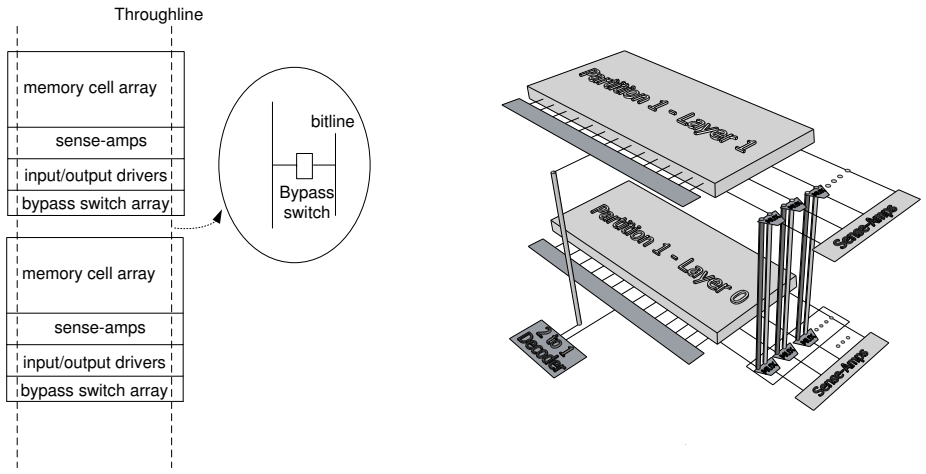
## 3.3 Stackable structures for resource pooling

This section describes the circuit and architectural modifications required to allow resources on vertically adjacent cores to participate in pooling. Specifically, we describe the changes required in each of the pipeline components.

### 3.3.1 Reorder buffer and register file

The reorder buffer (ROB) and the physical register file (RF) are multi-ported structures typically designed as SRAM, with the number of ports scaling with the issue width of the core. Our goal is to share them across multiple cores with minimal impact on access latency, the number of ports, and the overall design. We take advantage of a modular ROB (and register file) design proposed in [PKG06] which is shown to be effective in reducing the power and complexity of a multi-ported 2D SRAM structure. Our baseline multi-ported ROB/RF is implemented as a number of independent partitions. Each partition is a self-standing and independently usable unit, with a precharge unit, sense amps, and input/output drivers. Partitions are combined together to implement a larger ROB/RF, as shown in figure 3.3(a). The connections running across the entries within a partition (such as the bit-lines) are connected to a common through line using bypass switches.

To add a partition to the ROB/RF, the bypass switch for a partition is turned on. Similarly, the partition can be deallocated by turning off the corresponding bypass switch. The modular baseline architecture of our register file




(a) (b)

Figure 3.3: (a) Partitioned ROB and RF design, (b) logical view of two stacked RF(ROB) partitions.

allows individual partitions to participate in resource pooling. To avoid increasing the number of read and write ports of individual partitions of the ROB/RF, we simply assume that an entire partition is always exclusively owned by one core — either the core (layer) it belongs to (host core) or another core (guest core). This significantly simplifies the design, but restricts the granularity of sharing.

Note that before a partition participates in resource pooling (or before it is re-assigned) we need to make sure that all of its entries are empty. This can be facilitated by using an additional bit in each row (entry) of the partition to indicate whether it is full or empty – in most cases, that bit will already exist.

Figure 3.3(b) shows a logical view of two stacked register files, participating in resource pooling (only one partition of the RF from each layer is shown in this figure). The additional multiplexers and decoder shown in figure 3.3(b) are used to route the address and data from/to a partition in one layer from/to another partition in a different layer. The decoder shown in the figure enables stacking of the ROB/RF. To be able to pool up to 4 ROB/RF partitions on four different layers together, we need to use a 4-1 decoder and a 4-1 multiplexer. The register operand tag is also extended with 2 additional bits. The overall delay added to the ROB or RF due to additional multiplexing and decoding is fairly small. For the case of stacking four cores where a 4 input decoder/multiplexer is needed, the

additional delay is found to be below 20 ps (using SPICE simulation and assuming a standard cell 4 input multiplexer). In this design, the access latency of the original register file is only 280ps (using CACTI for an 8 read-port, 4 write-port, 64 entry register file). The additional 20 ps delay due to an additional decoder/multiplexer and the TSVs (5ps at most) still keep the overall delay below one processor cycle. Thus, the frequency is not impacted. For the ROB, the baseline delay is 230 ps and the additional delay can still be tolerated, given our baseline architectural assumptions.

Due to the circular FIFO nature of the ROB, an additional design consideration to implement resource sharing is required, which is not needed for the register file. The ROB can be logically viewed as a circular FIFO with head and tail pointers. The tail pointer points to the beginning of the free entry of the ROB where new dispatch instructions can be allocated. The instructions are committed from the head pointer. Resource sharing requires dynamically adjusting the size of the reorder buffer. To implement such dynamic resizing we use the technique proposed in [PKG06], where two additional pointers are added to the ROB to dynamically adjust its size.

### 3.3.2 Instruction queue and ld/st queue

Both the Instruction Queue (IQ) and the Load/Store Queue (LSQ) are CAM+SRAM structures which hold instructions until they can be issued. The main complexity of the IQ and LSQ stems from the associative search during the wakeup process [PJS97]. Due to large power dissipation and large operation delay, the size of these units does not scale well in a 2D design. The number of instruction queue and LSQ entries has not changed significantly in recent generations of 2D processors.

Figure 3.4(a) shows a conventional implementation of the instruction queue. The taglines run across the queue and every cycle the matchline compares the tagline value broadcast by the functional units with the instruction queue entry (source operand). We assume our baseline IQ utilizes the well-studied divided tagline (bitline) technique [Kar98]. As shown in figure 3.4(b), two or more IQ

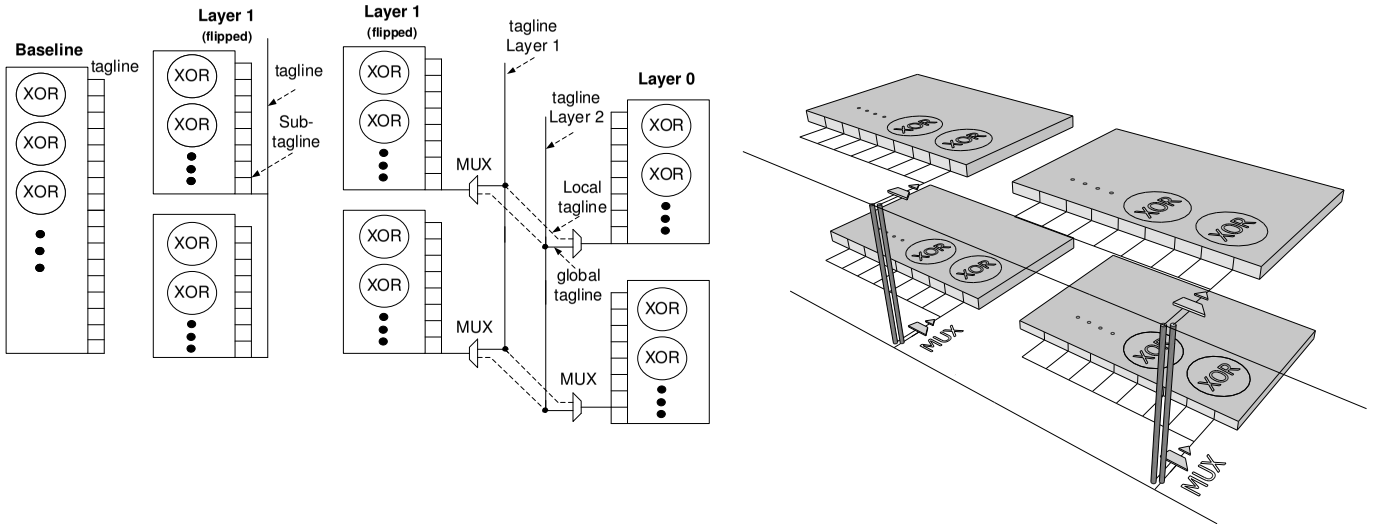



Figure 3.4: (a) Conventional implementation of the IQ, (b) partitioned IQ using divided tagline, (c) implementation of the stacked IQ, (d) logical view of the stacked instruction queue.

entries are combined together to form a partition and to divide the global tag line into several sub-tag lines. This way the IQ is divided into multiple partitions. In the non-divided tag line structure the tag line capacitance is *N * diffusion capacitance of pass transistors + wire capacitance* (usually 10 to 20% of total diffusion capacitance) where N is the total number of rows. In the divided tag line scheme the equivalent tagline capacitance is greatly reduced and is approximated as *M * diffusion capacitance + 2 * wire capacitance*, where M is the number of tagline segments. As tagline dynamic power dissipation is proportional to $CV^2$, reducing the effective capacitance will linearly reduce tagline dynamic power. The overhead of this technique is adding a set of pass transistors per sub-tagline. As a side effect, the large number of segments increases the area and power overhead [Kar98].

To be able to share two or more partitions of the instruction queue, we include one multiplexer per tagline and per IQ partition to select between the local tagline and the global taglines (shown in figure 3.4(c)). Similarly to the RF, to avoid increasing the number of taglines we simply assume that each partition is always allocated exclusively to a single core. This way the number of taglines remains the same and multiplexing, as shown in figure 3.4(c), will route the data on the tagline to the right partition. For the SRAM payload of the instruction queue we simply follow the same modification proposed for our SRAM register file. Bitline segmentation helps to reduce the number of die-to-die vias required for communication between two layers.

We also need to modify the instruction selection logic. Increasing the maximum size of the instruction queue increases the complexity of the selection logic [PJS97]. In a typical superscalar processor each instruction queue entry has a set of bid and grant ports to communicate with the selection logic. Increasing the size of the IQ increases the number of input ports of the selection logic which can negatively impact the clock frequency. To avoid increasing the complexity of the selection logic, we simply allow all partitions participating in resource pooling to share the same selection logic port along with the partition that belongs to the guest core (layer). In this case, we OR the bid signals (from the shared partition and the guest core partition) to the selection logic. The priority is given to the

older entry (age-based priority decoding).

The overall delay overhead in the selection logic is decided by the ORing operation and the age-based priority decoding. Note that the ORing of the bid signals only slightly increases the selection logic delay, by less than 20 ps (using SPICE simulation). This delay does not increase the selection logic access delay beyond a single clock period. For the age-based priority decoding we propose the following to hide its delay: we perform the age-priority computation in parallel with the selection logic (to overlap their delays). When the grant signal comes back, we use the now pre-computed age information to decide where to route the grant.

Under the given assumptions, this analysis indicates we can add the pooling logic without impacting cycle time; however, it is possible that under different assumptions, on different designs, these overheads could be exposed. We will examine the potential impact in the results section.

## 3.4 Adaptive mechanism for resource pooling

In addition to the circuit modifications that are necessary to allow resource aggregation across dies, we also need mechanisms and policies to control the pooling or sharing of resources.

In devising policies to manage the many new shared resources in this architecture, we would like to maximize flexibility; however, design considerations limit the granularity (both in time and space) at which we can partition core resources. Time is actually the easier issue. Because the aggregated structures are quite compact (in total 3D distance), we can reallocate partitions between cores very quickly, within a cycle or cycles. To reduce circuit complexity, we expect to physically repartition on a more coarse-grain boundary (e.g., four or eight entries rather than single entries).

In the results section, we experiment with a variety of size granularities for reallocation of pooled resources. Large partitions both restrict the flexibility of pooling and also tend to lengthen the latency to free resources. We also vary how

aggressively the system is allowed to reallocate resources; specifically, we explore various static settings for the minimum (MIN) and the maximum (MAX) value for the size of a partition, which determine the floor and the ceiling for core resource allocation.

Our baseline allocation strategy exploits two principles. First, we need to be able to allocate resources quickly. Thus, we cannot reassign active partitions, which could take hundreds of cycles or more to clear active state. Instead we actively harvest empty partitions into a free list, from which they can later be assigned quickly. Second, because we can allocate resources quickly, we need not wait to harvest empty partitions — we grab them immediately. This works because even if the same core needs the resource again right away, it can typically get it back in a few cycles.

We assume a central arbitration point for the (free) pooled resources. A thread will request additional partitions when a resource is full. If available (on the list of free partitions), and the thread is not yet at its MAX value, those resources can be allocated upon request. As soon as a partition has been found to be empty it is returned to the free list (unless the size of the resource is at MIN). The architecture could adjust MIN and MAX at intervals depending on the behavior of a thread, but this will be the focus of future work – for now we find static values of MIN and MAX to perform well. If two cores request resources in the same cycle, we use a simple round-robin priority scheme to arbitrate.

## 3.5 Methodology

In order to evaluate different resource adaptation policies, we add support for dynamic adaptation to the SMTSIM simulator [Tul96], configured for multicore simulation. Our power models use a methodology similar to [BTM00]. We capture the energy per access and leakage power dissipation for individual SRAM units using CACTI-5.1 [TMAP08] targeting 45nm technology. The energy and power consumption for each unit is computed by multiplying access counts by the per-access SRAM energy. For temperature calculation we use Hotspot 5.0 [SSH+03].

Table 3.1 gives the characteristics of our baseline core architectures. Note that for each of the register files, 32 registers are assumed to be unavailable for pooling, as they are needed for the storage of architectural registers.

### 3.5.1 Modeling interconnect for resource pooling

We model connections across dies, Tier-to-Tier (T2T), with Through Silicon Vias (TSV). TSVs enable low-latency, high-bandwidth, and very dense vertical interconnect among the pooled blocks across multiple layers of active silicon. We assume four dies are stacked on top of each other. Each tier has an Alpha processor (high-end core case) with die size of $6.4mm \times 6.4mm$ with 12 layers of metal from M1 to M12 and the redistribution layer (RDL). The 3D stacked chip model is flip chip technology and the tiers are connected face-to-back. In the face-to-back connection, the RDL of Tier 1 (T1) is connected to the package via flip chip bumps, and the RDL of Tier 2 (T2) is connected to the M1 of T1 via TSV and forms the T2T connection.

Each core is placed in a single tier of the stack. TSVs connect the Register File (RF), Instruction Queue (IQ), Reorder Buffer (ROB), and Load and Store Queue (LSQ) of each layer vertically. The connection from bottom tier M1 to M12 and RDL layer of the top tier is via TSV, and from M12 and RDL is with resistive via and local routing to the M1 of the sink in RF, IQ, ROB and LSQ.

The resistive through metal via connects metal layers of the tiers, e.g., M1 to M2 in each tier. The vertical and horizontal parasitics of the metals, via, and TSV connections have been extracted to build the interconnect model. A T2T connection includes a through silicon via and a $\mu$bump. The parasitics of the $\mu$bumps are small compared with the TSV [HDC10]. Hence, we only model the parasitics of the TSVs for T2T connections. The length, diameter, and dielectric linear thickness of the TSV which is used for the T2T connection in our model are, respectively, $50\mu m$, $5\mu m$, and $0.12\mu m$. A TSV is modeled as an RLC element with RL in series and C connected to the substrate, i.e., global ground in our model. The parasitic resistance, capacitance, and inductance of the T2T connections are modeled by $R_{TSV}$=$47m\Omega$, $L_{TSV}$=$34pH$, and $C_{TSV}$=$88fF$ [HDC10].

Table 3.1: Architectural specification.

| | Medium-End Core | High-End Core |
|---|---|---|
| Cores | 4 | 4 |
| Issue,Commit width | 2 | 4 |
| INT instruction queue | 16 entries | 32 entries |
| FP instruction queue | 16 entries | 32 entries |
| Reorder Buffer entries | 32 entries | 64 entries |
| INT registers | 48 | 64 |
| FP registers | 48 | 64 |
| Functional units | 2 int/ldst 1 fp | 4 int/ldst 2 fp |
| L1 cache | 16KB, 4-way, 2 cyc | 32KB, 4-way, 2 cyc |
| L2 cache (priv) | 256KB, 4-way, 10 cyc | 512KB, 4-way, 15 cyc |
| L3 cache (shared) | 4MB, 4-way, 20 cyc | 8MB, 8-way, 30 cyc |
| L3 miss penalty | 250 cyc | 250 cyc |
| Frequency | 2GHz | 2GHz |
| Vdd | 1.0V | 1.0V |

Table 3.2: Tier to tier delay via TSV path.

| Tier to Tier Path | Delay (ps) |
|---|---|
| T1 to T2 | 1.26 |
| T1 to T3 | 2.11 |
| T1 to T4 | 2.53 |
| T2 to T3 | 1.31 |
| T2 to T4 | 2.19 |
| T3 to T4 | 1.35 |

The power and signal TSVs connect the power/ground mesh from the package flip chip bumps to each layer. The TSV pitch for the tier to tier connection is assumed to be uniformly distributed with a density of $80/mm^2$ [HDC10]. We assume the TSV structures are via-last where the TSV is on top of the back end of the line (BEOL), i.e., RDL layer and the M1.

In our circuit model we extract the delay path from each SRAM pin (SRAM pin is a signal bump on top of the RDL layer) to the multiplexer of the next SRAM pin. The delay timing for each tier is around 1-2.5 ps as illustrated in Table 3.2 for tier 1 to 4.

The TSV lands on the $\mu$bump and landing pad. Surrounding the TSV and

Table 3.3: TSV area utilization.

| Blocks | pins | | TSV area ($mm^2$) | | TSV block out area ($mm^2$) | |
|---|---|---|---|---|---|---|
| 2 Layer Stack | medium-end | high-end | medium-end | high-end | medium-end | high-end |
| Register File | 876 | 1752 | 0.0688 | 0.1375 | 0.0876 | 0.1752 |
| Ld/St Queue | 792 | 1600 | 0.0622 | 0.1256 | 0.0792 | 0.1600 |
| Inst. Queue | 224 | 464 | 0.0176 | 0.0364 | 0.0224 | 0.0464 |
| Reorder Buff. | 128 | 256 | 0.0100 | 0.0201 | 0.0128 | 0.0256 |
| Total | 2020 | 4072 | 0.1586 | 0.3197 | 0.2020 | 0.4072 |
| 3 Layer Stack | medium-end | high-end | medium-end | high-end | medium-end | high-end |
| Register File | 1314 | 2628 | 0.1031 | 0.2063 | 0.1314 | 0.2628 |
| Ld/St Queue | 1188 | 2400 | 0.0933 | 0.1884 | 0.1188 | 0.2400 |
| Inst. Queue | 336 | 696 | 0.0264 | 0.0546 | 0.0336 | 0.0696 |
| Reorder Buff. | 196 | 384 | 0.0154 | 0.0301 | 0.0196 | 0.0384 |
| Total | 3034 | 6108 | 0.2382 | 0.4795 | 0.3034 | 0.6108 |
| 4 Layer Stack | medium-end | high-end | medium-end | high-end | medium-end | high-end |
| Register File | 1752 | 3504 | 0.1375 | 0.2751 | 0.1752 | 0.3504 |
| Ld/St Queue | 1584 | 3200 | 0.1243 | 0.2512 | 0.1584 | 0.3200 |
| Inst. Queue | 448 | 928 | 0.0352 | 0.0728 | 0.0448 | 0.0928 |
| Reorder Buff. | 265 | 512 | 0.0208 | 0.0402 | 0.0265 | 0.0512 |
| Total | 4049 | 8144 | 0.3178 | 0.6393 | 0.4049 | 0.8144 |

landing pad there is a *keep-out* area where no block, i.e., standard cell is allowed to place and route. We estimate the total TSVs required for connecting memory pins of the RF, IQ, ROB, and LSQ vertically for different stack up numbers in both medium-end and high-end cores. The total area for the TSV landing pad and the block out area is calculated and summarized in Table 3.3. The RF has the largest number of TSVs and the ROB has the fewest.

Power density in the stacked layers increases the thermal profile of the 3D configuration, compared to 2D. In a typical 3D design, TSVs can help with vertical heat transfer among the layers and reduce the thermal hotspots. Hence, additional TSVs which are required for the T2T communication in our architecture will help balance the temperature.

In Table 3.4 we show our Hotspot configuration. In our thermal model we assume a different packaging for medium-end and high-end architecture. For

Table 3.4: Temperature estimation related parameters.

| | High-End Core | Medium-End Core |
|---|---|---|
| Die thickness (um) | 150 | 150 |
| Ambient temperature | 40oC | 40oC |
| Convection capacitance | 140 J/K | 140 J/K |
| Convection resistance | 0.1 K/W | 1.5 K/W |
| Heat sink side | 0.076 m | 0.076 m |
| Heat spreader side | 0.035 m | 0.035 m |
| Interlayer Material Thickness (3D) | 0.02mm | 0.02mm |
| Interlayer Material Resistivity (w/o TSVs) | 0.25 mK/W | 0.25 mK/W |

high-end we assume a modern heat sink that is designed for 160W chips (4-cores). For the medium-end architecture we assume a convection resistance of 1.5 which represents a moderate packaging for 30W chips [MD05].

Note that the vertical delays we report in Table 3.2 (less than 3 ps) compare to the communication cost that would be incurred if resources are shared across 2D cores (which would be about 3 nanoseconds, or 6 cycles) – more than 3 orders of magnitude higher. We do not provide a detailed comparison of our proposed scheme with a 2D dynamically heterogeneous architecture. However, even assuming a generous two-cycle communication latency between all cores in 2D, our results show that just the pipeline changes required to accommodate an equivalent level of sharing would cost each core more than 13% performance.

### 3.5.2 Benchmarks and metrics

We compose multi-program workloads with 2 and 4 threads. The applications are selected from among the SPEC2000 and SPEC2006 benchmark suites, selecting representative sets of memory-intensive and compute-intensive benchmarks. The two- and four-thread groupings are selected alphabetically to avoid bias. Table 3.5 summarizes our workload mixes. For each application in the mix we fast-forward to skip the initialization phase and then simulate until each thread executes 200 million instructions.

Table 3.5: Workload Mix. Spec2006 benchmarks are denoted with "_06"

| Two thread workloads | | | |
|---|---|---|---|
| 2T0 | applu - apsi | 2T7 | libquantum_06 - lucas |
| 2T1 | art - bwaves_06 | 2T8 | mcf_06 - mesa |
| 2T2 | bzip2 - cactusADM_06 | 2T9 | mgrid - milc_06 |
| 2T3 | facerec - galgel | 2T10 | omnetpp_06 - perl_06 |
| 2T4 | gcc_06 - gromacs_06 | 2T11 | povray_06 - sixtrack |
| 2T5 | h264_06 - hmmer_06 | 2T12 | soplex_06 - swim |
| 2T6 | lbm_06 - leslie_06 | 2T13 | vortex - vpr |
| Four thread workloads | | | |
| 4T0 | applu - apsi - art - bwaves | | |
| 4T1 | bzip - cactusADM_06 - facerec - galgel | | |
| 4T2 | gcc_06 - gromacs_06 - h264_06 - hmmer_06 | | |
| 4T3 | lbm_06 - leslie_06 -libquantum_06 - lucas | | |
| 4T4 | mcf_06 - mesa - mgrid - milc_06 | | |
| 4T5 | omnetpp_06 - perl_06 - povray_06 - sixtrack | | |
| 4T6 | soplex_06 - swim_06 - vortex - vpr | | |

We are interested in studying both the performance and fairness effect of our techniques. We report weighted speedup and fairness results. Fairness is defined in [EE08]. Fairness close to 1 indicates a completely fair system where all threads have uniform performance degradation or gain (relative to single-thread performance on a baseline core). Fairness close to 0 indicates that at least one thread starves. We also present weighted speedup [ST00], using the non-pooling core as the baseline.

## 3.6 Results

This section demonstrates the performance and energy advantages of resource pooling. We examine the granularity of resource partitioning, setting of limits on resource usage, and the impact on temperature, power, and energy.

Figure 3.5 shows the weighted speedup and fairness for several configurations of our dynamically heterogeneous processor architecture. All configurations pool resources among four cores, whether the workload is four threads or two. All

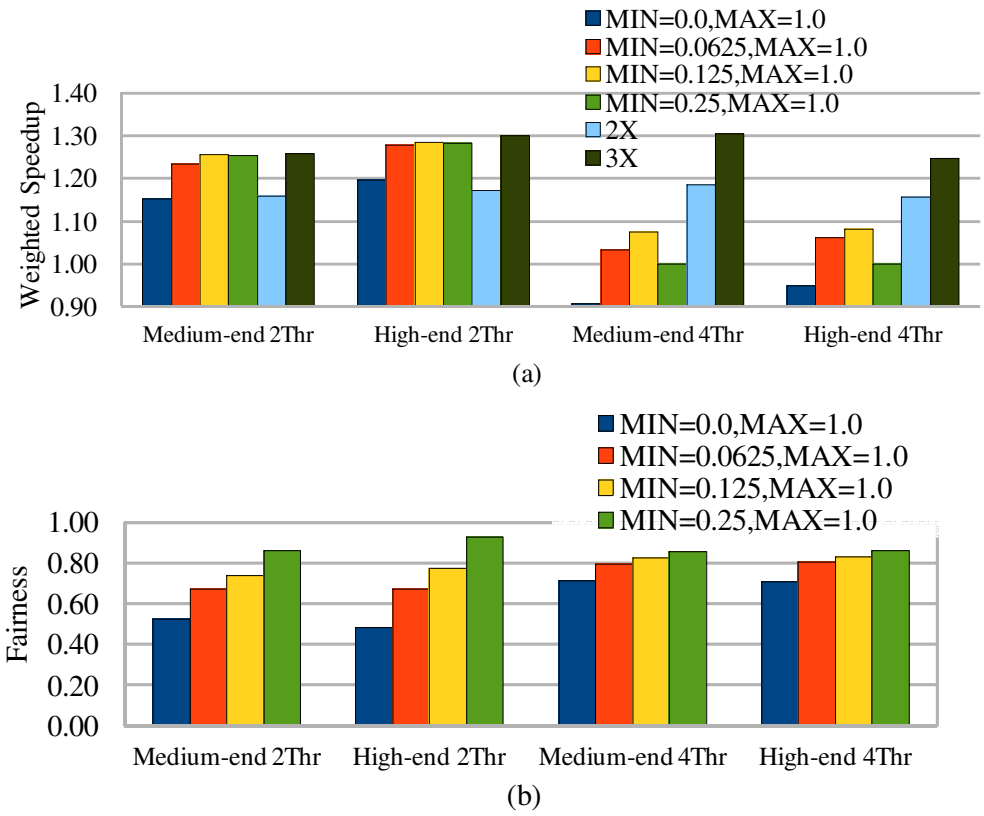



Figure 3.5: (a) Weighted speedup and (b) fairness for dynamically heterogeneous cores, relative to cores with no sharing, for two-thread and four-thread workloads. These results vary MIN and MAX, which determine the floor and the ceiling for core resource allocation.

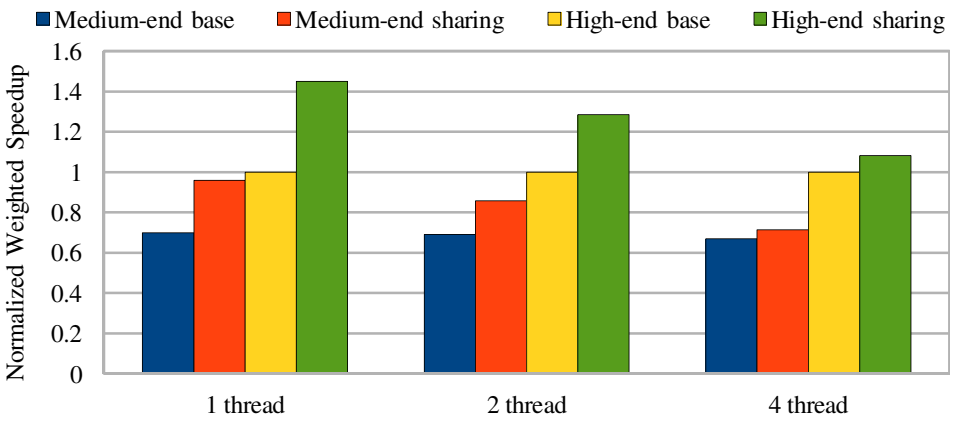



Figure 3.6: Comparison between the medium-end and the high-end core with and without 3D sharing.

allocate resources greedily, within the constraints of the MIN and MAX settings. The different results in this graph represent different values for MIN and MAX, assuming MIN and MAX are constant over time and the same for all cores. For comparison, we also show the performance we get from doubling or tripling all resources.

From this graph we see that while we can get significant performance gains (8-9%) with full utilization (four threads), gains are dramatic when some cores are idle. With two threads we get 26-28% performance for the best policy. In fact, with two threads, performance far exceeds statically doubling the resources, and is equivalent to tripling each core's resources.

Not surprisingly, setting a MIN value to zero, in which case a core can actually give up all resources (for example if it is stalled for an Icache miss) appears to be a bad idea. The best result comes when we reserve one eighth of the total resources (half of a single core's resources) for the core. We see (results not shown) no performance advantage in setting MAX below 1.0. This means that there is no apparent need to restrict one core's ability to grab all available resources if it needs it.

In figure 3.5, the medium-end and high-end performance results are normalized to different baselines, so we cannot directly compare those results. Therefore, we show the results for the two architectures (no sharing and sharing with

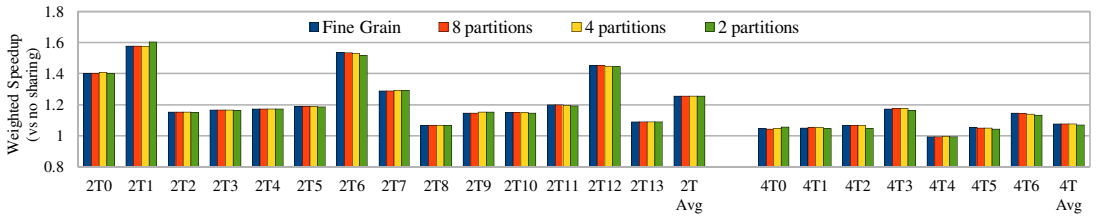



Figure 3.7: Weighted speedup for dynamic heterogeneity, as the granularity of reallocation is changed, for 2-thread and 4-thread workloads (medium-end cores).

MIN=0.125 and MAX=1.0) all normalized to the high-end, no sharing result in figure 3.6. From this graph we can see that resource pooling makes the medium core significantly more competitive with the high-end. Without sharing, the medium core operates at 67% of the performance of the high end. With pooling and four active threads it operates at 71%, with two active threads, it operates at 86%, and with one active thread, it operates at 97% of the performance of the high-end core.

Finally, we performed a sensitivity analysis to study the impact of clock frequency scaling on the performance benefit of resource pooling. If the worst-case logic overhead of 25 ps were fully exposed (given our assumptions, it should not be exposed at all), increasing the cycle time by that same amount (5%), this architecture still gains 4% running 4 threads, 20% running 2 threads, and 33% running one thread, relative to an architecture with no sharing for the medium-end core. For the high-end core, the respective gains are 6.5% running 4 threads, 25% running 2 threads, and 42% running one thread.

### 3.6.1 Fine vs. coarse partitioning

We also study the performance across different partitioning granularities for the best allocation technique. Larger (coarser) granularity of reallocation simplifies the circuits and the reallocation manager. In figure 3.7 we report weighted speedup for dynamic heterogeneity, as the granularity of reallocation is changed (fine grain is one entry per allocation).

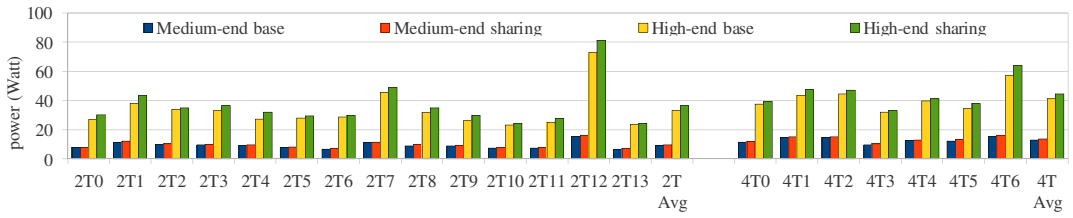



Figure 3.8: Power consumption per core for MIN=0.125, MAX=1.0 as well as the baseline (no sharing) for 2-thread workloads and 4-thread workloads.

We observe that performance is very tolerant of large partition sizes – we apparently gain little from increased flexibility. The reason is that most of the resource allocations and deallocations occur in bursts. Once a thread misses in the data cache, it will keep requesting resources until it either consumes the whole pool or reaches its MAX limit. Once this happens, the thread will retain the resources for the duration of the miss. Large partitions actually make it easier to meet a thread's sudden demand quickly. We use 4 partitions (per core) for the remaining experiments described in this chapter.

### 3.6.2 Power, temperature, and energy

Figure 3.8 shows the power consumption of the various architectures. The pooling processor architectures pay a small price in power, in large part because of the enhanced throughput.

The small additional power overhead is in contrast in some cases to the large performance benefit (in terms of weighted speed up). This is due in part to a subtlety of our experiments. Weighted speedup weights application speedups equally (rather than over-weighting high-IPC threads, which throughput measures do). Because we get some large speedups on low-IPC threads, we see high average speedup, but smaller increase in total instruction throughput and thus smaller increase in power.

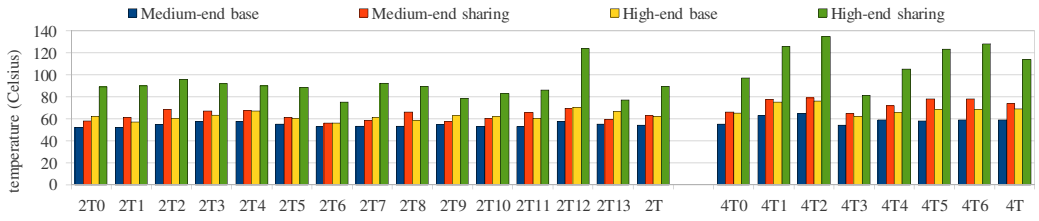



Figure 3.9: Max temperature for MIN=0.125,MAX=1.0 and baseline for 2-thread workloads and 4-thread workloads.

Because of the layout advantages (remember, the baseline processor places all cores right next to the heat sink), the cost in maximum temperature is more significant (figure 3.9). Interestingly, the temperature of the medium resource-pooling core is comparable to the high-end core. This is in part because we assume the medium core is laid out tightly, resulting in a slightly higher max temperature for four-thread workloads. For two-thread workloads, the medium resource-pooling core has slightly lower temperature than the high-end core (average 2 degree lower). If the medium core covered the same area as the high-end core, for example, the max temperature would be significantly lower. Even still, at equal temperature, the more modest cores have a significant advantage in energy efficiency measured in $MIPS^2/W$ ($MIPS^2/W$ is the inverse of energy-delay product), as seen in figure 3.10. This is a critical result. By outperforming the non-pooling medium core, and approaching the performance in some cases of the large core (due to its just-in-time provisioning of resources), the dynamically heterogeneous medium-end core provides the highest energy efficiency.

## 3.7 Related work

Prior research has attempted to provide dynamically heterogeneous computing in the 2D domain. Core Fusion [IKKM07] and TFlex [KSG+07] aggregate resources at the granularity of an entire core, creating large cores out of smaller

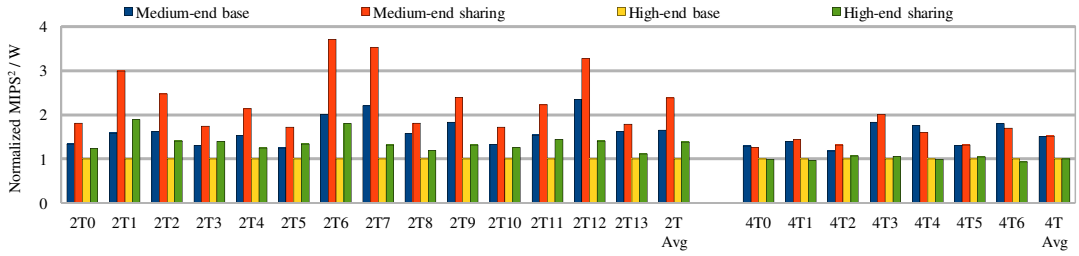



Figure 3.10: MIPS$^2$ per Watt for the 2-thread and the 4-thread workloads normalized to the high-end configuration without sharing.

cores when ILP is high and/or thread level parallelism is low. The dynamically heterogeneous architecture described in this chapter shares resources at a much finer granularity. Other research provides the ability to share execution resources between cores. The Conjoined Core architecture [KJT04] shares over-provisioned resources between cores, but is limited in what resources it can share and has limited ability to adapt to changing resource needs. The StageNet architecture [GFA+08] can access stages from other core pipelines, allowing it to reconfigure around faults.

There is also a large body of prior work in 3D stacked architectures, including several others that exploit logic-on-logic designs. Some previous research focuses on partitioning the pipelined architecture and split the function unit blocks across different layers [VHW+07, PL07]. Other researchers maintain the functional blocks in each layer and take advantage of floorplan and physical design to gain performance benefits from stacking [HVE+07]. A re-partitioned design of the Intel 3D Pentium 4 was designed by Black, et al. [BNWS04] with 15% performance improvement but 15% increased power. They demonstrate that by re-designing and splitting the IA32 processor, thermal hotspots can be reduced without sacrificing timing.

Both design and process technology are evolving to address the 3D thermal and reliability challenges. For example, Coskun, et al. [CAR+10] examine new inter layer cooling schemes. Commercial products and some recent test chips already leverage 3D technology in several domains, such as image sensors [YKT+09]

and stacked memories [Tez, HAG+10], including some examples that map logic on logic [WSLL10, Tho10]. Likely the first steps in 3D stacked chips will involve heterogeneous technologies, as these designs pose significantly smaller challenges in terms of thermals and require minimal modifications to existing designs. To the best of the author's knowledge, this work is the first to propose stacking homogeneous chips in order to dynamically share back-end core resources with minimal pipeline changes and boost single-thread performance.

## 3.8 Conclusion

This chapter describes a dynamically heterogeneous 3D stacked architecture which enables very fine-grain reallocation of resources between cores on a stacked chip multiprocessor architecture. This architecture enables fine-grain resource sharing not possible in a conventional 2D architecture. It can do so because we can leverage our current expertise in creating tight 2D pipelines on one layer, while accessing pooled resources of the same type on other layers.

This research examines the sharing of instruction scheduling window resources, in particular, describing circuit-level techniques to enable fast reallocation of resources between cores. We find that a processor with poolable resources shared among four cores can outperform a conventional multiprocessor by 41% when one thread is running, 23% when two threads are running, and 9% when four threads are running.

By eliminating the need to over-provision each core, modest cores become more competitive with high-performance cores, enabling an architecture that gives up little in performance, yet provides strong gains in energy-delay product over a conventional high-performance CMP architecture.

## Acknowledgments

Chapter 3 contains material from Dynamically heterogeneous cores through 3D resource pooling, by Houman Homayoun, Vasileios Kontorinis, Amirali Shayan,

Ta-Wei Lin, and Dean Tullsen, which appears in *Proceedings of the The 18th International Symposium on High Performance Computer Architecture (HPCA)*. The dissertation author contributed equally with the main author of the work, Houman Homayoun. The material in this chapter is copyright ©2012 IEEE. Personal use of this material is permitted. However, permission to reprint/republish this material for advertising or promotional purposes or for creating new collective works for resale or redistribution to servers or lists, or to reuse any copyrighted component of this work in other works must be obtained from the IEEE.

# Chapter 4

# Managing peak power for data centers

We have previously discussed how to remove over-provisioning and the associated costs at the core level by disabling underutilized resources, at the chip level by using modest cores in 3D stacked dies and pooling their resources to boost performance. In this chapter, we will discuss again how to remove over-provisioning and lower costs at an even higher level, the data center.

The costs of building and running a data center, and the capacity to which we can populate it, are driven in large part by the peak power available to that data center. This work demonstrates techniques to significantly reduce the observed peak power demand for data centers with distributed UPS batteries, enabling significant increases in data center capacity and reductions in cost.

Modern data center investments consist of one-time infrastructure costs that are amortized over the lifetime of the data center (capital expenses, or capex) and monthly recurring operating expenses (opex) [HB09]. Capex costs are proportional to the provisioned IT power per facility, estimated at $10-20 per Watt [Dup07, pr07, TB09], as each Watt of computing power requires associated support equipment (cooling, backup, monitoring, etc.). Utilities typically charge a power premium that is tied to the peak power. This can become a significant portion of the monthly bill, up to 40% [GSU11]. This chapter examines the use of distributed batteries in the data center to reduce both capex and opex costs.

Power infrastructure is commonly over-provisioned in data centers to accommodate peaks and to allow for future expansion. However, to improve common case utilization, we can intentionally over-subscribe (under-provision) the power infrastructure [FWB07, HB09, KZL+10, LKL11, PMZ+10]. Over-subscribing provisions power infrastructure to support a lower demand than the largest potential peak and employs techniques to prevent power budget violations. In the worst case, such violations could trip circuit-breakers and disable whole sections of the data center, causing costly down time. To avoid this, data centers can employ power capping approaches such as CPU capping, virtual CPU management, and dynamic voltage and frequency scaling (DVFS) [LKL11, NS07, RRT+08]. CPU capping limits the time an application is scheduled on the CPU. Virtual CPU management limits virtual machine power by changing the number of virtual CPUs. DVFS attacks the peak power problem by reducing chip voltage and frequency. However, all of these techniques result in performance degradation. This is a problem for any workload that has performance constraints or service-level agreements because power management policies apply these performance-reducing mechanisms at the exact time that performance is critical – at peak load.

Govindan, et al. [GSU11] introduce a new approach that has no performance overhead in the common case. They leverage the energy stored in a centralized data center UPS to provide energy during peak demand, effectively hiding the extra power from the power grid. This technique is shown to work well with brief (1-2 hours), high-magnitude power spikes that can be completely "shaved" with the energy stored in batteries; however, it is less effective for long (8-10 hour) spikes. For longer spikes, they suggest a hybrid battery-DVFS approach.

However, many large data centers do not employ centralized batteries. Distributed, per-server batteries represent a more economical solution for battery backup. They scale naturally with the data center size and eliminate a potential single point of failure. Google employs this topology in their state-of-the-art data centers [Goo09].

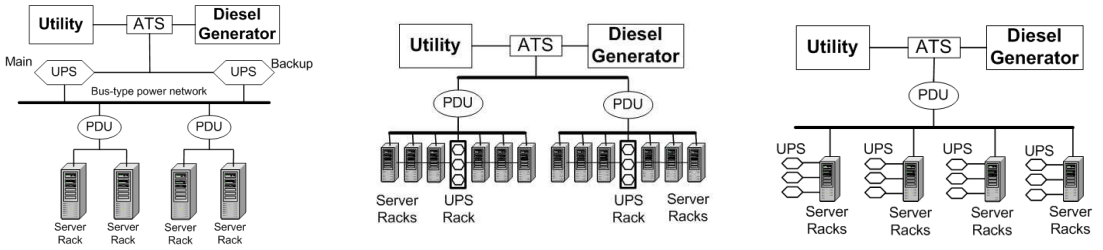
When leveraging a distributed UPS architecture to shave peak power, challenges arise due to the lack of heavy over-provisioning and the distributed nature

of the batteries. The absence of over-provisioned UPSs means we need to justify the use of larger batteries based purely on cost savings from power capping. We need policies to determine how many batteries to enable, which batteries to enable, and when. However, there are also opportunities compared to prior solutions. In a centralized UPS architecture, all power typically comes from either the battery or the utility. Thus, when batteries are enabled, they supply all datacenter power and drain quickly – if we only supply the over-threshold power, the batteries can sustain longer peaks. This is easily done in the distributed architecture by simply enabling enough batteries to hide the desired peak.

In this work, we discuss the applicability of battery-enabled power capping to distributed UPS topologies. We present details on the sizing and the technology alternatives of per-server batteries and consider several approaches that orchestrate battery charging and discharging while addressing reliability and availability concerns. This research goes beyond prior work by modeling realistic data center workload patterns over a multi-day period and by arguing that battery over-provisioning is financially beneficial. Enabling the placement of additional servers under a given power budget permits reductions of the data center total cost of ownership per server on the order of 6%. This is equivalent to more than $15M for a datacenter with 28,000 servers.

The work in this chapter makes the following unique contributions. (1) It is the first work to describe a solution for peak power capping which aggressively utilizes distributed UPS batteries. (2) It provides a full financial analysis of the benefits of battery-based power capping, including optimal battery technology and battery size. (3) It is the first work on battery-based power capping which models realistic workloads and demonstrates implementable policies for battery management.

This chapter is organized as follows. Section 4.1 presents common UPS topologies and the associated trade-offs. Section 4.2 describes our total cost of ownership analysis. In section 4.4 we contrast alternative battery technologies for frequent battery charge/discharge in the data center context and elaborate on their properties. In section 4.5, we present our policies. In section 4.6, we discuss our




(a) Centralized (b) Distributed (Facebook) (c) Distributed (Google)

Figure 4.1: Power hierarchy topologies with (a) centralized UPS and (b,c) distributed UPS solutions.

experimental methodology and give experimental results in section 4.7. Section 4.8 reviews related work in power capping techniques, and section 4.9 concludes.

## 4.1 Background

Primary power delivery in data centers is through a utility line. Data centers are also equipped with a diesel generator unit which acts as a secondary source of power during a utility failure. To facilitate switching power between the utility and the diesel generator, an automatic transfer switch (ATS) selects the source of power, which takes 10-20 seconds [GSU11]. During this short and critical interval, the UPS units supply the power to the data center. In the centralized topology shown in figure 4.1(a), the power from a single UPS is fed to several Power Distribution Units (PDUs) to route the power to racks and servers. To eliminate the transfer time of the power line to the UPS, data centers commonly use double conversion UPSs. With double conversion UPSs, power is transformed from AC-to-DC to be stored in batteries and then from DC-to-AC to be used by the racks and servers. Although this organization has zero transfer time to the UPS (the UPS is always in the power path), the availability of the whole data center is dependent on the UPS. Additionally, double conversion introduces 4-10% power losses during normal operation [Goo09].

The centralized UPS topology in figure 4.1(a) does not scale well for large

data centers. This topology either requires double conversion, so that the power network distributes AC power, to be converted again to DC, or it distributes DC over the large network, resulting in higher cable losses. The inefficiency of AC-DC-AC conversions becomes more costly at scale. The UPS is also a single point of failure and must be overprovisioned.

Figure 4.1(b) shows the distributed design adopted by Facebook. A cabinet of batteries for every 6 racks, or a total of 180 servers, replaces the centralized UPS [Fac11]. This design avoids double conversion by customizing the server power supply unit to support both AC power (from the grid) and DC power (from the battery cabinet). DC power is distributed from the UPS to the servers, but in this case that is a much shorter distance. Google goes even further, attaching a battery on every server after the Power Supply Unit (PSU) [Goo09], as depicted in figure 4.1(c). This design also avoids the AC-DC-AC double conversion, saving energy under normal operation, and brings the AC distribution even closer to the IT load, before it is converted.

Availability in data centers is a function of how often failures happen, the size of the failure domain, and the recovery time after each failure. UPS placement topology impacts the availability of the data center, particularly the associated failure domain. The more distributed the UPS solution, the smaller the failure domain. Thus, the centralized design requires full redundancy, while the Google approach provides none (loss of a single node is insignificant), further reducing cost.

## 4.2 Total cost of ownership analysis

Modern data centers are typically power limited [TB09]. This means that the overall capacity (number of servers) is limited by the initial provisioning of the power supporting equipment, such as utility substations, diesel generators, PDUs, and cooling. If we reduce the peak computing power, we can add additional servers while remaining within the same power budget, effectively amortizing the initial investment costs over a larger number of servers. Moreover, extra work done per

$$
\begin{aligned}
TCO/server = (&dataCenterDepreciation + dataCenterOpex + \\
&serverDepreciation + serverOpex\ )\ /\ N_{servers} \\
= (\ (&FacilitySpaceDepr + \\
&UPSDepr + \\
&PowerInfrastructureDepr + \\
&CoolingDepreciation + \\
&RestDepr\ ) + dataCenterOpex + \\
&serverDepr + \\
&(ServerRepairOpex + \\
&(ServerEnergyOpex + ServerPowerOpex) * PUE\ )\ )/N_{servers}
\end{aligned}
\tag{4.1}
$$

data center should result in fewer data centers, greatly reducing infrastructure capital expenses.

Distributed UPSs are currently designed to support the whole computing load long enough to ensure safe transition from the main grid to the diesel generator. This time window (less than one minute) translates to batteries with insufficient stored energy for meaningful peak power shaving. Therefore, to enable peak power capping using UPS stored energy in the distributed context, we need to over-provision per server battery capacity. This section discusses the TCO model we use to examine the battery over-provisioning that makes financial sense and maximizes total profits.

The profitability of an investment is defined as the generated revenue minus the associated total cost of ownership (TCO). The data center revenue equals the number of servers times the income per server. We assume constant income per server. Therefore, maximizing the profitability per server is equivalent to minimizing the TCO per server. We now explain how placing more servers within the same power budget reduces TCO per server. Our TCO analysis is inspired

by the data center cost chapter in Barroso and Hölzle [HB09]. For simplicity, we assume sufficient initial capital, hence there are no monthly loan payments, and full capacity for the data center (limited by the provisioned power) from the first day. The TCO/server is given by equation 4.1.

In this equation, data center depreciation is the monthly depreciated cost of building a data center (we assume 10 year straight-line depreciation [HB09]) The assets required for a data center are land, UPS and power infrastructure (diesel generators, PDUs, back-room switchgear, electrical substation), cooling infrastructure (CRAC, economizers), as well as several other components such as engineering, installation labor, racks, and system monitors that we include in *RestDepreciation*. The data center opex is the monthly cost for running the data center (infrastructure service, lighting). We collect the depreciation and opex cost information for a data center with 10MW provisioned computing power (critical power) from APC's commercial TCO calculator [APC08].

Servers typically have shorter lifetimes and are depreciated over 4 years. Server opex consists of server repairs and the electricity bill. Utility charges have a power component and an energy component. The power component is based on the peak sustained power for a 15 minute window over the period of a month [Duk09] while the energy is based on the total data center energy used (different charging models provide similar results). To account for the electricity consumed by infrastructure, excluding servers, we scale the total server peak power and energy by the power usage effectiveness (PUE), assumed at 1.15 [Goo09]. We assume a customized commodity server similar to Sun Fire X4270, with 8 cores (Intel Xeon 5570) at 2.40 GHz, 8 GB of memory, and costing $1500. The inputs to our TCO model are summarized in table 4.1.

We show the breakdown of TCO/month/server in the table and the pie chart of Figure 4.2. The major TCO component is server depreciation (40.6%). Infrastructure related components (facility space, power, cooling, and data center opex) account for more than 35%. In the same table, we also present how the ratio of each TCO component per server changes when we are able to add additional servers within the same power budget. Server depreciation, server opex, and UPS

| TCO component | TCO/month (TCO/month/srv) | TCO/server trend with extra servers |
|---|---|---|
| Facility Space depreciation | 96,875$ (3.46$) | Decreasing |
| UPS depreciation | 3,733$ (0.13$) | Constant |
| Power Infrastructure depreciation | 169,250$ (6.04$) | Decreasing |
| Cooling infrastructure depreciation | 70,000$ (2.50$) | Decreasing |
| Rest depreciation (racks, monitoring,engineering,installation) | 255,594$ (9.13$) | Decreasing |
| Data center opex (maintenance, lighting) | 213,514$ (7.63$) | Decreasing |
| Server depreciation | 875,000$(31.25$) | Constant |
| Server opex (Service/repairs) | 43,750$ (1.56$) | Constant |
| PUE overhead | 55,467$ (1.98$) | Constant |
| Utility monthly energy cost | 252,179$ (9.01$) | Constant |
| Utility monthly power cost | 117,600$ (4.20$) | Decreasing |
| Total | 2,152,961$(76.89$) | Decreasing |

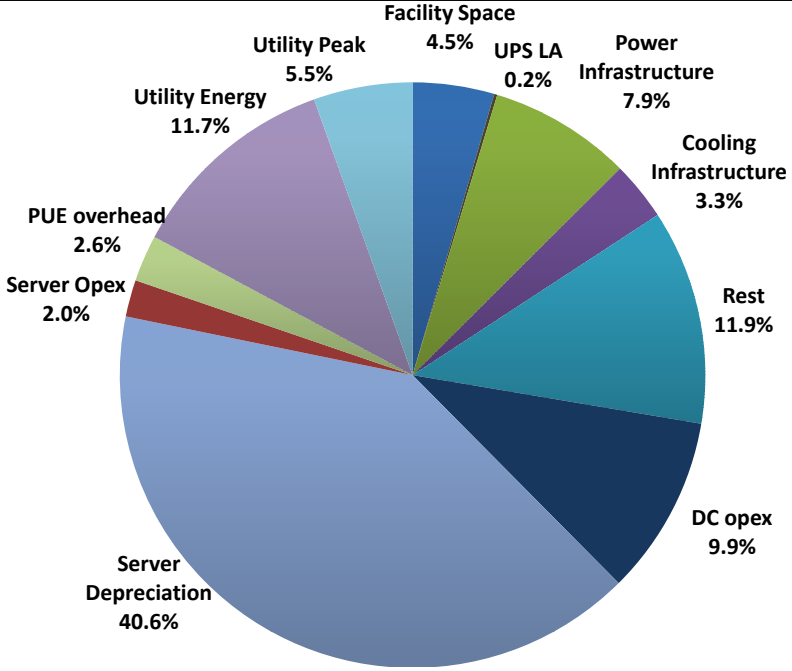



Figure 4.2: Total Cost of Ownership (TCO) [APC08]. TCO/server decreases as we increase servers under same power budget

Table 4.1: TCO model assumptions

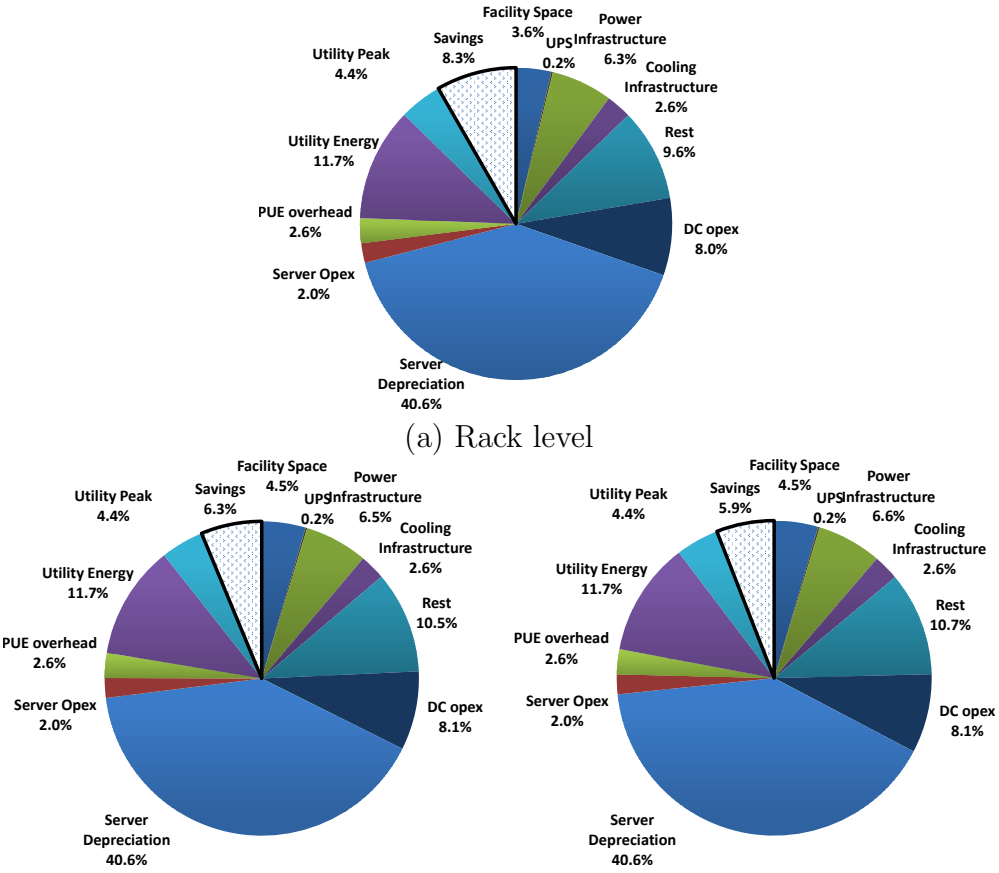
| | |
|---|---|
| Data center Critical Power | 10 MW |
| Server | Idle Power: 175W, Peak Power: 350W (measured) |
| Number of servers | 28000 (critical power / server peak) |
| Average Server Utilization | 50% [HB09] |
| Utility Prices | Energy: 4.7c/KWh,<br>Power: 12$/KW [Duk09, GSU11] |
| Server cost | $1500 |
| PUE | 1.15 [Goo09] |
| Amortization Time | Infrastructure: 10 years, Servers: 4 years [HB09] |

TCO scale with the number of servers and are constant. The energy component of the utility bill also scales with the number of servers, but the power component stays the same and is amortized over more servers. Infrastructure costs are also amortized over a larger number of servers. The UPS cost (estimated as the total cost of the server-attached batteries) represents a very small portion of the TCO; it is marginally visible in the pie chart. Our proposal over-provisions batteries and increases the cost of the distributed UPS. In return, we amortize the cost of several large components over a larger set of servers. The full TCO model described here can be found in [Kon12].

## 4.3 The benefits of power oversubscription

In this section we evaluate the benefit of power oversubscription ignoring the cost of the power management solution that prevents power budget violations. This extra cost can be expressed in performance loss and hence income reduction for example when DVFS or consolidation is applied. Or additional cost for hardware modifications that ensure zero performance degradation, such as battery enabled power capping.

The three pie charts in figure 4.3 show the benefits of oversubscribing the supporting equipment at different levels of the power hierarchy. Power oversubscription at higher levels incurs extra costs. Oversubscription at the rack level,




(a) Rack level

(b) PDU level

(c) Cluster level

Figure 4.3: The power oversubscription benefits at different levels of the power hierarchy. Oversubscribing at lower levels results in more savings.

assuming there is sufficient rack space, comes effectively for free. At the PDU level we account for the extra rack cost as well as the additional facility space that the extra racks occupy. Finally at the cluster level on top of the rack and facility cost we need to account for the additional PDU cost to accommodate the newly added racks. As a result, the benefits in terms of TCO per server decrease as we oversubscribe higher in the power hierarchy. Note here that oversubscribing at lower levels is more beneficial, however at higher levels of the power hierarchy averaging effects of server power result in larger margins for power reduction and oversubscription, at no performance cost.

At this point we should stress that these benefits will vary significantly according to the underlying assumptions for power oversubscription as well as the component costs. To highlight this effect we perform a sensitivity analysis on the cost and the peak power of the server in figure 4.4. Power oversubscription becomes more effective with high power, low cost servers and less effective with low power, high cost servers. With smaller peak power servers we can pack more servers under the same supporting equipment and amortize capital expenses better. This means that the supporting equipment constitutes a smaller portion of the TCO per server to begin with and therefore power oversubscription is less helpful. Similarly, higher server cost for server with same peak power, translates to higher cost for the same number of server. This is equivalent to smaller portion of the TCO part for the supporting equipment and less effective power oversubscription. The best scenario in terms of power oversubscription savings is low cost, high power servers. For this scenario, we reduce the total cost of ownership per server by almost 10%. This reduction is achieved with 24% more servers powered by the same infrastructure. Such degree of oversubscription is possible with the UPS batteries that we describe in the next section.

## 4.4 Characterizing distributed UPS batteries

Current UPS designs rely on lead-acid batteries because of their ability to provide large currents for high power applications at low cost. In this section,

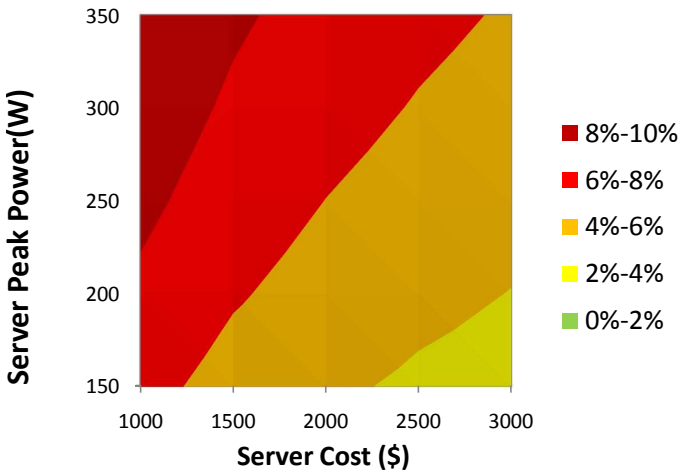



Figure 4.4: TCO per server reduction due of oversubscribing power infrastructure by 24%, as the peak power of the server and the cost of the server change.

we discuss alternative battery technologies for distributed UPSs, model battery behavior when employed for peak power capping, and elaborate on the selection of parameters (capacity, cost, depth of discharge) to minimize TCO/server.

The spider graph in figure 4.5 compares the major competing battery technologies for high power applications, typical for servers, at the range of 12V and 15-30A: lead-acid (LA), Lithium Cobalt Oxide (LCO), and Lithium Iron Phosphate (LFP). Other technologies like NiCd, NiMH, or other lithium derivatives are excluded because they are dominated by one of the discussed technologies across all metrics. LA never performs best along any dimension except at low temperatures. While LA is cheapest per Wh, LFP offers an order of magnitude more recharge cycles, hence provides better $/Wh/cycle than LA. LCO is the most expensive technology and provides comparable recharge cycles to LA. The advantage of LCO technology is its high volumetric density (Wh/l) and gravimetric density (Wh/Kg). Lithium batteries have longer service life than LA and also recharge faster. LFP has higher margins for over-charging and is safer than LA (may release toxic gases when over-charged) and LCO (may catch fire).

Properly selecting the technology and battery size depends on its use. UPS batteries in modern data centers are discharged only during a power outage. Ac-

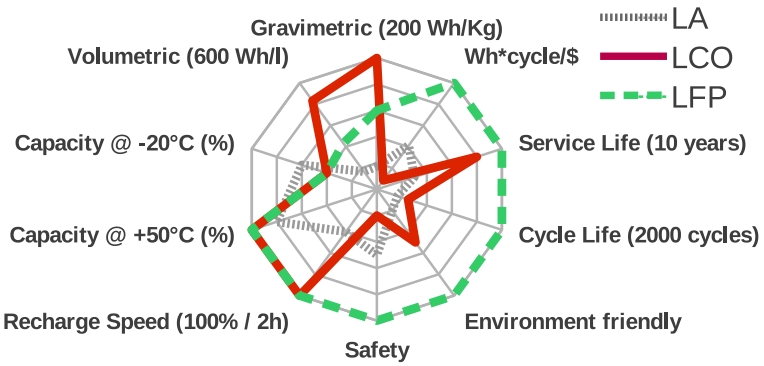



Figure 4.5: Comparison of battery technologies [AA 12, Uni03]

cording to [Pon10], the number of utility outages that affect data centers ranges from 1.5 to 4.4 per year. Therefore, cost, service life, and size are the most important parameters. The selection criteria become quite different when we re-purpose the batteries to be aggressively charged and discharged. Recharging cycles become crucial because continuous battery use may drastically shorten battery lifetime, resulting in frequent replacement costs that negatively affect TCO/server. Hence $/Wh/cycle is a better metric than $/Wh alone. Since LCO does poorly on both cost and cycles, it is not considered further.

We now focus on the per server distributed UPS design and explore the degree of over-provisioning that is most financially beneficial. Battery cost is estimated based on its 20h-rated capacity in Amp-hours (Ah) and the cost per Ah. We derive the required battery capacity based on the amount of power we want to shave and the corresponding energy stored in a battery for a given daily power profile. We derive the cost per Ah from [AA 12, Ele01]. Tables 4.2 and 4.3 show all the inputs for the battery sizing estimation.

To derive the required battery capacity, we first set a peak power reduction goal and estimate the total energy that needs to be shaved at the data center level over the period of a day. We assume all batteries get charged and discharged once per day because, according to [Goo12], all the traffic profiles of large distributed applications demonstrate a single peak. The daily shaved energy is equivalent to

Table 4.2: Input values for battery cost estimation.

| Input | Value | | Reference |
|---|---|---|---|
| | LA | LFP | |
| Service time | 4yrs | 10yrs | [Win09, Ele01] |
| Battery Cost per Ah | 2$/Ah | 5$/Ah | [Ele01, AA 12] |
| Depth of Discharge | 40% | 60% | Estimated (see figure 4.8) |
| Peukert's exponent | 1.15 | 1.05 | [Har09] |
| Existing Server Battery Capacity | 3.2Ah | | [Goo09] |
| Recharge Cycles | f(DoD) – Table 4.3 | | [STR08, Win09] |
| Battery Voltage | 12V | | [Goo09] |
| Max Battery Discharge Current | 23A | | Estimated (ServerPeak * PSUeff / Voltage) |
| PSUeff | 0.8 | | [Cli11] |
| Discharges per day | 1 | | Based on data from [Goo12] |
| Battery losses | 5% | | [RL11, ZLI+11] |

Table 4.3: Recharge cycles as a function of depth of discharge (DoD). Deep battery discharge results in a fewer recharge cycles[STR08, Win09].

| DoD (%) | 10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | 90 | 100 |
|---|---|---|---|---|---|---|---|---|---|---|
| Rcycles LA | 5000 | 2800 | 1860 | 1300 | 1000 | 830 | 650 | 500 | 410 | 330 |
| Rcycles LFP | 100000 | 40000 | 10000 | 6000 | 4000 | 3000 | 2000 | 1700 | 1200 | 1000 |

the integral between the power curve and the flat power line we set as the peak goal. For simplicity in this section we consider a power peak as a diurnal square pulse with a specified height and duration. For that workload, the required data center discharge energy is given by equation 4.2.

$$E_{DataCenter} = DataCenterPeakPower \times PowerReduction \times PeakTimePerDay \tag{4.2}$$

To simplify the analysis, we assume that all servers discharge their batteries at the same rate. We relax this assumption (and several other assumptions applied to this initial analysis) in section 4.7. Equation 4.3 estimates the energy

that each battery should provide to the associated server. Since the distributed battery is attached after the power supply the power drawn from the battery goes directly to the motherboard and is not wasted on losses of the Power Supply Unit (PSUefficiency).

$$E_{server} = \frac{E_{DataCenter} \times PSUefficiency}{N_{servers}} \tag{4.3}$$

Given the energy each battery must provide, we estimate the energy stored per battery and the corresponding battery capacity using Peukert's law. This relation is given by equation 4.4, where $C_{1h}$ is the battery capacity in Ah (1h means that the battery capacity, equivalent to charge, is measured drawing constant current for 1h), I is the discharge current, PE is Peukert's exponent, and T is the battery discharge time [Sma11, RL11]. Lead-acid batteries typically have a Peukert's exponent in the range of 1.05-1.25 while Lithium Iron Phosphate batteries are in the range of 1.03-1.07 [Har09].

$$T = \frac{C_{1h}}{I^{PE}} \Rightarrow C_{1h} = T \times I^{PE} = \frac{E_{server}}{V \times I} \times I^{PE} = \frac{E_{server}}{V} \times I^{PE-1} \tag{4.4}$$

We also account for battery depth of discharge (DoD), the degree to which we allow the battery to be drained. Fully discharging the battery (100% DoD) to extract the required amount of energy would seriously degrade the lifetime of the battery and translate to higher battery replacement costs (see table 4.3). Limiting DoD also allows us to use excess capacity for power capping without increasing exposure to power failures. Consequently, we only want to discharge the battery partially. However, the less we discharge a battery, the larger battery capacity we need in order to discharge the same amount of energy. For discharge current, we conservatively assume the max value of the server current ($I_{MAX} = 23A$). Additionally, batteries lose a portion of their capacity as they age. Once they reach 80% of their original capacity, battery manufacturers consider them dead. We pessimistically take this effect into account by scaling the capacity by a factor of 1/0.8. Using equation 4.4, we get the provisioned 1h-rated battery capacity for each server battery (equation 4.5).

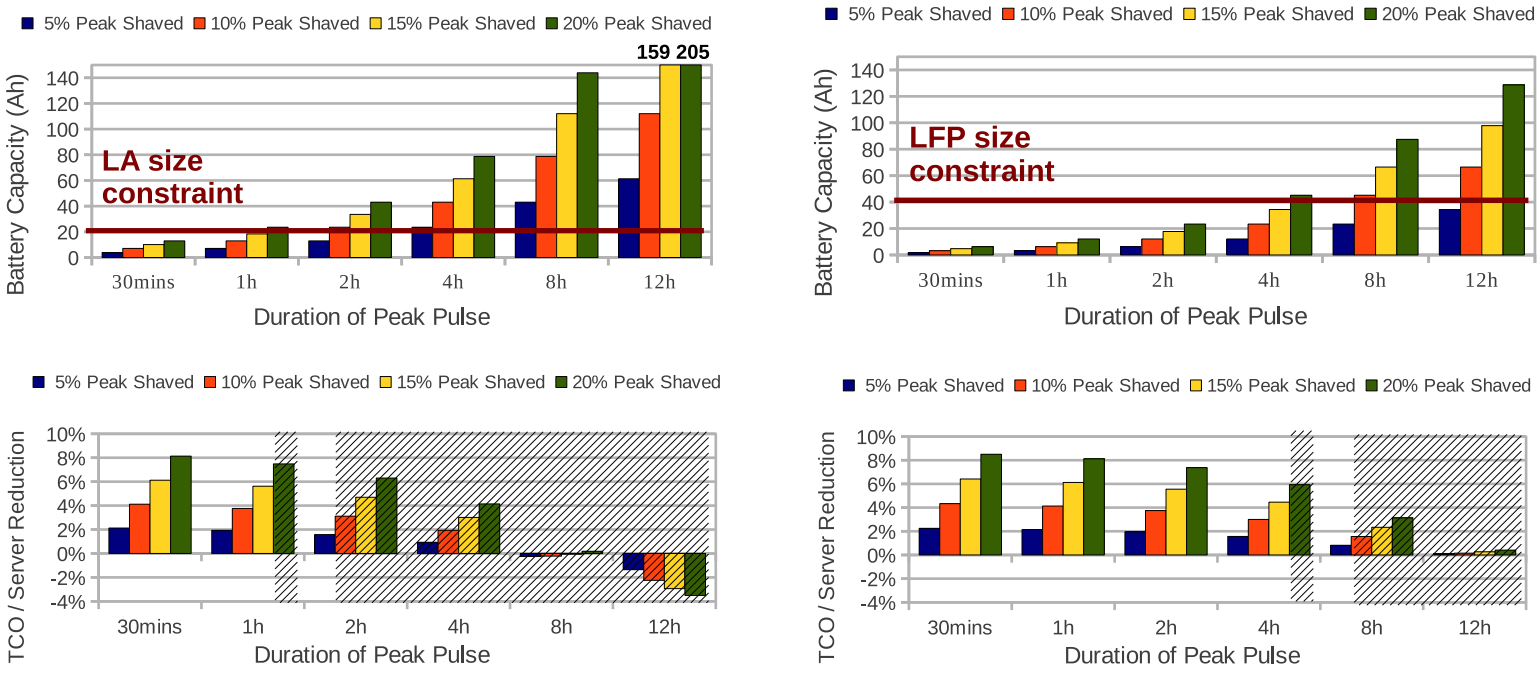
$$C_{1h}\ prov. = C_{1h} \times \frac{1}{DoD} \times \frac{1}{0.8} = \frac{E_{server}}{V} \times I_{discharge}^{PE-1} \times \frac{1}{DoD} \times \frac{1}{0.8} \quad (4.5)$$

Finally, we convert the 1h-rated capacity into 20h-rated capacity [RL11, Sma11], the value reported by battery manufacturers.

The previous capacity estimation methodology allows us to translate a peak power reduction goal to per-server provisioned battery capacity and the associated cost. To compute the monthly UPS depreciation, we also need to know the average battery lifetime. The battery lifetime is equal to the min of the battery service time in months and the number of recharge cycles as a function of depth of discharge, divided by 30 (one recharge cycle per day):

$$UPSDepr = \frac{C_{20h}\ prov. \times BatteryCostPerAh \times N_{servers}}{MIN\left(serviceLife, cycles(DoD)/30\right)} \quad (4.6)$$

We use the described equations to contrast LA with LFP technologies as we vary the peak time in the power profile, study the effect of decreasing battery cost per Ah, and identify the depth of discharge that minimizes TCO/server. Figure 4.6 shows the provisioned battery capacity for a given peak power time and a targeted reduction in peak power as well as the respective TCO/server reduction. More energy needs to be stored in the battery to achieve the same reduction in peak power as the duration of peak power demand increases. Hence, the cost of the distributed UPS increases. In the LA case, over-provisioning is no longer helpful when the peak power lasts for 12 hours. This means that the additional distributed UPS cost is greater than the reduction of TCO/server due to amortization of the infrastructure costs on more servers. LFP batteries remain beneficial at 12 hours of peak demand. Size constraints only allow shaving 5% of the 2-hour peak demand, in the LA case, while we can shave 5% of an 8-hour pulse with LFP. In the TCO/server diagrams in figure 4.6, we denote the battery capacities that do not fit in a 2U server by hatch shading the respective columns. For the same spike duration, it always makes sense to shave more peak with a bigger battery, within size limitations. To further quantify these profits, we find using the analysis of section 4.2 that 6.8% monthly TCO/server reduction translates to $6.4 per




(a) LA

(b) LFP

Figure 4.6: Battery capacities for different pulse widths and portion of peak power shaved. We also show the monthly TCO per server savings, assuming current battery costs, for the specified capacities of Lead-acid (LA) and Lithium Iron Phosphate (LFP) batteries. When the battery cannot fit within a 2U server, the associated savings are hatch shaded.

month per server, or more than $21M over the 10-year lifetime of a data center with 28,000 servers.

Figure 4.7 presents the monthly TCO/server savings as the battery costs change. The projection for LA batteries is that costs do not change, while LFP prices are expected to be reduced due to the push for cheaper hybrid and electric cars [Eco08]. For these graphs we assume that LFP cost reduces yearly at 8% [And09]. At 4h peak per day, we achieve 7% TCO/server reduction for lead-acid, ignoring space considerations, while this value drops to 1.35% for a battery that fits within a 2U server design. Using LFP batteries today we can achieve 8.5% TCO/server reduction and these savings will increase to 9.6% in the next 6 years.

Figure 4.8 presents the relation between depth of discharge and the TCO / server gains for both LA and LFP technology. There is a clear peak for the values 40% and 60% DoD, respectively. For low DoD values, the battery costs dominate the savings, because we need larger batteries to provide the same capping. For large DoD values, the lifetime of the battery decreases and more frequent replacements increase the UPS cost. The peak reduction of TCO/server occurs when the number of recharge cycles / 30 is equal to the battery service life. Note that due to the battery over-provisioning, less than 5% charge can sustain the server for 1 min and ensure data center continuity. Therefore, battery lifetime considerations affect TCO/server well before data center continuity becomes a concern.

To summarize our discussion on battery technologies and battery properties, we conclude: (1) Battery-based peak power shaving using existing batteries is only effective for brief spikes. To tolerate long spikes, larger batteries are necessary. However, the benefits from increased peak power shaving outweigh the extra battery costs even when high demand lasts 12 hours. (2) LFP is a better, more profitable choice than LA for frequent discharge/recharge cycles on distributed UPS designs. This is due to the increased number of cycles and longer service lifetime, better discharge characteristics, higher energy density, and the reduction in battery costs expected in the near future. (3) It makes sense to increase the capacity of the battery to the extent that it fits under the space constraints. This translates to increased power reduction and more savings. (4) For each battery

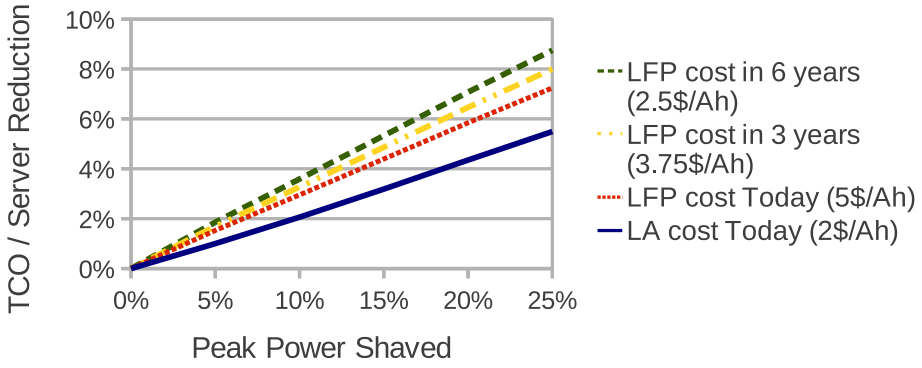
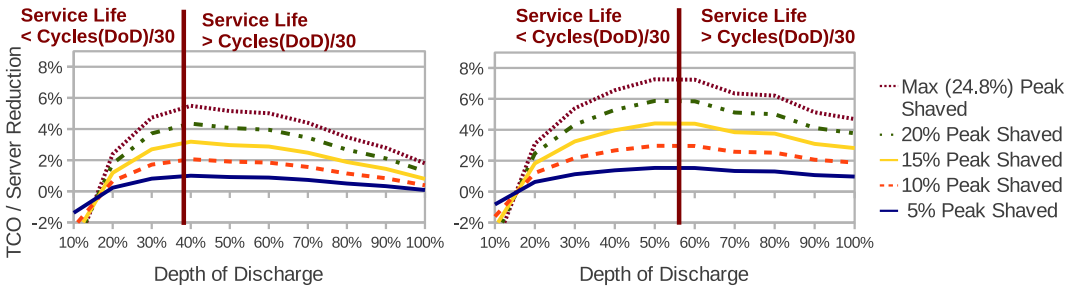



Figure 4.7: For the 2h pulse we show the projection of savings (ignoring space constraints) as the battery cost changes in the future [And09].




(a) LA (b) LFP

Figure 4.8: The relation between targeted depth of discharge and the reduction in TCO.

technology, there is a depth of discharge value that maximizes savings (40% for LA and 60% for LFP). This is the point where battery lifetime is no longer limited by the battery service time and needs to be replaced earlier due to frequent charging and discharging.

## 4.5 Policies

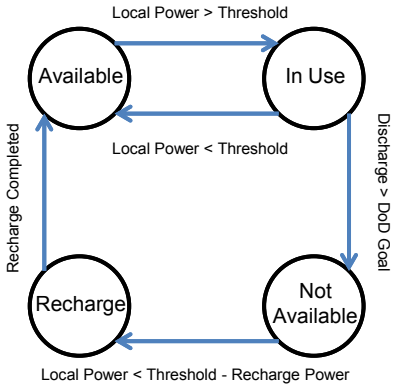
The analysis in the previous section assumes a simplified model of the power profile and perfect coverage of that peak by the batteries. As we move to a more

complex model of real data center workloads and the associated power profiles, we investigate a number of policies for peak power shaving which react to the observed load on the data center power infrastructure.

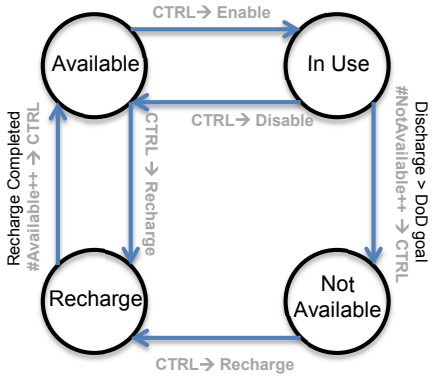
We evaluate three policies for peak power shaving using distributed, per-server batteries. We examine policies that operate at different levels of the power hierarchy. The first policy budgets power at the server level. The second operates at the PDU level to coordinate the batteries of all the servers powered by the same PDU. Finally, power budgeting at the cluster level coordinates all the machines in a cluster. The communication protocol to remotely enable/disable batteries or start recharge can be easily supported with existing network interfaces, such as SNMP or IPMI. The actual control algorithm can be implemented entirely in software. The policies manage battery discharge and also recharge. Recharging the batteries requires appreciable power and is thus best performed when the overall data center load is low. We consider the following policies:

**1. Server with Local Controller (ServCtrl)** When a server's power exceeds a preset threshold, this policy switches that server from grid power to battery power. The value of the power threshold defines how aggressively we cap power. When the overall power consumption is less than the threshold and there is sufficient margin to recharge the battery without exceeding the budget, battery recharge is enabled. Each server has its own local LFP battery and a controller that periodically monitors server power. Figure 4.9(a) shows the state machine for this controller. If measured server power is higher than the local power threshold (peak power cap / number of servers), then the controller switches the server to battery power. Recharge activates when battery depth of discharge reaches the set goal (60% for LFP) and there is sufficient margin between the current server power and the target power cap to accommodate recharge power.

**2. PDU with Centralized Controller (PduCtrl)**. This policy implements a controller per PDU. Each controller coordinates the operation of the batteries associated with the servers under a common PDU in order to match the energy to be shaved with the number of discharging batteries. It periodically estimates the power difference between current PDU power and the targeted PDU




(a) Server controller state machine




(b) PDU/cluster controller state machine

State=[NumAvail,NumInUse,NumNotAvailable,NumRecharging]
**/* NumAvail: Batteries with charge currently idle (Available state)**
**NumInUse: Batteries with charge currently discharging (Inuse)**
**NumNotAvail: Batteries without sufficient charge (NotAvailable)**
**NumRecharging: Batteries currently recharging (Recharge) */**

```
1: delta = load - threshold /* Get delta of current and targeted power */
2: ΔBats = abs(delta)/serverAveragePower      /* Get delta in batteries */
3: if (delta > 0) then
4:     EnBats = min(NumAvail, ΔBats)          /* Over peak goal */
5:     NumInUse += EnBats
6:     NumAvail -= EnBats
7:     Enable EnBats batteries
8: end if
9: if (delta < 0) and (ΔBats > 25) then
10:    DisBats = min(NumInUse,ΔBats)          /* Under peak goal */
11:    NumAvail += DisBats
12:    NumInUse -= DisBats
13:    Disable DisBats batteries
14:    RSlackBats = ΔBats-DisBats
15:    if RSlackBats > 0 then
16:        NumRecharging += RSlackBats
17:        Recharge RSlackBats batteries
18:    end if
19: end if
```

(c) PDU/cluster controller coordination algorithm

Figure 4.9: State machines for the local policy ServCtrl (a) and coordinated policies PduCtrl and ClustCtrl (b). The coordination algorithm is presented in (c)

peak. As soon as this delta becomes positive, the controller estimates the approximate number of batteries that should start discharging (*abs(delta) / serverAveragePower*). Similarly, when the delta is negative and there are discharging batteries, the local controller will signal a number of batteries proportional to the magnitude of the estimated difference to stop discharging. We introduce an additional condition that the number of batteries we want to stop needs to be more than 25, which provides some hysteresis. The value 25 is a function of how fast the workload changes and how fast our controller responds (controller period is 3 mins).

Figure 4.9(b) and 4.9(c) show the state machine for the local battery controller and pseudo-code for the algorithm running on the PDU level controller. Arcs labeled in light color correspond to events sent to or from the centralized controller, whereas the other arcs, such as determining when the DoD goal has been met, remain local decisions. The controller attempts to distribute the enabling and disabling of batteries evenly by employing a static sequence that is interleaved across racks. When no batteries are currently enabled, the controller gradually signals discharged batteries to recharge. The controller also forces batteries that have not yet discharged to the DoD goal, but have not recently been recharged, to begin recharging in anticipation of the next day's peak. Staggering recharge limits the possibility of power violations during low demand periods due to recharge power drawn from the utility.

**3. Cluster with Centralized Controller (ClustCtrl)**. This policy applies the same logic as *PduCtrl*, but at the cluster level. Data center power delivery involves circuit breakers at several levels of the hierarchy. The previous policy, *PduCtrl*, maintains a power budget at the PDU level allowing additional servers at the PDU level. This policy targets a power budget at the cluster level, enabling over-subscription at the cluster level. We again employ a sequence to enable and disable batteries to evenly distribute the power load across the levels of the power hierarchy.

## 4.6 Methodology

Section 4.2 derives upper bound power savings based on a simplified model of the workload and oracle knowledge of that workload. Here we present the tools used to model a variable, data-driven workload and realistic reactive capping policies that do not rely on oracle knowledge.

We developed a discrete-event simulator that captures the behavior of 1000 server nodes at the back-end of large distributed web applications. Each server is modeled as a queue with 8 consumers (cores) per server to service the incoming requests. Thus, we simulate a large network of M/M/8 queues. Our simulator monitors all levels of the data center power delivery hierarchy, namely the servers, racks, PDU, cluster, and data center. We measured the idle power of a Sun Fire X4270 server with a Hioki powermeter as 175W and the peak while fully utilized as 350W. We model server power as varying linearly between these two values, based on utilization.

For our results, we assume a distributed UPS with 12V LFP batteries attached to each server, provisioned at 40Ah, the maximum capacity that fits within the server size constraints. These batteries can sustain peak load for 92 minutes and take 2 hours to recharge once fully drained, measure 285.27 $in^3$ or 16% of the 2U rack space and should fit in front of the server in the cold aisle. To properly capture Peukert's effect during discharge, we recalculate remaining charge time every time the power draw on an individual server changes.

Table 4.4 presents the parameters of the workloads we use in our simulator. We assume a mix of web search, social networking, and MapReduce background computation. To capture the dynamic behavior of our workloads throughout the day, we use the Google Transparency Report [Goo12] and scale interarrival time accordingly. We collect the traffic data for a whole year (10/1/2010-9/30/2011) for two google products in the United States. Google unencrypted search represents search traffic, and Orkut represents social networking traffic (similar to Facebook). MapReduce is a company internal product and, as such, does not appear in the Transparency report. Instead, we reproduce the weekly waveform that appears in figure 3 of [CGGK11] and repeat it over the period of a year.

Table 4.4: Workloads

| Workload | Service Time Mean | Interarrival Time Mean | Reference |
|---|---|---|---|
| Search | 50ms | 42ms | [MSB+11] |
| Social Networking | 1sec | 445ms | [Apa12] |
| MapReduce | 2 mins | 3.3 mins | [CGGK11] |

Table 4.5: Relative traffic numbers as obtained from [Ale12]. MapReduce jobs are 15% of the load.

| Workload | Relative Normalized Traffic |
|---|---|
| Search | 29.2% |
| Social Networking | 55.8% |
| MapReduce | 15% |

We model a data center which serves all three types of workloads, with relative total demand placed on the servers in the ratios shown in Table 4.5. The relative loads of search vs Facebook/Orkut is chosen to match worldwide demand as reported by www.alexa.com [Ale12]. Note that we use Orkut data to define the shape of the social networking demand curve, but use Facebook data to gauge the magnitude of the load. The maximum daily peak of the aggregate load is set to 80% of the data center peak computational capability. This number leaves sufficient computing margin to ensure stability of the system, and is consistent with published data center design goals, as shown in figure 1 in [MSB+11]. Note that because of this restriction the peak observed value of the average server power, 315W, is less than than the peak achievable power of 350W.

Figure 4.10(a) shows the day-to-day variation of the daily data center energy. The yearly daily average corresponds to 240.1W per server and varies moderately throughout the year. Weekends and summer months demonstrate lower traffic. We test our policies on the three consecutive days with the highest demand in energy. Graph 4.10(b) zooms in on these days (11/17/2010-11/19/2010) and presents the daily power profile for each workload separately, as well as the

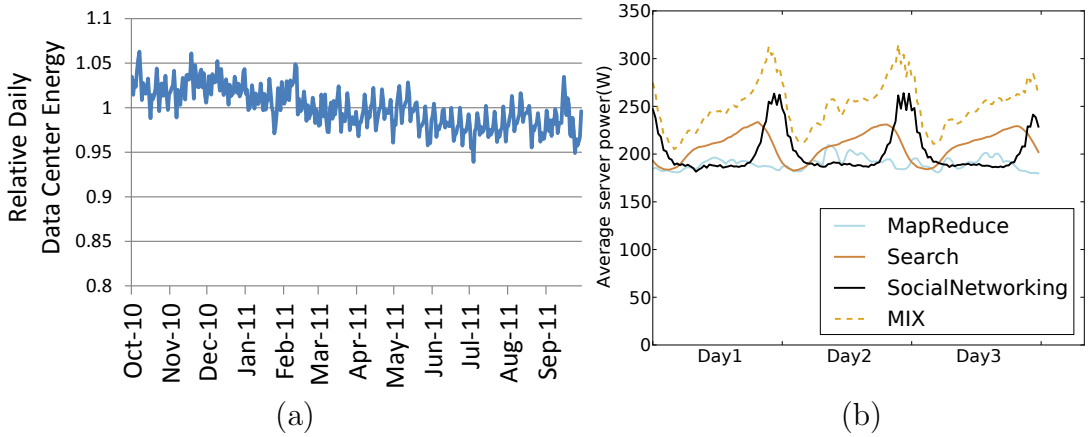



Figure 4.10: On the left we see the variation of data center energy throughout the year. During weekends and the summer traffic is lower. The average energy corresponds to 240.1W per server or utilization of 37.1%. On the right we zoom on the three days with highest energy requirements (11/17/2010-11/19/2010). We show average server power for each service and for MIX, the aggregate load. The average power for these days is 250.5W and the corresponding utilization 43%.

combined mix. The peaks of Search and Facebook are adjacent resulting in a waveform with broader peak. MapReduce traffic increases the variance of the graph.

We evaluate the workload mix in figure 4.10(b) under two different web service allocations: 1) restricting each service to its own dedicated set of servers (*split case*), 2) co-locating all web services, with highest priority for search, lower for social networking and lowest for MapReduce jobs (*mixed case*).

Additionally, we emulate the scheduling of jobs across individual servers. Specifically, we consider a simple round-robin scheduling policy, similar to the operation of a naive web traffic load balancer, and a load-aware policy with knowledge of server CPU utilization. This scheduler is responsible for allocating the work among servers and is independent from the per-server scheduler that maps jobs to specific cores. In our simulated data center, the load-aware policy is extremely effective at distributing the load evenly, probably unrealistically so. Thus, the round-robin scheduler represents a more uneven distribution of work. A deployed load-aware scheduler probably falls somewhere between the two.

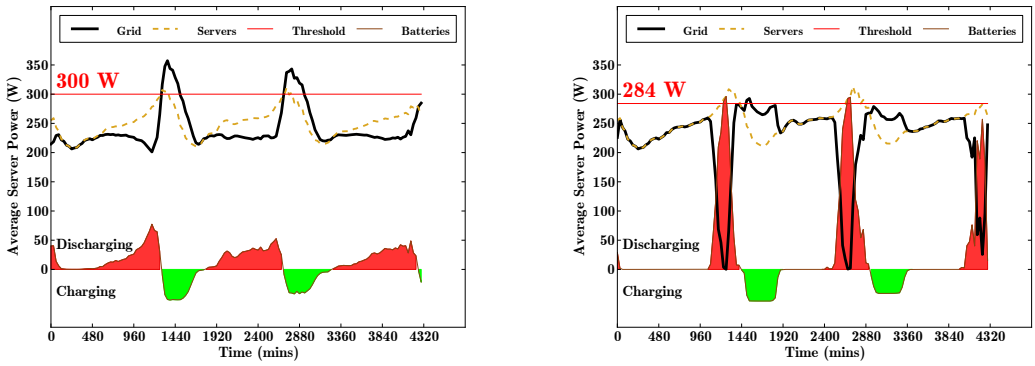
## 4.7 Results

The capacity of the battery, as well as the targeted depth of discharge, place an upper limit on the power capping that is possible for a given traffic pattern. In practice, though, the max achievable power capping also depends on the effectiveness of the policies that control the batteries. Setting the power capping threshold aggressively creates lower margins for wasted energy in our solution. There are two sources of battery energy waste: spatial and temporal. Spatial waste enables more batteries than necessary to shave a portion of overall power, while temporal waste enables batteries when capping is not required.

In this section, we gradually lower the peak power threshold until a policy begins to violate it. We show results for the lowest threshold (per server) that succeeds (horizontal line in figures 4.11, 4.12). Thus, we can compare policies based on that threshold. Some policies are not effective enough to cap power over a reasonable range. For those we give examples to illustrate why they fail. On an average day, it is to be expected that conservative estimates of peak power will result in a decent margin between battery capacity and the shaved peak load (some days the batteries may not be used at all). However, because we are modeling the worst days of the year, it is reasonable to expect that the available battery capacity is fully utilized. This methodology is reflective of what would happen on those days.

The *ServCtrl* policy (Figure 4.11(a)) assumes distributed, per-server batteries and does not require any centralized coordination. It relies completely on local server information. It is easy to implement, but due to the lack of coordination this scheme does not make efficient use of battery stored energy. Specifically, *ServCtrl* introduces temporal energy waste when transient effects create imbalances in the load across servers, resulting in battery discharge even if the total data center power does not exceed the threshold, leaving fewer batteries available to hide the real peak. We can even have batteries recharging during the peak. In the round-robin case, we cannot effectively shave peak for any meaningful power threshold,

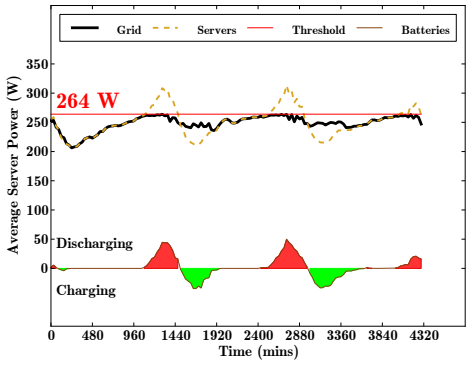
When very effective load-balancing is in place, we see fewer instances of



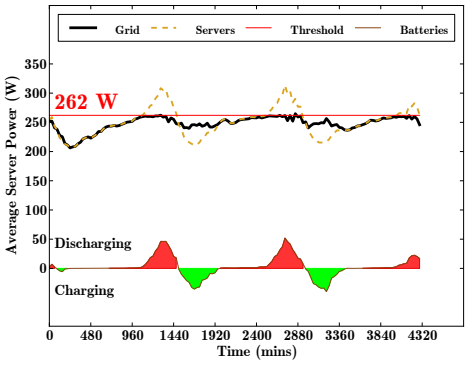

(a) ServCtrl - Round-robin



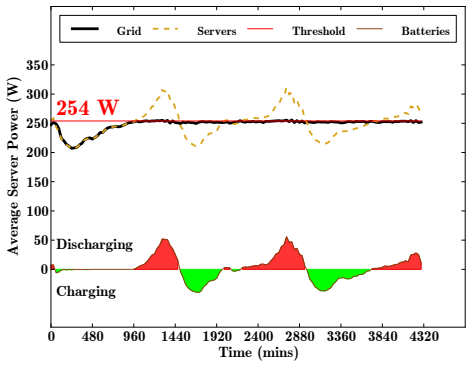

(b) ServCtl - Balanced



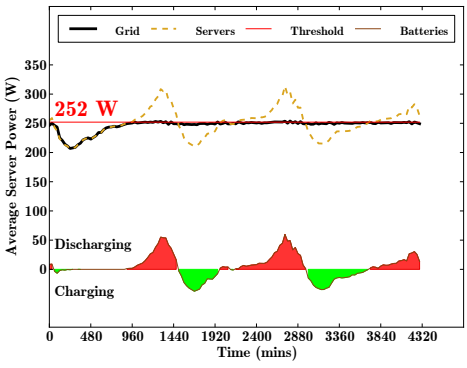

(c) PduCtrl - Round-robin




(d) PduCtl - Balanced




(e) ClustCtrl - Round-robin




(f) ClustCtrl - Balanced

Figure 4.11: These plots show the average server, grid and batter power during battery discharge and charge. Grid power is equivalent to server minus battery power. Power capping at higher power hierarchy levels is more effective.

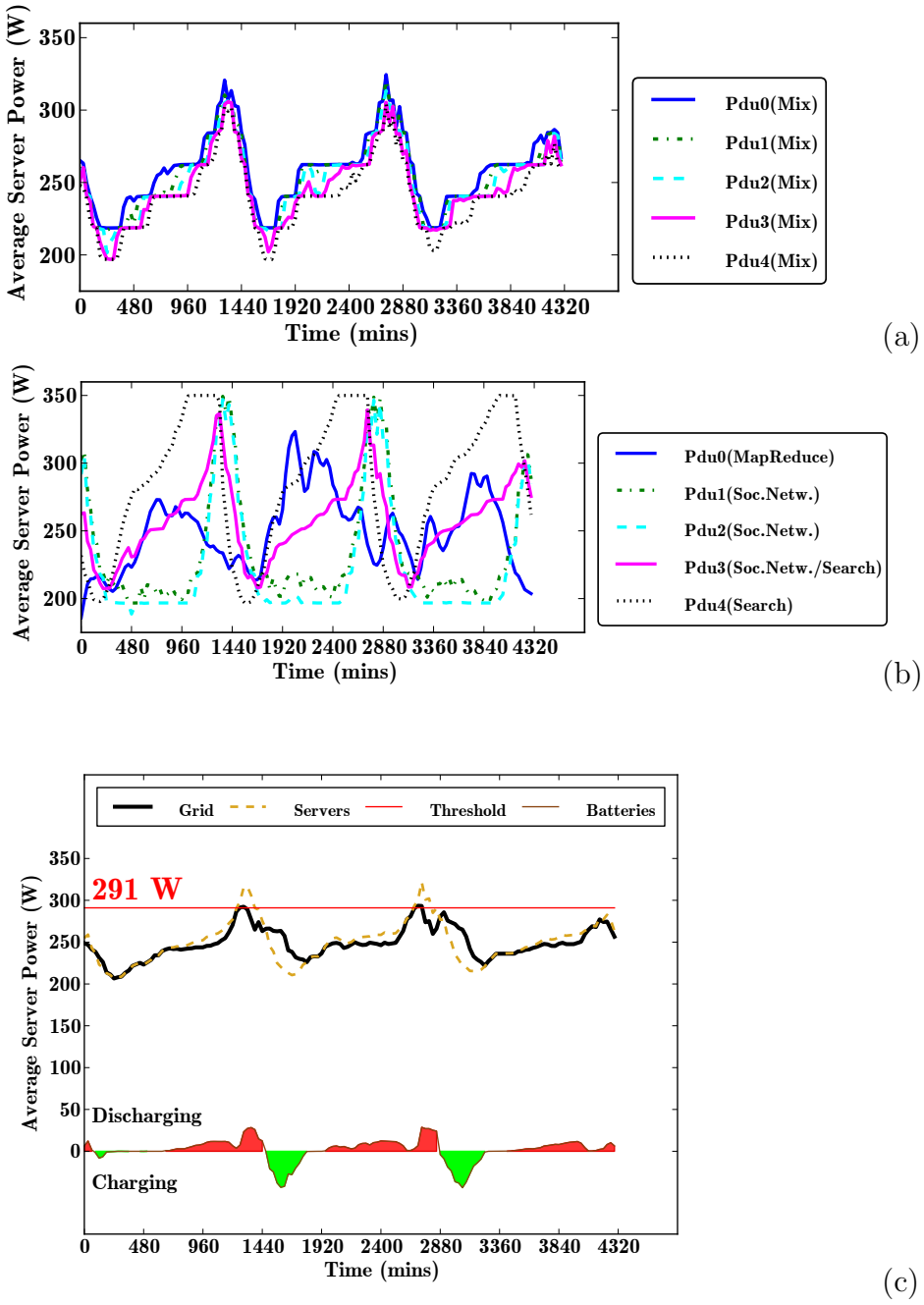



Figure 4.12: Here we quantify the effect of segmenting webservices into predefined PDUs. In (a) we show the server average power per PDU (without batteries) for the mixed case. In (b) we show the split case. When webservices run on split servers there are fewer available batteries to deal with a power peak. This is why in (c), when we use the batteries we can only guarantee peak power of 291W.

unnecessary discharge, but we observe a new problem. Once traffic increases to the degree that the power of each server crosses the threshold, all batteries begin discharging. As a result, a power dip follows. This effect is clearly visible in figure 4.11(d). Because the batteries reduce overall datacenter power well below the threshold, this overuses the total battery capacity. This is a similar effect experienced with power capping on a centralized UPS that can only produce power from the grid or the UPS, but not both. With this scheme, the batteries cannot sustain the peak, and grid power eventually exceeds the threshold.

There is a trade-off between recharge time and recharge current. Large values for recharge current (power) reduce recharge time but make it harder to find the necessary margin to initiate a recharge without violating the power budget. On the other hand, low recharge current provides ample margin for batteries to recharge, but risk having the battery still charging when the next peak arrives. For the *ServCtrl* policy we use a small recharge current of 3.7A (∼0.1C) In the coordinated policies, *PduCtrl* and *ClustCtrl*, the controller initiates the recharge of each server battery. It is much easier to find sufficient power slack to recharge a battery without violating the PDU or the cluster power budget respectively. For these policies, we use a high recharge current of 18.5A (∼0.5C) that corresponds to a charge time of 2 hours.

Figure 4.11b shows that the *PduCtrl* policy performs much better than *ServCtrl*, maintaining a power threshold of 264W for round-robin and 262W for the load-aware scheduler. This is the result of coordination among batteries to achieve a local cap at the PDU level. Just enough batteries in each PDU region are activated to reduce power below the threshold, thus preserving other batteries to ride out the full peak. Battery recharge is similarly coordinated so that no more than the available spare power is used for recharge. Global imbalances in the loads seen by each PDU result in slight noise in the total power; however, because each PDU is enforcing its threshold, that noise only results in grid power varying a little below the threshold.

That result holds when all three services run on all PDUs, because each PDU sees a similar power profile. For the *PduCtrl* we also study the scenario

where each service is allowed to run on a subset of the PDUs. In this case, batteries are statically partitioned. As a result, search batteries are not available to help with the Facebook peak, and vice versa. Globally, we have batteries charging and discharging at the same time, which is clearly suboptimal. The lowest power budget that we can enforce in the split case is 291W (figure 4.12). This analysis motivates resource sharing among applications, despite the associated complexity for fairness and quality of service.

Figures 4.11c, 4.11f show the *ClustCtrl* policy applied on the mixed scenario for the round-robin balancing and the load-aware balancing. The lowest power cap for this policy is 254W and 252W for the two cases. Note that both of these results are very close to the ideal scenario which would reduce power to 250W (average power for the worst day). This increased efficiency is a direct result of being able to take a more global view of power. Imbalances between the PDUs no longer result in undershooting the power threshold, allowing us to preserve batteries that much longer.

There are many considerations that might determine the right level to apply our battery control policies. Our results show that the policy becomes most effective as we move up the power hierarchy. Most importantly, the policy should be applied no lower than the level at which the component workloads of the datacenter are mixed together. These results indicate that with properly sized batteries and an effective control policy, we can do much more than shave the extreme peaks of the load – in fact, we almost completely flatten the power profile very close to the average power. Capping peak power from 315W to 254W corresponds to a reduction of 19.4%. This reduction will allow 24% more servers within the same provisioned power and reduce TCO/server by 6.3% (see section 4.2), resulting in more than $15M savings in costs for a data center with 28,000 servers over its lifetime.

**Guard band and DVFS – when projections fail** Prior work on power capping either applied performance degrading techniques, like DVFS, at peak load, or fall back to it as a failsafe when the batteries fail [GSU11]. However, applying techniques such as DVFS at peak load is often an unacceptable option. Many

datacenter applications track performance by watching the tail of the performance distribution – e.g., 95th percentile response time. Applying DVFS at peak load, even for a short time, can have a disastrous effect on those metrics. DVFS not only extends latencies, but also reduces throughput and induces higher queuing delays. Reducing performance at peak load increases the response time of those jobs already at the tail of the distribution.

Our technique does not apply DVFS, even when the peak power exceeds our conservative estimates, nor do we give up and allow the grid power to increase. In all the previous algorithms we disable the batteries once we hit the DoD goal (preserving battery lifetime – see Section 4.4). However, another benefit of the high DoD limit is additional stored energy in our batteries that can be used in case of emergency. With LFP per-server batteries there is approximately 35% guard band before we are in danger of not having sufficient reserves to survive a grid failure. This guard band can be used on days where the power profile exceeds worst-case peak power estimates. Our projections for the optimal DoD level were based on daily discharge; however, going below 40% to say, 35% or 30%, a couple times a year, or even once a month, will have no significant impact on the lifetime of the battery. Thus, we never need to apply performance-reducing mechanisms at peak load unless our estimated peak power is off by enormous margins. That does not mean that DVFS cannot still be an effective power management technique. But in our system, we would apply it during *low* utilization, where there is slack in the response times. By reducing power at low demand, we create more margin for recharging and accelerating the recharge cycle. This technique is not employed in the results shown in this chapter, but would allow us to shave even more power with minimal performance impact.

**Failure analysis** In large data centers it is common to cluster maintenance operations to reduce costs. This means that a non-negligible portion of batteries may be unusable for peak power shaving purposes before these batteries get replaced. Figure 4.13 shows how the lowest achievable peak changes when we assume that a portion of batteries has failed. We compare the best policy *ClustCtrl* with and without the use of the additional energy provided by discharging our batteries be-

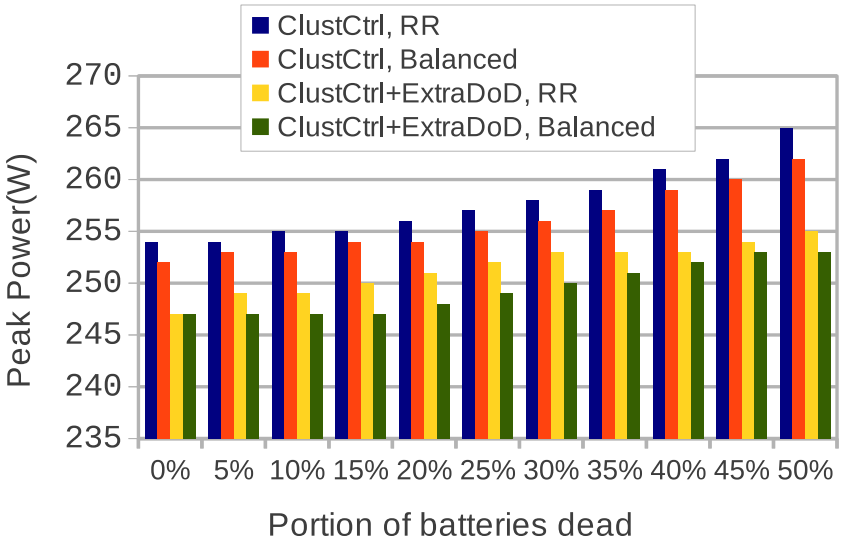



Figure 4.13: As the number of unusual batteries increase, the lowest possible peak power increases. Allowing to exceed our DoD goal occasionally, permits even higher peak power reduction. Load imbalances discharge batteries at different rates and make power capping harder.

yond the DoD goal. The peak threshold gradually increases with a larger portion of dead batteries. However, the increase is relatively small. Even when half of the batteries are dead we can still shave 16% of peak power. For these experiments, we find that we do not need to modify the algorithm of the controller to handle the unusable components. The controller signals a faulty component to start discharging, but no decrease in power takes place. As a result, the controller signals additional batteries in the next round and eventually corrects for the failure without any direct feedback. We also observe that the additional energy from (rare) deeper discharge of the batteries allows us to shave more power with fewer batteries. However, if the datacenter is allowed to enter deeper discharge frequently while dead batteries stack up for an extended period, then it can have an impact on the battery lifetime.

**Energy proportionality** The server used for this study is a representative modern platform, with idle power close to 50% of peak, based on our measurements. In the future, servers are expected to become increasingly energy proportional. We

model the impact of a server that is completely energy proportional.

Energy proportional servers essentially increase the height of the peak, relative to the average power, since power is significantly lower during off-peak periods. Consequently, we can further reduce the power threshold. Our simulations indicate the ability to reduce the peak observed power from 280W to 175W, a reduction of 37.5%. That results in an increase in server capacity of 60%.

## 4.8 Related work

**Peak Power Provisioning and Capping:** Reducing power consumption in server clusters is a well-studied problem in the literature [RLIC06, NS08, MGT09, GCU+09]. The overall idea is to combine CPU throttling, dynamic voltage/frequency scaling (DVFS), and switching entire servers on/off depending on the workload. Raghavendra, et al. [RLIC06] note that more efficient power management solutions are possible by managing power at the rack level than at individual blades. They devise proactive and reactive policies based on DVFS to cap power budgets at the rack level. Nathuji and Schwan [NS08] introduce the notion of power tokens to deal with heterogeneity across hardware platforms. Govindan, et al. [GCU+09] combine applications with heterogeneous characteristics in terms of burstiness. As a result, the power budget is exceeded statistically infrequently. DVFS is used as a failsafe mechanism to prevent against lasting peak power violations.

Femal et al. [FF05] were among the first to use formal control theory to maximize throughput while capping power. Raghavendra, et al. [RRT+08] extend the control theory idea to present a hierarchy of coordinated controllers that cap power across different levels of the power hierarchy and minimize performance impact. They argue for nesting controllers that operate at different time granularities to ensure stability and emphasize the information flow between the controllers.

**Using batteries in data centers:** Battery power management has been studied in the embedded/mobile system domain with various works proposing techniques to adjust the drain rate of batteries in order to elongate the system

operation time [PDR+01, RV03, Rak05, RP03]. Prior research has also investigated analytical models for battery capacity and voltage in portable devices [PDR+01, RP03, JH08].

The work of Chang, Pedram et al [PCKW10] investigates hybrid electric energy storage systems. These systems, which includes supercapacitors, li-ion as well as lead-acid batteries, can dynamically select the most appropriate energy storage device and drive down cost. Similarly to our work, they find li-ion battery technology to be more cost-effective than lead-acid for frequent use while shaving peak power. Admittedly, the idea of hybrid energy storage solutions in the context of data centers is worth further exploration. However, super-capacitors are better suited for really short and tall power spikes and do not map well to our observed data center loads. Additionally, their follow-up work [WKX+11] admits significant energy conversion loses to migrate charge between energy storage elements. This argues towards a simple solution with as few energy conversions as possible.

Govindan, et al [GSU11] introduce the idea of reducing data center peak power by leveraging the stored energy in a centralized UPS. During peak load, power from the UPS batteries augments the main grid, effectively hiding the peak from the utility service. During low load, the batteries recharge, consuming additional power.

In a follow-up work [GWSU12], they extend their prior work to also use distributed UPSs for peak power capping. That work focuses on power capping at the rack, using small lead-acid batteries to shave peak power. This approach allows them to prevent rare, brief power emergencies without performance degradation and relies on DVFS and load migration for peaks longer than several minutes. In our work, we examine solutions at multiple levels of the power hierarchy, show the financial advantages of more aggressive batteries with a more detailed model that incorporates battery lifetime considerations, and employ solutions that sacrifice no performance – the desired solution in a performance-sensitive data center under peak load.

In a separate work [GWC+11], the same authors also argue for a distributed UPS solution from a cost and reliability perspective. They find that a hybrid

distributed UPS placement, at PDU and server level, yields the most promising topology. They do not consider battery energy for peak power capping in that work, but this finding provides additional motivation for our work on the use of distributed batteries for power capping.

## 4.9 Conclusions

State-of-the-art data centers such as Google's and Facebook's have adopted a distributed UPS topology in response to the high cost associated with a centralized UPS design. In this work we explore the potential of using battery-stored energy in a distributed UPS topology to shave peak power. We describe how to provision the capacity of the battery and elaborate on how recharge cycles, the depth of discharge, and the workload power profile affect the potential for peak power shaving. We leverage the distributed nature of the batteries and design a controller to use them only when needed and thus prolong the duration of their usage, without violating the targeted power budget. Significant peak power reductions of up to 19.4%, are possible with our technique. Power capping in the data center context reduces over-provisioning of power delivery infrastructure, allows us to accommodate more servers under the same power budget and to reduce the TCO per server by 6.3%, significantly increasing the computation that can be done per facility and saving millions of dollars per datacenter.

## Acknowledgments

Chapter 4 contains material from Managing Distributed UPS Energy for Effective Power Capping in Data Centers, by Vasileios Kontorinis, Liuyi Zhang, Baris Aksanli, Jack Sampson, Houman Homayoun, Eddie Pettis, Dean Tullsen and Tajana Rosing, which appears in *Proceedings of the 39th International Symposium on Computer Architecture (ISCA)*. The dissertation author was the primary investigator and author of this paper. The material in this chapter is copyright ©2012 IEEE. Personal use of this material is permitted. However, permission to

reprint/republish this material for advertising or promotional purposes or for creating new collective works for resale or redistribution to servers or lists, or to reuse any copyrighted component of this work in other works must be obtained from the IEEE.

# Chapter 5

# Summary

Because of fundamental transistor scaling properties increasing active transistors per chip while keeping its power constant has become challenging. In this new regime, it is imperative to mitigate any form of power over-provisioning. In this thesis, we explore mechanisms that reduce power over-provisioning, and thus manage peak power, by powering-off unused or underutilized resources in three different domains: microprocessor core, across 3D-stacked dies, and the data center.

## 5.1 Core peak power management

An the core level we reduce the level of core over-provisioning by keeping a portion of the core, particularly those resources least required for performance, constantly off. We demonstrate that most applications have few resource bottlenecks and application can retain most of their performance as long as those critical resources are fully provided. We use this fact to dynamically provide the appropriate resources and power-gate the rest and in doing so we cap peak power.

Prior work on average power reduction relies on local, uncoordinated heuristics to power-off under-utilized resources. However, this approach does not provide any peak power guarantees as there can be periods of time where all resources are maximally configured and hence we still need to provision the power delivery for the worst case. In contrast, we manage the enabled and disabled resources centrally.

Centralized coordination ensures that not all resources are maximally configured at any time. To lower the overhead of dynamic reconfiguration and make a hardware implementation feasible we use a table-driven approach. A table placed next to the core holds valid configurations for a particular peak power budget. This approach can lower core peak power budget by 25% while giving up less than 5% of performance compared to the core with all resources enabled. Additionally, our dynamic approach outperforms the best static design that respects the power budget by more than 10%. Power capping in the core translates to 5.3% less on-chip decap area to achieve the same voltage variation goal, or 26% reduction in voltage variation for a given on-chip decoupling capacitance.

There is nothing preventing a single architecture from having multiple tables, or modifying the table contents over time, switching between them either at verification/test time, bootup time, or even runtime. This raises a number of new and promising potential uses of this technology to be explored in future work.

(1) We could use it in conjunction with Dynamic Thermal Management – thermal events trigger the core to go into a lower peak power envelope. We could even have different power envelopes for other events – plugged in/not plugged in, low battery, etc.

(2) We could account for thermal heterogeneity in the processor (e.g., cores in the center tend to run hot, those on the edges cooler [CSTSR09]) by applying a different peak power envelope to different cores.

(3) Similarly, we could counteract the effects of process variation by filling the ROM at verification time with a set of configurations which match the exact thermal and power characteristics of that individual core.

(4) Our technique can be coupled with fault tolerance. If a hard error is detected on an adaptable component, then the configurations that incorporate that specific partition can be disabled, as part of the core's overall fault isolation solution. In this way, we naturally redirect the power from faulty components to those that can use them.

(5) We could maintain an overall processor peak power envelope, but use it judiciously to maximize global throughput. We allow a core that can use the extra

power for more performance to go to $P + \Delta$ while another is forced to $P - \Delta$. Our architecture, applied to a CMP, already provides a mechanism for transforming a homogeneous design into a heterogeneous architecture (each core configured separately to make the best use of its peak power limit). This optimization further enhances that ability, as we now have the ability to set the peak power limit of each core according to the needs of the individual application.

## 5.2 Resource pooling for power efficiency

In the context of 3D stacked dies we reduce over-provisioning by eliminating idle resources across the boundary of a core. Previous attempts to share resources across cores in the 2D plane were limited by long wire delays and high communication latency. With a 3D architecture, we can dynamically pool bottleneck resources within a single cycle and therefore enable much more efficient resource sharing.

To make this architecture possible we need to address several key issues. We discuss circuit level implementation details for the resources we pool: the reorder buffer, instruction queues, and register files. Our design partitions the pooled resources and dynamically allocates each partition so that it is owned by a single core at any point of time. We quantify the overheads associated with our technique in terms of additional logic and potential frequency degradation, which are found to be small and we show that fine grain resource partitioning is not required, because back-end resources get populated in bursts.

We advocate the use of a low-power core instead of a complex, high-power core to deal with thermal and power concerns raised because of stacking logic on top of logic. With this approach, we can design a simpler, less over-provisioned core and reclaim the performance difference using resource pooling. To achieve significant improvement we have to carefully balance the degree of flexibility on pooling resources. Allocating large portions of core resources to the common pool may lead to resource starvation, allocating small portions limits the potential benefits of pooling.

The performance of the low-power core is on average 67% of the high-power core performance. With resource pooling and all four cores active, low-power core performance improves by 9% and reaches 71% of the high-power core performance. With only two out of four cores used, simple core performance is 27% better and reaches 86% of complex core performance. When a single core is active and the rest cores contribute to the pool, performance improves by 45% and the simple core operates at 97% of the performance of the complex core.

This study has opened several new research directions that are worth pursuing in the future.

(1) Pooling core back-end resources exposes different bottlenecks. For example we cannot take advantage of the additional instruction level parallelism exposed, unless we have sufficient instruction issue bandwidth. Additionally, there are applications that depend on first level cache to hide long latency misses. By pooling additional resources at the core level, such as execution units and first level cache ways we can potentially achieve even better performance.

(2) This architecture can be useful for finer granularity thermal management. If the register file is a hotspot on one core, we can completely decommission the registers on that core without stopping (or perhaps even slowing) execution on that core. Through adaptation we can mitigate hot spots.

(3) We can also leverage 3D pooling to provide fine-grain reconfiguration around faulty components. Once we identify a partition of pooled resource as faulty, we can remove it from the pool and continue chip operation until the pool is completely emptied.

## 5.3 Managing peak power for data centers

Finally, we study an idea to reduce over-provisioning costs for data centers. We use the energy stored in distributed UPS batteries to provide additional power capacity during high activity and recharge the batteries during low activity.

We perform a detailed analysis of data center total cost of ownership and make the associated models publicly available. Our analysis, shows that the bene-

fits from power capping justify investing in larger, more capable per server batteries. Additionally, we find that a different battery technology than the lead acid is better suited for aggressive battery-based power capping. Lithium Iron Phosphate batteries offer an order of magnitude more recharge cycles and higher energy density than lead acid batteries, hence are more cost-effective once re-purposed for frequent use.

We also design a controller that activates battery charge and discharge in the data center and we discuss alternatives regarding the level in the power hierarchy where we apply the power capping and the associated trade-offs. Cluster level power capping is more beneficial than server or power supply unit level capping. Our best performing techniques can reduce provisioned power by 19.5%. By allowing 24% more server under the same power infrastructure we better amortize initial, one-time capital investments in the data center and we decrease total cost of ownership per server by 6.3%. This reduction corresponds to $15M savings for the lifetime of a data center.

This study examines the applicability of battery-based power capping for data centers with distributed UPSs. There are several ways to extend this work in the future.

(1) We can explore the same idea of battery-based power capping for data centers with heterogeneous server configurations. Several data centers contain servers with different capabilities. As ideas like specialization gain prominence we will see more embedded or streaming devices employed in this context.

(2) The idea of battery capping can be combined with dynamic voltage frequency scaling (DVFS). These two techniques can be complimentary. DVFS may negatively affect performance by decreasing throughput at high utilization levels, while battery-based capping does not. We can aggressively use DVFS during low utilization periods and create additional margins for battery based capping.

(3) We can explore power capping together with workload migration. Virtualization in modern data centers significantly simplifies load consolidation. Once we treat battery energy as a resource we need to incorporate battery charge awareness in virtual machine placement management algorithms.

## 5.4 Concluding remarks

Our ability to extract performance under tight power budgets will define how long performance trends will remain exponential. Key to achieving this is removing any form of over-provisioning in the computing stack. This thesis presents techniques towards this direction in three different domains: the core, the chip and the data center.

# Bibliography


[AA 12] AA Portable Power Corporation. Portable Power Product design, assembly and quality control. http://www.batteryspace.com/lifepo4cellspacks.aspx, 2012.

[ABD+03] David H. Albonesi, Rajeev Balasubramonian, Steven G. Dropsho, Sandhya Dwarkadas, Eby G. Friedman, Michael C. Huang, Volkan Kursun, Grigorios Magklis, Michael L. Scott, Greg Semeraro, Pradip Bose, Alper Buyuktosunoglu, Peter W. Cook, and Stanley E. Schuster. Dynamically tuning processor resources with adaptive processing. *IEEE Computer*, December 2003.

[AG] International Electron Devices Meeting 2002 Keynote Luncheon Speech Andy Grove. http://www.intel.com/pressroom/archive/speeches/grove_20021210.pdf.

[Alb99] D. H. Albonesi. Selective cache-ways: On demand cache resource allocation. In *International Symposium on Microarchitecture*, 1999.

[Ale12] Alexa. Web information, traffic metrics, search analytics, demographics for websites. http:/www.alexa.com, 2012.

[And09] D. Anderson. An evaluation of current and future costs for lithium-ion batteries for use in electrified vehicle powertrains. Master's thesis, Duke University, 2009.

[AP07] Behnam Amelifard and Massoud Pedram. Optimal selection of voltage regulator modules in a power delivery network. In *the annual conference on Design Automation*, 2007.

[Apa12] Apache. Olio, a web 2.0 benchmark. http://incubator.apache.org/olio/, 2012.

[APC08] APC. InfraStruxure Total Cost of Ownership. http://www.apc.com/tools/isx/tco/, 2008.

[BAS+01] Alper Buyuktosunoglu, David Albonesi, Stanley Schuster, David Brooks, Pradip Bose, and Peter Cook. A circuit level implementation of an adaptive issue queue for power-aware microprocessors. In *Great Lakes Symposium on VLSI*, 2001.

[BG01] James Burns and Jean-Luc Gaudiot. Area and system clock effects on SMT/CMP processors. In *International Conference on Parallel Architectures and Compilation Techniques*, 2001.

[BKAB03] A. Buyuktosunoglu, T. Karkhanis, D.H. Albonesi, and Pradip Bose. Energy efficient co-adaptive instruction fetch and issue. In *International Symposium on Computer Architecture*, 2003.

[BM01] D. Brooks and M. Martonosi. Dynamic thermal management for high-performance microprocessors. In *International Symposium on High Performance Computer Architecture*, 2001.

[BNWS04] B. Black, D.W. Nelson, C. Webb, and N. Samra. 3D processing technology and its impact on IA32 microprocessors. In *International Conference on Computer Design*, 2004.

[BTM00] David Brooks, Vivek Tiwari, and Margaret Martonosi. Wattch: A framework for architectural-level power analysis and optimizations. In *International Symposium on Computer Architecture*, 2000.

[CAH+10] D. Cuesta, J. Ayala, J. Hidalgo, M. Poncino, A. Acquaviva, and En. Macii. Thermal-aware floorplanning exploration for 3D multi-core architectures. In *Great Lakes Symposium on VLSI*, 2010.

[CAR+10] Ayse K. Coskun, David Atienza, Tajana Rosing, Thomas Brunschwiler, and Bruno Michel. Energy-efficient variable-flow liquid cooling in 3D stacked architectures. In *Design Automation and Test in Europe*, 2010.

[CGB97] Yi-Shing Chang, Sandeep K. Gupta, and Melvin A. Breuer. Analysis of ground bounce in deep sub-micron circuits. In *IEEE VLSI Test Symposium*, 1997.

[CGGK11] Yanpei Chen, Archana Ganapathi, Rean Griffith, and Randy Katz. The case for evaluating MapReduce performance using workload suites. In *Technical Report No. UCB/EECS-2011-21*, 2011.

[Cli11] Climate Savers Computing. Power supply efficiency specifications. http://www.climatesaverscomputing.org/resources/certification, 2011.

[CSTSR09] Ayse K. Coskun, Richard Strong, Dean M. Tullsen, and Tajana Simunic Rosing. Evaluating the impact of job scheduling and power management on processor lifetime for chip multiprocessors. In *International joint conference on measurement and modeling of computer systems (SIGMETRICS)*, 2009.

[DBB+02] S. Dropsho, A. Buyuktosunoglu, R. Balasubramonian, D. H. Albonesi, S. Dwarkadas, G. Semeraro, G. Magklis, and M. L. Scott. Integrating adaptive on-chip storage structures for reduced dynamic power. Technical report, Univ. of Rochester, 2002.

[DGR+74] R.H. Dennard, F. H. Gaensslen, V. L. Rideout, E. Bassous, and A. R. LeBlanc. Design of Ion-Implanted MOSFET's with Very Small Physical Dimensions. In *IEEE Journal of Solid-State Circuits*, October 1974.

[Duk09] Duke Energy. Utility bill. http://www.duke-energy.com/pdfs/ scscheduleopt.pdf, 2009.

[Dup07] Dupont Fabros Technology, Inc. Sec filing (s-11) 333-145294, August 9, 2007.

[Eco08] Economist. In search of the perfect battery. http://www.mitenergyclub.org/assets/2009/9/25/Economist_Batteries 2008.pdf, March 2008.

[EE08] Stijn Eyerman and Lieven Eeckhout. System-level performance metrics for multiprogram workloads. *IEEE Micro*, 28:42–53, May 2008.

[Ele01] Electic motor sport. EV construction, thundersky batteries. http://www.electricmotorsport.com/store/ems_ev_parts_batteries .php, 2001.

[Fac11] Facebook. Hacking conventional computing infrastructure. http://opencompute.org/, 2011.

[FF05] Mark E. Femal and Vincent W. Freeh. Boosting data center performance through non-uniform power allocation. In *ACM International Conference on Autonomic Computing*, 2005.

[FKM+02] Krisztián Flautner, Nam Sung Kim, Steve Martin, David Blaauw, and Trevor Mudge. Drowsy caches: simple techniques for reducing leakage power. In *International Symposium on Computer Architecture*, 2002.

[FWB07] Xiaobo Fan, Wolf-Dietrich Weber, and Luiz Andre Barroso. Power provisioning for a warehouse-sized computer. In *International Symposium on Computer Architecture*, 2007.

[GAT03] Ed Grochowski, David Ayers, and Vivek Tiwari. Microarchitectural di/dt control. In *IEEE Design and Test*, 2003.

[GCU+09] Sriram Govindan, Jeonghwan Choi, Bhuvan Urgaonkar, Anand Sivasubramaniam, and Andrea Baldini. Statistical profiling-based techniques for effective power provisioning in data centers. In *EuroSys*, 2009.

[GFA+08] Shantanu Gupta, Shuguang Feng, Amin Ansari, Jason A. Blome, and Scott A. Mahlke. The stagenet fabric for constructing resilient multicore systems. In *International Symposium on Microarchitecture*, 2008.

[Goo09] Google Summit. http://www.google.com/corporate/datacenter/events/dc-summit-2009.html, 2009.

[Goo12] Google. Transparency report. http://www.google.com/transparencyreport/traffic/, 2012.

[GSU11] Sriram Govindan, Anand Sivasubramaniam, and Bhuvan Urgaonkar. Benefits and limitations of tapping into stored energy for datacenters. In *International Symposium on Computer Architecture*, 2011.

[GWC+11] Sriram Govindan, Di Wang, Lydia Chen, Anand Sivasubramaniam, and Anand Sivasubramaniam. Towards realizing a low cost and highly available datacenter power infrastructure. In *HotPower Workshop on Power-Aware Computing and Systems*, 2011.

[GWSU12] Sriram Govindan, Di Wang, Anand Sivasubramaniam, and Bhuvan Urgaonkar. Leveraging stored energy for handling power emergencies in aggressively provisioned datacenters. In *International Conference on Architectural Support for Programming Languages and Operating Systems*, 2012.

[HAG+10] Michael B. Healy, Krit Athikulwongse, Rohan Goel, Mohammed M. Hossain, Dae Hyun Kim, Young-Joon Lee, Dean L. Lewis, Tzu-Wei Lin, Chang Liu, Moongon Jung, Brian Ouellette, Mohit Pathak, Hemant Sane, Guanhao Shen, Dong Hyuk Woo, Xin Zhao, Gabriel H. Loh, Hsien-Hsin S. Lee, and Sung Kyu Lim. Design and analysis of 3D-maps: A many-core 3D processor with stacked memory. In *IEEE Custom Integrated Circuits Conference*, 2010.

[Har09] Frank Harvey. Listing of ev batteries. http://www.39pw.us/car/batteryTable.html, 2009.

[HB09] Urs Hoelzle and Luiz Andre Barroso. *The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines*. Morgan and Claypool Publishers, 2009.

[HBS+04] Zhigang Hu, Alper Buyuktosunoglu, Viji Srinivasan, Victor Zyuban, Hans Jacobson, and Pradip Bose. Microarchitectural techniques for power gating of execution units. In *International Symposium on Low Power Electronics and Design*, 2004.

[HDC10] Xiang Hu, Peng Du, and Chung-Kuan Cheng. Exploring the rogue wave phenomenon in 3D power distribution networks. In *IEEE Conference on Electrical Performance of Electronic Packaging and Systems*, 2010.

[HVE+07] M. Healy, M. Vittes, M. Ekpanyapong, C. S. Ballapuram, S. K. Lim, H.-H. S. Lee, and G. H Loh. Multiobjective microarchitectural floorplanning for 2-D and 3-D ICs,. In *International Conference on Computer-Aided Design*, 2007.

[IBC+06] Canturk Isci, Alper Buyuktosunoglu, Chen-Yong Cher, Pradip Bose, and Margaret Martonosi. An analysis of efficient multi-core global power management policies: Maximizing performance for a given power budget. In *International Symposium on Microarchitecture*, 2006.

[IBM03] IBM Corporation. PowerPC 750 RISC. In *Microprocessor Technical Summary*, August 2003.

[IKKM07] E. Ipek, M. Kirman, N. Kirman, and J.F. Martinez. Core fusion: Accommodating software diversity in chip multiprocessors. In *International Symposium on Computer Architecture*, 2007.

[Int00] Intel Corp. *Intel Pentium 4 Processor in the 423-pin Package Thermal Design Guidelines, Datasheet*, November 2000.

[JH08] M.R. Jongerden and B.R. Haverkort. Battery modeling. Technical report, TR-CTIT-08-01, CTIT, 2008.

[Kar98] Ashish Karandikar. Low power SRAM design using hierarchical divided bit-line approach. In *International Conference on Computer Design*, 1998.

[KFJ+03] Rakesh Kumar, Keith I. Farkas, Norman P. Jouppi, Parthasarathy Ranganathan, and Dean M. Tullsen. Single-ISA Heterogeneous Multi-core Architectures: The Potential for Processor Power Reduction. In *MICRO 36: Proceedings of the 36th annual IEEE/ACM International Symposium on Microarchitecture*, 2003.

[KHM01] Stefanos Kaxiras, Zhigang Hu, and Margaret Martonosi. Cache decay: exploiting generational behavior to reduce cache leakage power. In *International Symposium on Computer Architecture*, 2001.

[KJT04] Rakesh Kumar, Norman P. Jouppi, and Dean M. Tullsen. Conjoined-core chip multiprocessing. In *International Symposium on Microarchitecture*, 2004.

[KMW98] R.E. Kessler, E.J. McLellan, and D.A. Webb. The alpha 21264 microprocessor architecture. In *International Conference on Computer Design*, 1998.

[Kon12] Vasileios Kontorinis. Battery-aware Data Center TCO models. http://cseweb.ucsd.edu/ tullsen/DCmodeling.html, 2012.

[KSG+07] Changkyu Kim, Simha Sethumadhavan, M. S. Govindan, Nitya Ranganathan, Divya Gulati, Doug Burger, and Stephen W. Keckler. Composable lightweight processors. In *International Symposium on Microarchitecture*, 2007.

[KTJ06] Rakesh Kumar, Dean M. Tullsen, and Norman P. Jouppi. Core architecture optimization for heterogeneous chip multiprocessors. In *International Conference on Parallel Architectures and Compilation Techniques*, 2006.

[KTJR05] R. Kumar, D.M. Tullsen, N. Jouppi, and P. Ranganathan. Heterogeneous chip multiprocessing. *IEEE Computer*, November 2005.

[KTR+04] Rakesh Kumar, Dean M. Tullsen, Parthasarathy Ranganathan, Norman P. Jouppi, and Keith I. Farkas. Single-ISA Heterogeneous Multi-core Architectures for Multithreaded Workload Performance. In *International Symposium on Computer Architecture*, June 2004.

[KZL+10] Aman Kansal, Feng Zhao, Jie Liu, Nupur Kothari, and Arka A. Bhattacharya. Virtual machine power metering and provisioning. In *ACM Symposium on Cloud Computing*, 2010.

[LB08] Benjamin C. Lee and David Brooks. Efficiency trends and limits from comprehensive microarchitectural adaptivity. In *International Conference on Architectural Support for Programming Languages and Operating Systems*, 2008.

[LKL11] Harold Lim, Aman Kansal, and Jie Liu. Power budgeting for virtualized data centers. In *USENIX*, 2011.

[LNR+06] Feihui Li, C. Nicopoulos, T. Richardson, Yuan Xie, V. Narayanan, and M. Kandemir. Design and management of 3D chip multiprocessors using network-in-memory. In *International Symposium of Computer Architecture*, 2006.

[LXB07] Gabriel H. Loh, Yuan Xie, and Bryan Black. Processor design in 3D die-stacking technologies. *IEEE Micro*, 27:31–48, May 2007.

[MBB01] Roberto Maro, Yu Bai, and R. Iris Bahar. Dynamically reconfiguring processor resources to reduce power consumption in high-performance processors. In *Proceedings of the First International Workshop on Power-Aware Computer Systems-Revised Papers*, 2001.

[MD05] J. W. Schultze M. Datta, T. Osaka. *Microelectronics Packaging*. CRC Press, 2005.

[MES+07] P. Muthana, A.E. Engin, M. Swaminathan, R. Tummala, V. Sundaram, B. Wiedenman, D. Amey, K.H. Dietz, and S. Banerji. Design, modeling, and characterization of embedded capacitor networks for core decoupling in the package. *Transactions on Advanced Packaging*, 2007.

[MGT09] David Meisner, Brian Gold, and Wenisch Thomas. Powernap: Eliminating server idle power. In *International Conference on Architectural Support for Programming Languages and Operating Systems*, 2009.

[MJDS08] Ke Meng, Russ Joseph, Robert P. Dick, and Li Shang. Multi-optimization power management for chip multiprocessors. In *International Conference on Parallel Architectures and Compilation Techniques*, 2008.

[MSB+11] David Meisner, Christopher M. Sadler, Luiz André Barroso, Wolf-Dietrich Weber, and Thomas F. Wenisch. Power management of online data-intensive services. In *International Symposium on Computer Architecture*, 2011.

[MZM+09] N. Madan, Li Zhao, N. Muralimanohar, A. Udipi, R. Balasubramonian, R. Iyer, S. Makineni, and D. Newell. Optimizing communication and capacity in a 3D stacked reconfigurable cache hierarchy. In *High-Performance Computer Architecture*, 2009.

[NKH+07] K. Najeeb, Vishnu Vardhan Reddy Konda, Siva Kumar Sastry Hari, V. Kamakoti, and Vivekananda M. Vedula. Power virus generation using behavioral models of circuits. In *VLSI Test Symposium*, 2007.

[NS07] Ripal Nathuji and Karsten Schwan. Virtualpower: coordinated power management in virtualized enterprise systems. In *ACM Symposium on Operating Systems Principles*, 2007.

[NS08] Ripal Nathuji and Karsten Schwan. Vpm tokens: virtual machine-aware power budgeting in datacenters. In *ACM Symposium on High-Performance Parallel and Distributed Computing*, 2008.

[PAV+01] Michael D. Powell, Amit Agarwal, T. N. Vijaykumar, Babak Falsafi, and Kaushik Roy. Reducing set-associative cache energy via way-prediction and selective direct-mapping. In *International Symposium on Microarchitecture*, 2001.

[PCKW10] Massoud Pedram, Naehyuck Chang, Younghyun Kim, and Yanzhi Wang. Hybrid electrical energy storage systems. In *International Symposium on Low Power Electronics and Design*, 2010.

[PDR+01] Debashis Panigrahi, Sujit Dey, Ramesh R. Rao, Kanishka Lahiri, Carla-Fabiana Chiasserini, and Anand Raghunathan. Battery life estimation of mobile embedded systems. In *VLSI Design*, 2001.

[PJS97] S. Palacharla, N.P. Jouppi, and J.E. Smith. Complexity-effective superscalar processors. In *International Symposium on Computer Architecture*, 1997.

[PKG06] Dmitry V. Ponomarev, Gurhan Kucuk, and Kanad Ghose. Dynamic resizing of superscalar datapath components for energy efficiency. *IEEE Transactions on Computers*, February 2006.

[PL06] Kiran Puttaswamy and Gabriel H. Loh. Dynamic instruction schedulers in a 3-dimensional integration technology. In *Great Lakes Symposium on VLSI 2006*, 2006.

[PL07] K. Puttaswamy and G.H. Loh. Thermal herding: Microarchitecture techniques for controlling hotspots in high-performance 3D-integrated processors. In *High-Performance Computer Architecture*, 2007.

[PMF08] M. Popovich, A. V. Mezhiba, and E. G. Friedman. *Power Distribution Networks with On-Chip Decoupling Capacitors*. Springer, 2008.

[PMZ+10] Steven Pelley, David Meisner, Pooya Zandevakili, Thomas F. Wenisch, and Jack Underwood. Power routing: dynamic power provisioning in the data center. In *International Conference on Architectural Support for Programming Languages and Operating Systems*, 2010.

[Pon10] Ponemon Inst. *National Survey on Data Center Outages*, 2010.

[PPWT00] Mondira Deb Pant, Pankaj Pant, D. Scott Wills, and Vivek Tiwari. Inductive noise reduction at the architectural level. In *International Conference on VLSI Design*, 2000.

[pr07] SAVVIS press release. Savvis sells asserts related to two datacenters for $200 million. http://www.savvis.com/en-US/Company/News/Press/Pages/SAVVIS+Sells+Assets+Related+to+Two+Data+Centers+for+200+Million.aspx, June 29 2007.

[PR08] V.S. Pandit and Woong Hwan Ryu. Multi-ghz modeling and characterization of on-chip power delivery network. In *IEEE Conference on Electrical Performance of Electronic Packaging*, Oct. 2008.

[Rak05] Daler N. Rakhmatov. Battery voltage prediction for portable systems. In *IEEE International Symposium on Circuits and Systems*, 2005.

[RL11] T. B. Reddy and D. Linden. *Linden's Handbook of Batteries (4th edition)*. McGraw-Hill, 2011.

[RLIC06] Parthasarathy Ranganathan, Phil Leech, David Irwin, and Jeffrey Chase. Ensemble-level power management for dense blade servers. In *International Symposium on Computer Architecture*, June 2006.

[RMD+05] P. Royannez, H. Mair, F. Dahan, M. Wagner, M. Streeter, L. Bouetel, J. Blasquez, H. Clasen, G. Semino, J. Dong, D. Scott, B. Pitts, C. Raibaut, and Uming Ko. 90nm low leakage soc design techniques for wireless applications. In *IEEE International Solid-State Circuits Conference*, 2005.

[RNA+12] Efraim Rotem, Alon Naveh, Avinash Ananthakrishnan, Doron Rajwan, and Eliezer Weissmann. Power-management architecture of the intel microarchitecture code-named sandy bridge. *IEEE Micro*, February 2012.

[RP03] Peng Rong and Massoud Pedram. An analytical model for predicting the remaining battery capacity of lithium-ion batteries. In *Design Automation and Test in Europe*, 2003.

[RRT+08] Ramya Raghavendra, Parthasarathy Ranganathan, Vanish Talwar, Zhikui Wang, and Xiaoyun Zhu. No "power" struggles: coordinated multi-level power management for the data center. In *International Conference on Architectural Support for Programming Languages and Operating Systems*, 2008.

[RV03] Daler N. Rakhmatov and Sarma B. K. Vrudhula. Energy management for battery-powered embedded systems. *ACM Transactions in Embedded Computing Systems*, 2003.

[SAT+08] K. Sakuma, P. S. Andry, C. K. Tsang, S. L. Wright, B. Dang, C. S. Patel, B. C. Webb, J. Maria, E. J. Sprogis, S. K. Kang, R. J. Polastre, R. R. Horton, and J. U. Knickerbocker. 3D chip-stacking technology with through-silicon vias and low-volume lead-free interconnections. In *IBM Journal of Research and Development*, November 2008.

[Shm05] Roger R. Shmidt. Liquid cooling is back. *Electronics Cooling*, 2005.

[SHP+09] Amirali Shayan, Xiang Hu, He Peng, Wenjian Yu, Wanping Zhang, Chung-Kuan Cheng, Mikhail Popovich, Xiaoming Chen, Lew Chua-Eaon, and Xiaohua Kong. Parallel flow to analyze the impact of the voltage regulator model in nanoscale power distribution network. In *International Symposium on Quality Electronic Design*, 2009.

[SK09a] John Sartori and Rakesh Kumar. Distributed peak power management for many-core architectures. In *Design Automation and Test in Europe*, March 2009.

[SK09b] John Sartori and Rakesh Kumar. Three scalable approaches to improving many-core throughput for a given peak power budget. In *High Performance Computing Conference*, December 2009.

[Sma11] SmartGauge. Peukert's law equation. http://www.smartgauge.co.uk/peukert.html, 2011.

[SPHC02] Timothy Sherwood, Erez Perelman, Greg Hamerly, and Brad Calder. Automatically characterizing large scale program behavior. In *International Conference on Architectural Support for Programming Languages and Operating Systems*, October 2002.

[SSH+03] Kevin Skadron, Mircea R. Stan, Wei Huang, Sivakumar Velusamy, Karthik Sankaranarayanan, and David Tarjan. Temperature-aware microarchitecture. In *International Symposium on Computer Architecture*, 2003.

[ST00] A. Snavely and D. M. Tullsen. Symbiotic jobscheduling for a simultaneous multithreading architecture. In *Proceedings of International Conference on Architectural Support for Programming Languages and Operating Systems*, 2000.

[STR08] Maciej Swierczynski, Remus Teodorescu, and Pedro Rodriguez. Lifetime investigations of a lithium iron phosphate (LFP) battery system connected to a wind turbine for forecast improvement and output power gradient reduction. In *Stationary Battery Conference, BattCon*, 2008.

[TB09] W. Pitt Turner and Kenneth G. Brill. Cost Model: Dollars per kW plus Dollars per Square Floor of Computer Floor, Uptime Inst. White paper, 2009.

[Tez] Tezzaron Semiconductor. www.tezzaron.com.

[Tho10] T. Thorolfsson. Two 3DIC case studies: Memory-on-logic and logic-on-logic. In *IBM Research Student Workshop on 3D System Integration*, 2010.

[TMAP08] Shyamkumar Thoziyoor, Naveen Muralimanohar, Jung Ho Ahn, and Norman P.Jouppi. Tech report CACTI 5.1. Technical report, HPL, 2008.

[TRfS03] International Technology Roadmap for Semiconductors. http://public.itrs.net, 2003.

[Tul96] D.M. Tullsen. Simulation and modeling of a simultaneous multithreading processor. In *22nd Annual Computer Measurement Group Conference*, December 1996.

[Uni03] Battery University. Online university education about batteries. http://batteryuniversity.com/, 2003.

[Uni04] Granich Unikowsky. Allocating decoupling capacitors to reduce simultaneous switching noise on chips. *MIT PhD Thesis*, 2004.

[VHW+07] Balaji Vaidyanathan, Wei-Lun Hung, Feng Wang, Yuan Xie, Vijaykrishnan Narayanan, and M.J. Irwin. Architecting microprocessor components in 3D design space. In *International Conference on VLSI Design*, 2007.

[WDW10] Yasuko Watanabe, John D. Davis, and David A. Wood. WiDGET: Wisconsin decoupled grid execution tiles. In *International Symposium on Computer Architecture*, 2010.

[Win09] Windsun. Lead-acid batteries: Lifetime vs Depth of discharge. http://www.windsun.com/Batteries/Battery_FAQ.htm, 2009.

[WKX+11] Yanzhi Wang, Younghyun Kim, Qing Xie, Naehyuck Chang, and Massoud Pedram. Charge migration efficiency optimization in hybrid electrical energy storage (hees) systems. In *International Symposium on Low Power Electronics and Design*, 2011.

[WSLL10] Dong Hyuk Woo, Nak Hee Seong, D.L. Lewis, and H.-H.S. Lee. An optimized 3D-stacked memory architecture by exploiting excessive, high-density TSV bandwidth. In *High-Performance Computer Architecture*, 2010.

[YCH07] Hao Yu, Chunta Chu, and Lei He. Off-chip decoupling capacitor allocation for chip package co-design. In *the annual conference on Design Automation*, 2007.

[YDT+05] Soner Yaldiz, Alper Demir, Serdar Tasiran, Yusuf Leblebici, and Paolo Ienne. Characterizing and Exploiting Task-Load Variability and Correlation for Energy Management in Multi-Core Systems. In *Workshop in Embedded Systems for Real-Time Multimedia*, 2005.

[YKT+09] Hiroshi Yoshikawa, Atsuko Kawasaki, Tomoaki, Iiduka, Yasushi Nishimura, Kazumasa Tanida, Kazutaka Akiyama, Masahiro Sekiguchi, Mie Matsuo, Satoru Fukuchi, and Katsutomu Takahashi. Chip scale camera module (cscm) using through-silicon-via (TSV). In *IEEE International Solid-State Circuits Conference*, 2009.

[ZHC02] Yumin Zhang, Xiaobo (Sharon) Hu, and Danny Z. Chen. Task scheduling and voltage selection for energy minimization. In *the annual conference on Design Automation*, 2002.

[ZLI+11] Lu-Lu Zhang, Gan Liang, Alexander Ignatov, Mark C. Croft, Xiao-Qin Xiong, I-Ming Hung, Yun-Hui Huang, Xian-Luo Hu, Wu-Xing Zhang, and Yun-Long Peng. Effect of Vanadium Incorporation on Electrochemical Performance of LiFePO4 for Lithium-Ion Batteries. In *Journal of Physical Chemistry*, June 2011.

[ZWX+00] Xunwei Zhou, Pit-Leong Wong, Peng Xu, F.C. Lee, and A.Q. Huang. Investigation of candidate VRM topologies for future microprocessors. *Transactions on Power Electronics*, Nov 2000.

[ZXD+08] Xiuyi Zhou, Yi Xu, Yu Du, Youtao Zhang, and Jun Yang. Thermal management for 3D processors via task scheduling. In *International Conference of Parallel Processing*, 2008.