Energy Optimising Methodologies On Heterogeneous Data Centres

Rajiv Nishtala

Department of Computer Architecture
Universitat Politècnica de Catalunya

This dissertation is submitted for the degree of
Doctor of Philosophy

Barcelona, July 2017

This page is intentionally left blank.

Energy Optimising Methodologies On Heterogeneous Data Centres

by
Rajiv Nishtala

A Dissertation
Presented to the Department of Computer Architecture
at
Universitat Politècnica de Catalunya
in Candidacy for the Degree of
Doctor of Philosophy.

Thesis Advisors:

Xavier Martorell Bofill, Prof.
Universitat Politècnica de Catalunya, Spain

Daniel Mossé, Prof.
University of Pittsburgh, USA

Barcelona, July 2017

This page is intentionally left blank.

Energy Optimising Methodologies On Heterogeneous Data Centres

This page is intentionally left blank.

Abstract

In 2013, U.S. data centres accounted for 2.2 % of the country's total electricity consumption, a figure that is projected to increase rapidly over the next decade. A significant proportion of power consumed within a data centre is attributed to the servers, and a large percentage of that is wasted as workloads compete for shared resources. Many data centres host interactive workloads (e.g., web search or e-commerce), for which it is critical to meet user expectations and user experience, called Quality of Service (QoS). There is also a wish to run both interactive and batch workloads on the same infrastructure to increase cluster utilisation and reduce operational costs and total energy consumption. Although much work has focused on the impacts of shared resource contention, it still remains a major problem to maintain QoS for both interactive and batch workloads. The goal of this thesis is twofold. First, to investigate how, and to what extent, resource contention has an effect on throughput and power of batch workloads via modelling. Second, we introduce a scheduling approach to determine on-the-fly the best configuration to satisfy the QoS for latency-critical jobs on any architecture.

To achieve the above goals, we first propose a modelling technique to estimate server performance and power at runtime called *Runtime Estimation of Performance and Power* (**REPP**). REPP's goal is to allow administrators' control on power and performance of processors. REPP achieves this goal by estimating performance and power at multiple hardware settings (dynamic frequency and voltage states (DVFS), core consolidation and idle states) and dynamically sets these settings based on user-defined constraints. The hardware counters required to build the models are available across architectures, making it architecture agnostic.

We also argue that traditional modelling and scheduling strategies are ineffective for interactive workloads. To manage such workloads, we propose **Hipster** that combines both a heuristic, and a reinforcement learning algorithm to manage interactive workloads. Hipster's goal is to improve resource efficiency while respecting the QoS of interactive workloads. Hipster achieves its goal by exploring the multicore system

and DVFS. To improve utilisation and make the best usage of the available resources, Hipster can dynamically assign remaining cores to batch workloads without violating the QoS constraints for the interactive workloads.

We implemented REPP and Hipster in real-life platforms, namely 64-bit commercial (Intel SandyBridge and AMD Phenom II X4 B97) and experimental hardware (ARM big.LITTLE Juno R1). After obtaining extensive experimental results, we have shown that REPP successfully estimates power and performance of several single-threaded and multiprogrammed workloads. The average errors on Intel, AMD and ARM architectures are, respectively, 7.1 %, 9.0 %, 7.1 % when predicting performance, and 8.1 %, 6.5 %, 6.0 % when predicting power. Similarly, we show that when compared to prior work, Hipster improves the QoS guarantee for Web-Search from 80 % to 96 %, and for Memcached from 92 % to 99 %, while reducing the energy consumption by up to 18 % on the ARM architecture.

Contents

This page is intentionally left blank.

List of Figures

List of Tables

To,
Amma, Nanna, Akka,
and the late Rajeswara Rao Tattagaru.

Acknowledgements

This thesis would never have been possible without the ideas, friendship, support, and assistance of numerous people. Foremost among them is, of course, Xavier Martorell, who is an infinitely patient advisor, caring and understanding. His immense faith in my abilities and our research group is shown in the diverse research topics we could explore. And yet, on each of these topics, he was a constant source of ideas and inspiration, and I feel enormously privileged to have him as my advisor. In addition to his undying enthusiasm and energy, he also directed me to numerous people to generate a wide range of scientific contacts that made this journey successful.

I am grateful to Marc Gonzalez for the initial insights on building the power and performance models in the early stages of my PhD. His guidance and willingness to engage in discussions made the initial proceedings of this thesis possible.

I owe a lot of thanks to Paul M. Carpenter. Firstly, for guiding me to produce high quality results and for lending me a book on writing style [1]. I cannot thank him enough for his generosity and willingness to help in the hard times of my PhD i.e., with multiple paper rejections and not knowing what to do. The amount of time spent together discussing problems (be it late evenings or mid afternoon), brainstorming ideas, and comprehending some concepts from text books was invaluable in understanding the causal-effect relationships presented in chapter 5. This PhD would have been harder without Paul teaching me – consciously and unconsciously – how to understand a problem in an abstract way. It was a great pleasure having him as my advisor at Barcelona Supercomputing Center.

For over half a decade, I have had the pleasure to work with Daniel Mossé who taught me how to sketch out an idea and design experiments. His dedication and resilience to do excellent research, by focusing on each data point and each aspect of this, is shown in section 3.4.

I especially owe much to Vinicius Petrucci, Tomasz Patejko and Pieter Bellens, who took time during and after numerous cups of coffee, for the countless days of brainstorming (with their humorous comments), correcting papers and showing

me how to rewrite a scientific article. Being a slow learner, they taught me twice. Many atimes my crazy ideas bounced off them and came back to me looking *even more crazy*. We had a lot of fruitful discussions and I enjoyed working with them.

My friends in Barcelona (now everywhere) made my life both in and outside of work engaging and enjoyable. These people are (in alphabetical order) Alfonso Rodríguez, Amandine Iseux, Joanna Hoddinott, Kallia Chronaki, Mary Ocampo, Rahul Gayatri, Rommel Sanchez, Sicong Zhuang and Siddharth Bhaskara. Although we did not get a chance to work together, their support and patience to listen to all my failures and joy of paper acceptances was considerate. They always cheered me up after a disastrous evening with, a long bicycle ride, good food and deep philosophical conversation.

Since I am half-way into my Academic Awards speech here anyway, I also want to thank my parents Padmavathi and Anand Kumar Nishtala. Ever since the beginning of elementary school you have always insisted in me the importance of education (even if that meant pulling me from underneath the bed!) and raised me with the love and importance of science and math from an early age. Amma, I will always remember you waking me up every single day during my stay away from home and for teaching me to be patient even in hard times. Nanna, I clearly remember you taking time from your busy schedule to learn and teach me to program in BASIC and LOGO at the age of 7 – your dedication and work ethic are truly inspiring. Along with my sister Dr. Nirupama Nishtala, my family continued to believe in me even when I did not. I hope I made them proud.

Last, but certainly not least, I want to thank my best friend, Aleksandra Dokurno for her moral support, for listening to all my problems, and to each day's success and failures. Her astonishing drive to pay meticulous attention to detail is contagious, and without her, I might still be the one without "any" academic papers! I would also like to thank her for staying up with me – at times – on the long nights of paper writing and giving me amazing food. Thank you for what you have done to me.

Cheers to y'all, you made this ride memorable!

This work has been partially supported by the European Union FP7 program through the Mont-Blanc-2 (FP7-ICT-610402) project, by the Ministerio de Economia y Competitividad under contract Computación de Altas Prestaciones VII (TIN2015-65316-P), and the Departament de Innovació, Universitats i Empresa de la Generalitat de Catalunya, under project MPEXPAR: Models de Programació i Entorns d Execució Paral·lels (2014-SGR-1051).

Nomenclature

Latin Letters

$c_n = \arg\max_c R(w_n, c)$	The action with the highest reward, c_n among the possible set of actions, c, for the current state, w_n
c_n	The action chosen from a finite set of alternatives for the current state, w_n, in time interval t_n to t_{n+1}
F	Cache Friendly batch workload
N	Insensitive batch workload
$Power_{reward}$	The ratio of TDP to $Power$
QoS_{curr}	Currently measured tail latency at the 95^{th} or 99^{th} percentile for the latency-critical workload
QoS_D	QoS as a factor of a co-efficient between 0 and 1
QoS_{reward}	The ratio of $QoS_{current}$ to QoS_{target}
QoS_S	QoS as a factor of a co-efficient between 0 and QoS_D
QoS_{target}	The target tail latency for the latency-critical workload
$R(w, c)$	A lookup table of rewards for state w and action c
S	Thrashing batch workload
T	Cache Fitting batch workload
w_n	The current *"state"* in MDP refers to load of the latency-critical workload measured during time interval t_{n-1} to t_n
w_{n+1}	The next *"state"* in MDP based on the current action, c_n

Greek Letters

α_c	Current configuration, that is, (P_c, Cl_c)
α_f	Future configuration, that is, (P_f, Cl_f)
χ	Load (IPS) or power in a step-wise monotonic function
$\chi(\kappa)$	κ^{th} element in the data sequence for either load or power
$\chi(min), \chi(max)$	The minimum, maximum value of load or power
Cl_c	Current Cl-State
Cl_f	Future Cl-State
Cl_i	Intermediate Cl-State
Δ, β	Coefficients in the multi linear regression model
η_c	Performance (IPS) at current configuration
η_f	Performance (IPS) at future configuration
γ	Discounting factor in MDP
λ_n	Positive or negative reward in MDP for the chosen action, c_n is updated at t_{n+1}
$\mathcal{P}$	Number of DVFS states in a multicore architecture
$\mathcal{S}$	Number of cores in a multicore architecture
ω	The performance or power constraint per core
P_c	Current frequency for DVFS state
P_f	Future frequency for DVFS state
P_i	Intermediate frequency for DVFS state
ψ	Number of datapoints before the peak in a step-wise monotonic function
ρ_c	Power at current configuration

ρ_f	Power at future configuration
τ	Change_factor for the step-wise monotonic function
$\sum_{n=0}^{\infty} \gamma^n \lambda_n$	The aim of MDP is to maximise this function, total discounted reward
ξ	Learning factor in MDP

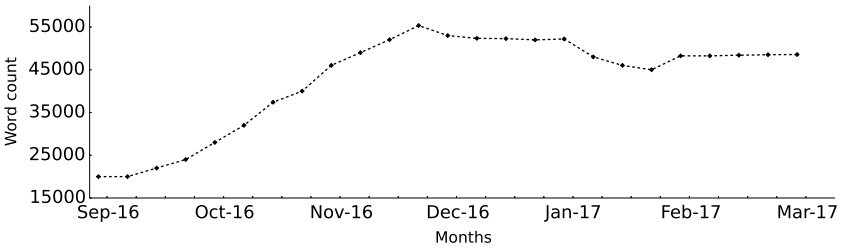
Abbreviations

AAE	Absolute Average Error
ACPI	Advanced Configuration and Power Interface
AR	Activity Ratio
Big Core	ARM Cortex-A57
BPU	Branch Predictor Unit
CFS	Completely Fair Scheduler
Cl-States	Idle states
DPM	Dynamic Power Management
DSDT	Differentiated State Description Table
DVFS	Dynamic Voltage and Frequency Scaling
FACTS	Frequency and Contention-Aware Thread Collocation Schema
FE	Front End
FP	Floating Point Unit
ILP	Instruction Level Parallelism
INT	Integer Unit
IPC	Instructions Per Cycle
IPS	Instructions Per Second
ISA	Instruction Set Architecture

L1	L1 Cache Memory
L2	L2 Cache Memory
LLC	Last Level Cache
MDP	Markov Decision Process
MIPS	IPS in millions
MSR	Machine Specific Registers
PAAE	Percentage Absolute Average Error
PUE	Power Usage Effectiveness
QoS	Quality of Service
QPS	Queries per Second
RAPL	Running Average Power Limit
REPP-C	Runtime Estimation of Performance and Power with workload consolidation
REPP	Runtime Estimation of Performance and Power
RL	Reinforcement learning
RPS	Requests per Second
Small Core	ARM Cortex-A53
STDEV	Standard Deviation
TDP	Thermal Design Power

Part I:

Prologue



This graph shows the number of words/week I wrote during the course of this thesis.

CHAPTER 1

Introduction

In 2013, U.S data centres consumed 91 billion kilowatt-hour, which is enough to power every single house in New York City twice [2, 3]. This is approximately 5×10^{10} kg of greenhouse gas emissions, equivalent to the total volume of gas *emitted by the United Kingdom, the 5^{th} largest economy in the world* [4]. Due to energy shortages and the difficulty to produce power at the levels predicted, energy efficiency has, in fact, become a major issue across the whole computing spectrum, from mobile devices to data centres and high-performance computing. For this reason, major chip manufacturers and designers, such as Intel Corp. [5] and ARM [6], nowadays sacrifice (single thread) performance for the sake of energy efficiency. This has been a recent development (i.e., over the last decade) while scientists have held different opinions throughout the past four decades regarding performance vs. energy efficiency. The debate started with Gordon Moore [7] advocating for an increase in performance through an increase in frequency and components complexity (high power), whereas recent reports from Intel Corp [5] suggest many small, simple cores (low power).

In addition, large-scale data centres like Facebook [8], Google [9], Twitter [10], Amazon [11], Rackspace [12] etc., operate under strict power limitations as energy constitutes approximately 30 % [13, 14] of the operating cost of a data centre. On the one hand, this implies intermittent high loads or software behaviours in data centres may breach hard power safety limits [15]. On the other hand, despite having strict power budgets, web-services like Facebook need to deliver services within a certain time frame otherwise users are likely to leave [16]. Recent study has shown that marginal delays (of hundreds of milliseconds) in Quality of Service (QoS) can impact revenue massively [17].

To improve energy efficiency (i.e., performance per watt of power consumed) in data centres, we first need to understand *where?* and *when?* do data centres spend a lot of time and energy? Several studies concluded that among all the components of a data

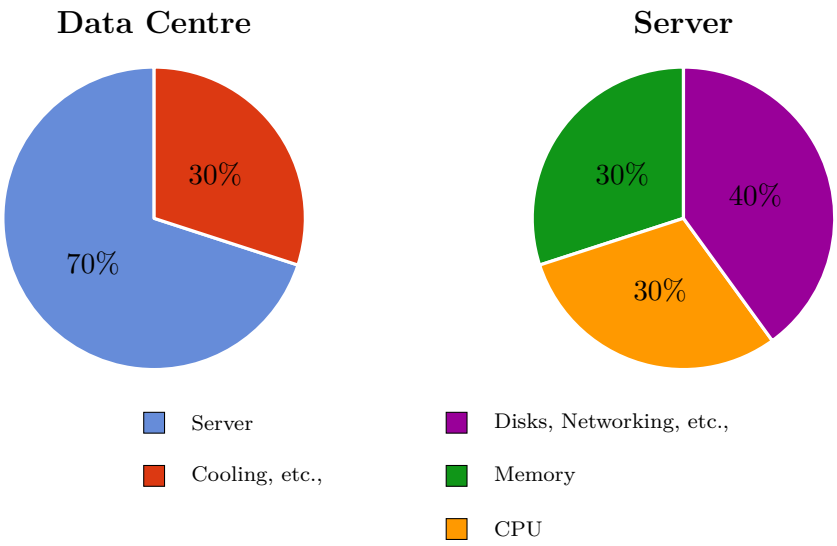


Figure 1.1 Power consumption breakdown in a data centre and server.

centre (servers, cooling, and other infrastructure), the servers consume approximately 70 % to 90 % of the total power consumption (see [18] and Figure 1.1). As for the energy consumption within a server, the CPU [19] and memory [20, 21] are the most dominant consumers [22] with each consuming approximately 30 % [23].

To improve energy efficiency in data centres, we also need to understand *how* do data centres spend the power distributed? Several studies [24–27] have shown that data centres like Google and Facebook operate at peak load only for a small fraction of the day (Figure 1.2, < 2 h), where the power consumption and load are proportional. However, for the remainder of the day data centres are often under-utilised (at or below 60 % mark) while the system power consumption is greater than 60 % of the maximum. My[1] thesis focuses on *modelling the performance and power of CPU and memory* (Part II), the two most power-hungry data centre resources, and proposing *scheduling strategies to improve energy efficiency* (Part III).

1.1 Challenges

We aim to increase data centre energy efficiency by targeting the problem of data centre particular demands and shared resource contention. Concretely, we address the following challenges:

[1] In what follows, unless stated otherwise, I will use the words: I, My, We, Our refer to my exclusive contributions

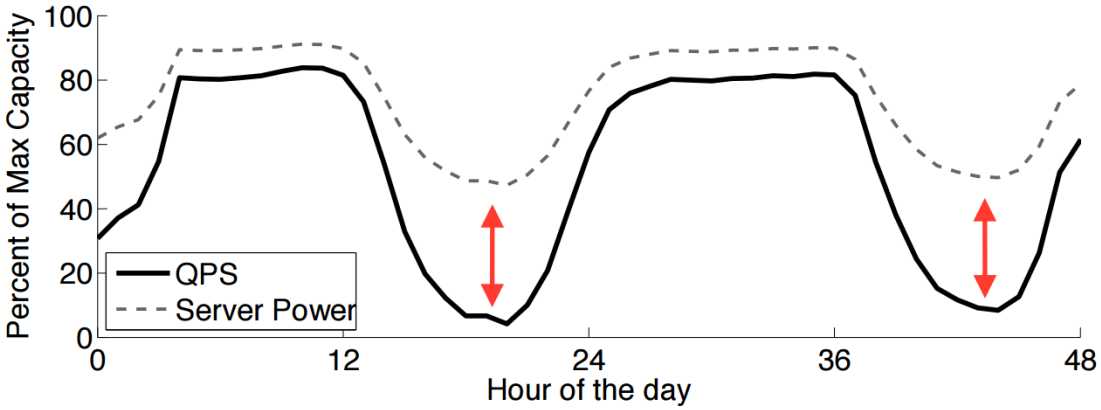


Figure 1.2 Typical server load (in Queries per Second – QPS) and power consumption in Google data centres. Workload is given as a percentage of maximum allowed workload. (Reproduced from [28, 29])

Shared Resource Contention. Despite significant research efforts [30–36], mitigating shared resource contention on multicore processors *still* remains a major factor for performance and power loss. A multicore server system (schematically shown in Figure 1.3) has a set of cores sharing multiple resources like the L2 cache, Last Level Cache (LLC) and DRAM memory, memory controllers, memory bus, etc. The latency to fetch data from different levels of cache hierarchy is known as a miss penalty and is measured in computational cycles. The miss penalty is defined as the ratio of the memory access time to the inverse of core/socket frequency. Intuitively, one may believe that a higher miss rate would cause a lower performance (that is, lower instructions per cycle (IPC) or instructions per second (IPS)). In addition to shared resource contention, current server systems are also limited by their memory size [35] and network saturation [37]. Part II of my thesis focuses on power and performance prediction in shared resource contention environment. This is a challenge for multiple reasons: (1) It is unclear how to use the hardware counter information (if any) to understand power and performance trade-offs. (2) Some multicore systems in highly complex data centres [38] do not necessarily provide all necessary hardware resource information [39] to estimate power and performance accurately. These power and performance estimations are essential to control power consumption and determine scalability on-the-fly. Part III of my thesis focuses on scheduling approaches to mitigate shared resource contention to improve energy efficiency.

Data centre particularities. Regardless of the current computational demand, server CPUs run at high load ($> 60\,\%$) for short periods, even with services like

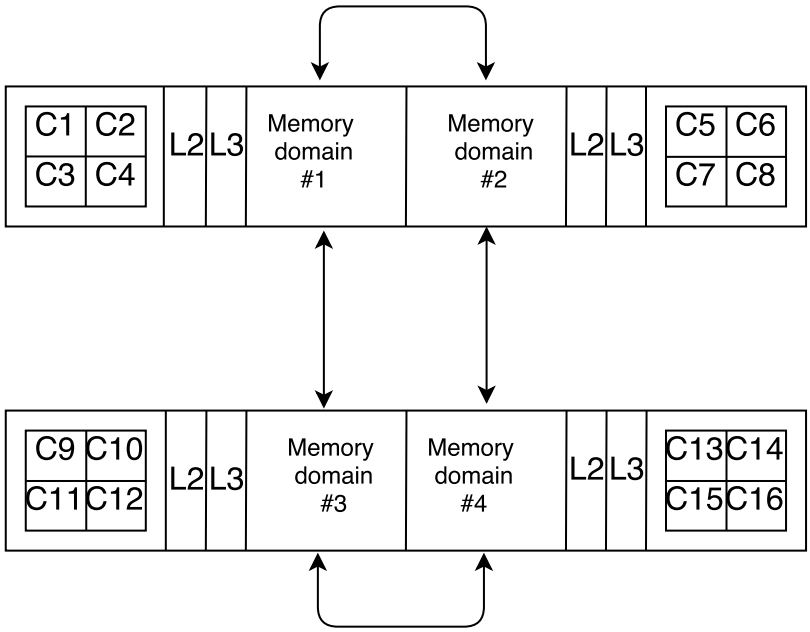


Figure 1.3 High level view of multicore server system. Each domain contains 4 cores with a shared L2 and LLC per domain.

Amazon Elastic Compute offered for a small price (under €5 for 8 virtual CPUs for 1 h [11]). As shown in Figure 1.2, lower utilisation leads to the disproportionate use of power consumption. To improve resource utilisation, intuitively, large-scale data centres like Google and Twitter wish to collocate batch and interactive workloads on the same node [40–42]. Doing so takes advantage of the idle periods in interactive applications to amortise idle power [40]. However, experimental results have shown that collocation of latency-critical and batch workloads violates Service Level Agreements (SLA) dramatically due to shared resource contention. For example, Figure 1.4 shows when a latency-critical workload is run in isolation and when collocated with a batch workload. It is clear from the graph that latency-critical workloads are sensitive to shared resource contention and violate latency even at low loads. For this reason, data centres tend to run workloads on individual servers instead of collocating. In Part III of my thesis, we address the problem of resource utilisation in data centres and show that it is possible to collocate workloads with intelligent oracles.

Transition to Heterogeneous processors. Despite power being a rarely found additional resource in today's data centres, it is often an under-optimised resource. Even with advanced hardware power techniques, commercially available processors cannot reduce the power consumption below the minimum operating power consump-

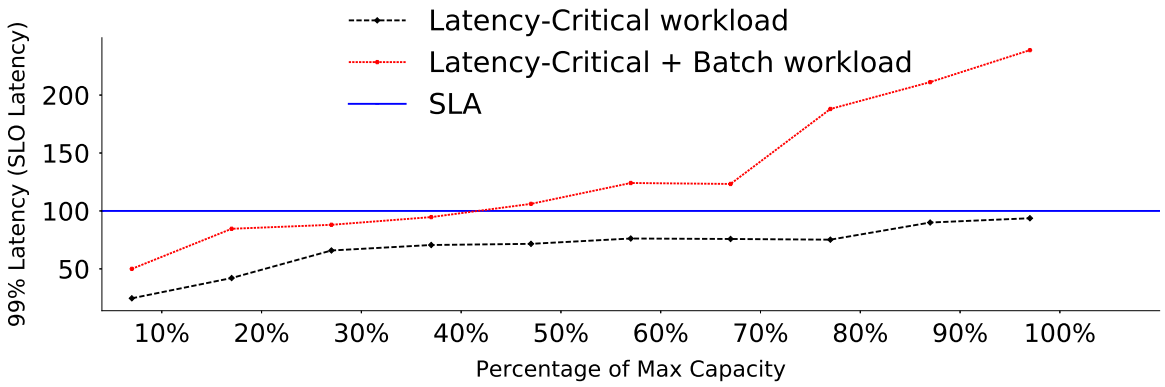


Figure 1.4 Collocation of latency-critical and batch workload. Percentage SLO for latency-critical workload (Web-Search) when run in isolation and when collocated with a batch workload.

tion [43, 44]. This has been the case for a very long time, and, as a result, the computing spectrum is moving towards *heterogeneous multicore processors* [45–48]. Heterogeneous multicore processors are a combination of high-performance and high power consumption processors, and low performance and low power processors with a single instruction set architecture (ISA).

On the one hand, the in-order processor (low performance) is interesting as it compromises performance to reduce the minimum operating power consumption during periods of low activity and improves power proportionality [49] and therefore, we deploy a heterogeneous platform in the scheduling solution presented in Part III. For this reason, these processors are gaining immense popularity [6, 50–53] in both supercomputers and data centre community. On the other hand, to build a prediction model to estimate performance and power (Part II) on a heterogeneous platform is a challenge for multiple reasons: (a) It is unclear if there exist (same) hardware counters on both out-of-order [54] and in-order [49] processors to estimate performance and power. (b) It is unclear how the performance and power varies across processor types and different hardware control settings.

Scheduling workloads. Recent work [31, 55–58] have shown that traditional power management practices and CPU utilisation measures are unsuitable to drive task management for modern data centre workloads. This is because the nature of workloads run on data centres is transitioning from batch workloads to latency-critical. Prior schemes, like OS-level dynamic voltage and frequency scaling (DVFS), work well to deliver long-term performance for batch workloads, but they can severely hurt the QoS of interactive data centre workloads. Part III of my thesis addresses the problem

of meeting SLA requirements of latency-critical workloads while improving energy efficiency.

1.2 Outline of thesis

The two most significant contributions of my research are as follows. (1) An architecture and application agnostic power and performance estimation model to control *on-the-fly* server power consumption and performance delivered. (2) A hybrid scheduling approach to manage resource allocation on heterogeneous server processors to improve resource utilisation while meeting performance guarantees. Below are details of the research questions and contributions.

Estimate and control performance and power. Large scale complex and heterogeneous data centres are often constrained to external power budget and strict performance SLA.[2] On the one hand, maintaining SLA is important as it drives the capital investment. On the other hand, not maintaining external power budgets penalises the budget. This performance and power trade-off in data centres raises the need to estimate performance and power at different hardware settings to ensure power budgets and SLA agreements are met. Prior state-of-the-art solutions addressed the need for a model on a single architecture while considering the contention for cache, memory, and numerous processors components. All these factors are essential in building a model, and prior solutions address some parts of the problem. However, we were interested in answering the question *"How do we build power and performance models that can be scalable and can integrate seamlessly in a dynamic data centre?"*

In Chapter 3, we introduce a systematic methodology to build models to predict power and performance at DVFS and idle-states at runtime for a *single core* and *multicore* processors (REPP: Runtime Estimation of Performance and Power). We suggest after extensive experimentation that accurate power and performance models can be built for modern architectures using basic hardware performance monitoring counters (PMCs) [59]. The PMCs chosen to build the models are available on all major architectures. Extensive real-machine experimentation results of batch workloads on multiple architectures results has shown that the models predict power and performance with at least 93 % accuracy.

[2] In this context, heterogeneous indicates different homogeneous processors [38] on a single server system.

Taking into account strict power restrictions data centres need to sustain, utilising all cores in idle periods is not beneficial. Therefore, data centres are shifting the trend towards consolidation workloads and shutting down cores to improve energy efficiency. Chapter 3 does not address the power consumption or performance characterisation of workloads with certain servers shutdown. Precisely, Chapter 4 extends REPP to introduce a mechanism to estimate performance and power with core consolidation, DVFS, and idle-states (Runtime Estimation of Performance and Power with Core Consolidation – REPP-C) in multicore architectures. At it's core REPP-C solves a multi-objective problem by first scheduling workloads from consolidated cores to maximise energy efficiency. Next, it models the impact of performance and power due to core shutdown.

Addressing resource utilisation. With the nature of applications running on data centres changing, collocating workloads that contend for shared resources is difficult (see Figure 1.4). This, in large-scale, translates to selecting either performance or power objective which are tightly coupled. Enabling higher power savings may imply performance SLA is violated, whereas lower utilisation implies bad power usage.

Prior approaches were based on a static oracle (based on offline profiling) for a given architecture to migrate across different configurations (a configuration may be defined as an iso-latency [40] policy to control the DVFS state [60] or cores allocated [24] and DVFS state, network bandwidth allocation [61], etc.) to satisfy performance guarantees of interactive workloads. This is a problem for two reasons. (1) Applications have different resource requirements to satisfy SLA at a given load. This, in part, is due to workloads having different characteristics. (2) With an erratic application behaviour or a sudden spike in external load, the reaction time is proportional to the number of configurations available, thereby they have a much higher potential to have a significant performance loss. These schedulers are necessary and essential to address some parts of the problem for a given application on an architecture. However, we were interested in answering the question *"How do we detect on-the-fly the best configurations for a latency-critical job on any architecture?"* In Chapter 5, we introduce a novel method based on an oracle and reinforcement learning to manage interactive and batch workloads. Our approach provides the interactive application with "just enough" resources to meet the real-time performance constraint, while the remaining resources are given to batch workloads to maximise throughput. After extensive experimentation, we argue that using only an single oracle is inefficient in today's data centres and demonstrate that our results outperform state-of-the-art schedulers.

The rest of the thesis is structured as follows: Chapter 2 provides a background on the performance and power monitoring tools, the different architectures used, their hardware capabilities and power efficiency details. Chapters 3 and 4 describe the performance and power modelling methodology and control technique for multicore architectures. Chapter 5 introduces a scheduling technique to improve resource efficiency in data centres while meeting the performance guarantees of interactive workloads. Chapter 6 discusses the related work, chapter 7 concludes the thesis, and chapter 8 describes the future work.

CHAPTER 2

Infrastructure

In recent years, multicore architectures have become the norm for server systems and are prevalent across all computing platforms/services. This implies server systems can exploit the inherent Instruction Level Parallelism (ILP) of the execution flow, which is essential to improve throughput. In addition, these processors can explore the enhanced dynamic power management (DPM) schemes/features present to enable adequate interactiveness under power constrained environments (e.g., mobile devices). But the paradox is that, in doing so, data centres need to profile server systems to understand the trade-off between performance and power, which are tightly coupled.

One of the projects commissioned by the European Union FP7 and Horizon 2020 program to understand the trade-off between performance and power in both future supercomputers and data centres is `MontBlanc` [6, 50, 62]. The goal of the project is to build a "power efficient" server system with multiple types of mobile cores and embedded cores [63]. Specifically, the MontBlanc prototype server at the supercomputing center in Barcelona (Spain), is equipped with multiple boards of Samsung Exynos 5 Dual [64], ARM Cortex-A15 [65] and ARM Mali GPU T-604 [66]. The number of clusters or servers deployed varies depending on the demand of the data centres or supercomputers. Similarly, prior research proposed [43, 44, 46, 47, 67–69] several new processors with different microarchitectural implementations, which are a direct result of different design goals and (or) shifts in technology.

The advent of such processors allowed data centre administrators to exploit different performance and power trade-off depending on the application requirements. Moreover, dynamic data centres allow more energy efficient computing services, in contrast to stand-alone homogeneous architectures [67].[1] Typically, each architecture is categorised

[1] "Dynamic data centres" are defined as data centres that continuously update and replace commodity servers to improve throughput (as in Chapter 3).

based the performance and power efficiency at numerous dynamic power management features. In this chapter, we investigate the various dynamic power management features available, the architectures used in this thesis, the performance and power monitoring tools used and finally the benchmarks on which the results of this thesis have been validated on.

2.1 Dynamic Power and Performance Management

Currently available modern operating systems control the power dissipated by dynamically controlling the power state in which the processor is at runtime. Most operating systems, provide an interface to monitor the processor's power state to communicate and control with the hardware subsystem. For instance, on Linux, Advanced Configuration and Power Interface (ACPI) defines a protocol to control these power states. In this section, we give a background of the hardware settings used in this thesis.

Dynamic Power Management (DPM). DPM schemes are ingrained as an important means to reduce power consumption in data centres having hundreds or thousands of cores, each of them with multiple power management features. We focus on the primary DPM features available our architectures: processor performance states (DVFS states), processor power states (C-States), clamp states (Cl-States) and core consolidation. DVFS states [17, 39, 70–74] and Cl-States [75] have been used in numerous previous works to provide fine-grained low-level support for a reasonable performance-power trade-off.

Processor performance states (P-States/DVFS). Each DVFS state represents a specified voltage and clock frequency of the cores [76] and is controlled by the ACPI. The frequency and voltage for each state is described in the Differentiated System Description Table (DSDT). The higher numbered DVFS states correspond to higher clock frequency and higher voltage, thereby delivering a higher performance subjected to the scalability of the workload.

The changes in DVFS state are governed using the ACPI CPUFreq governor using the acpi_cpufreq driver. The default governor available using the driver as mentioned above is ondemand [77], which allows the kernel to control the core's DVFS state based on the "load" on the processor. However, the governor *userspace* lets users to control the core's DVFS state, via the sysfs file system, manually irrespective of the "load".

Processor power states (C-States). Modern multicore processors support multiple C-States, also known as "Sleep States". These states allow various internal modules to be turned off. Higher-numbered C-States indicate a "deeper" sleep state with more internal modules disabled resulting in a lower power consumption, but entails a longer wake up period (up to 1024 ns on an Intel machine). At a given C-State, all DVFS states have same power consumption when idle [75]. The choice of C-States is architecture and processor dependent. For example, Intel Sandy Bridge supports C0, C1, C3, C6 and C7 states and they are accessible through the sysfs file system, similar to the DVFS states.

This technology on Intel and AMD processors is coined as "C-States" [78, 79], and on ARM processors as "CPUidleness" [80]. For instance, most Intel processors support C0, C1, C3, C6 and C7 states. The residency period indicates the duration of time a given core is present at a given C-State and can be controlled using the model-specific registers (MSR). The default time quantum for *entering* a C-State is 1024 ns.

Clamp States (Cl-States). Although C-States are a effective power reduction mechanisms, they are used to reduce power consumption only when the system is idle. On the other hand, *Cl-States* introduce synchronised idle injections across online CPU threads to enable forced and controllable C-State residency. The changes in Cl-States are governed by ACPI using the intel_powerclamp driver [81].

2.2 Architectures

In this section, we present the architectures and processors our contributions have been evaluated. We do not enable any other internal thermal/power management algorithms which are run by the firmware on any processor.

Intel. Intel Corei7 Sandy Bridge processor [78] has four out-of-order cores enabled, each with 64 KB on-chip private L1 cache and 256 KB private L2 cache. The total shared LLC was 6 MB, and the DRAM capacity of 8 GB with Linux (kernel 3.14.5). The processor is capable of DVFS from 0.8 GHz to 2.4 GHz. Turbo boost and Hyper-Threading were disabled. The number of Cl-States in this study are 50 per core.

AMD. AMD Phenom II [82] processor has four out-of-order cores enabled, each with 64 KB on-chip private L1 cache and 512 KB private L2 cache. The total shared LLC

was 6 MB, and the DRAM capacity of 8 GB with Linux (kernel 3.13). The processor is capable of DVFS from 0.8 GHz to 3.2 GHz. AMD processor has no Cl-States.

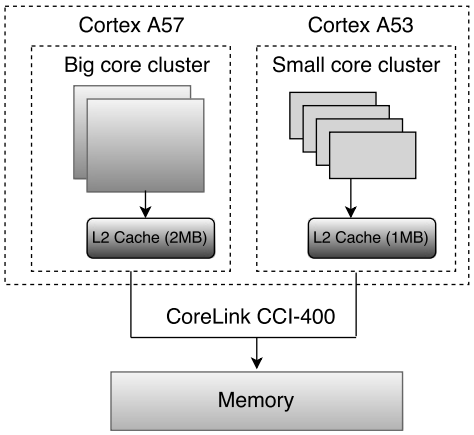


Figure 2.1 Heterogeneous processor platform (ARM Juno R1)

ARM. ARM Juno R1 developer board [83] has Linux (kernel 4.3). The Juno board is a 64-bit ARMv8 big.LITTLE architecture with two high-performance out-of-order Cortex-A57 (big) cores and four low-power in-order Cortex-A53 (small) cores. The cores are integrated on a single chip with off-chip 8 GB DRAM. The two big cores form a cluster with a shared 2 MB L2 cache, and the four small cores form another cluster with a shared 1 MB L2 cache. The big cores are capable of DVFS from 0.6 GHz up to 1.15 GHz, whereas the small cores are fixed at 0.65 GHz. The architecture is schematically shown in Figure 2.1. A cache coherent interconnect (CoreLink CCI-400) provides full cache coherency among the heterogeneous cores, allowing a shared memory application to run on both clusters simultaneously. ARM processor has no Cl-States.

X-Gene2. AppliedMicro X-Gene2 [84] board has Linux (kernel 4.1). The X-Gene2 is a 64-bit homogeneous architecture with eight out-of-order ARMv8-A cores enabled at 2.4 GHz with 128 GB DRAM.

In what follows, I refer to Intel Sandybridge as Intel, AMD Phenom II as AMD, and ARM Juno R1 as ARM.

2.3 Performance Monitoring

We use the performance monitoring tool, perf [85], on all architectures to gather *perf_events*. Alternatives to perf include the profiling tools (like Google wide profiler (GWP) [86, 87], perfmon [88], etc.,) supported by Docker, Kubernetes and LXC [89].

The ARM Juno development board raises two challenges while gathering PMCs: (1) It does not provide counters other than instructions, cache_misses, and cycles to be read for Cortex A53, the in-order processor, as the CPTR_EL3 (Architectural Feature Trap Register) is not implemented in the Juno chip. Therefore, the Cortex-A53 is used only to gather basic performance statistics when throughput-oriented workloads are collocated with interactive workloads. (2) There is a known bug [80] that causes perf to generate garbage values for all cores whenever any core enters an idle state. Since performance statistics are only needed when throughput-oriented workloads are run, we overcome this by disabling CPUidle [90, 91]. This prevents Linux from entering the cores in an idle state when changes in the mapping cause idle periods longer than 3500 µs.

The characterisation of workload is achieved by periodically collecting performance statistics from the latency-critical and batch workloads. For the latency-critical workload, we gather the appropriate application-level QoS metrics such as throughput (Requests Per Second – RPS or Queries Per Second – QPS) and latency (query tail latency). For the batch workload, we use perf to characterise the thread behaviour using the perf per thread session.

2.4 Power Meters

To gather power measurements, we use the machine's power sensor (in case of Intel and ARM) or an external power meter (in case of AMD as power sensors are not available) periodically during execution. The X-Gene2 board is equipped with neither a power sensor nor attached to a power meter as it is used only as a client side request generator for latency-critical workloads.

The power meters used are as follows. (1) The *RAPL* (Running Average Power Limit) register in the Intel architecture records power of core, uncore, and DRAM controller [73]. The RAPL interface is accessed using the MSR available on Intel processors [92]. (2) *WattsUP pro* for AMD: external device that records power of the entire system (power from outlet) [93]. (3) *Four native energy meters* for ARM [94, 95]. These registers report separately the power consumed by the big cluster, small cluster, and the rest

Table 2.1 Machines used in this dissertation

Processor	Linux Kernel	Power Meter	gcc	DFS (in GHz)	L2 (size in kB)	L3 (size in MB)
Intel Core i7-2760QM (4 cores)	3.14.5	RAPL	4.8.1	0.8-2.4	256	6
AMD Phenom II X4 B97 (4 cores)	3.13.0	WattsUp Pro	4.9.2	0.8-3.2	512	6
ARM Juno 64-bit	4.3.0	Native-	4.9.2			no L3
2 Cortex A57		energy-		0.6-1.15	2048	
4 Cortex A53		meter		0.65	1024	
Applied X-Gene2 64-bit (8 cores)	4.10	no meter	5.2.0	2.4	256	8

of the system (Juno's *sys* register [96]). The power consumption of the Mali GPU is also available, but it is negligible because the GPU is disabled in all our experiments.

Table 2.1 summarises the architecture, processor, Linux kernel version, the power meters, the compiler (gcc) version used for compiling the benchmarks, the range of core frequencies (min-max) when using DVFS, the L2 and L3 cache sizes (ARM does not have L3). The machines were available across multiple institutions (Barcelona Supercomputing Center, University of Pittsburgh, and Universidade Federal da Bahia), and therefore we did not have control over the kernel and gcc versions; regardless, the results for each architecture are consistent within that set of experiments.

2.5 Power Efficiency

Over the last decade, the number of services provided by data centres is increasing, and this led to the rapid rise in the number of computing nodes. This increase in the computational nodes raises the need to understand processors' power efficiency to maintain and deliver the performance SLA while constrained to a strict power budget. Moreover, data centres spend about 30 % of power allocated towards system cooling and thereby improving the processor life time. Recent studies [97–100] have shown marginal rise in temperature (80°c) is reported to reduce battery life by 50 %. Similarly, a 10°c increase in a capacitor component temperature will reduce its life by one half [97].

For instance large-scale data centres like Facebook combat this problem in two ways: (1) Reducing the system cooling cost by moving to cooler climates (Luleå, Sweden),

Table 2.2 Power and performance characterisation per architecture. Power and performance measured using a microbenchmark on each architecture

	$Power\ (Watts)$		$Perf.(IPS \times 10^6)$	
Processor	All cores	One core	All cores	One core
Intel				
Max frequency (2.4 GHz)	27.59	11.67	19,414	4,822
Min frequency (0.8 GHz)	11.08	5.76	7,993	1,627
AMD				
Max frequency (3.2 GHz)	141.8	81.4	27,248	6,812
Min frequency (0.8 GHz)	52.8	48.7	6,786	1,696
ARM				
Big A57 (1.15 GHz)	2.30	1.62	4,260	2,138
Small A53 (0.65 GHz)	1.43	0.95	3,298	826

thereby improving the power efficiency [101].[2] (2) Increasing the total available power to cool the systems by using alternate sources of fuel [102].

We quantify the power efficiency of our processors using the power meters described in Section 2.4 and the performance using the IPS measured using hardware performance counters. Table 2.2 summarises the power and performance characteristics per architecture. We characterise each platform by running a stress microbenchmark consisting of only mathematical operations without memory accesses. For a per cluster comparison, we run the microbenchmark on all the cores in the cluster. For a per core comparison, we run the microbenchmark on a single-core. Specific architectural details are as follows:

Intel platform. We characterise the homogeneous platform (four out-of-order cores at two different frequencies). We report the power consumption obtained from the Intel RAPL register as the sum of core, uncore and DRAM controller. To determine the per cluster efficiency, we run the microbenchmark on four cores at the highest and lowest frequencies concurrently. For the per core efficiency, we run the microbenchmark on a single-core at the highest and lowest frequencies concurrently.

Considering the system power, a single-core at the highest DVFS state is 0.6× less power efficient than all cores at the minimum DVFS state, in terms of IPS per watt. This is because, there is a minimal power consumption for the unused cores when idle and without considering the power consumption of the idle cores, the power efficiency is 1.3×. However, taking that all cores in a cluster, and assuming that all cores are

[2] According to Facebook, in 2015, Luleå warehouse was the most energy efficient data centre ever built.

fully utilised, four cores at the highest DVFS state is approximately (0.97×) as power efficient as four cores at the lowest DVFS state.

AMD platform. We characterise the homogeneous platform (four out-of-order cores at two different frequencies). We report the socket power consumption obtained from the WattsUp Pro meter. To determine the cluster efficiency, we run the microbenchmark on four cores at the highest and lowest frequency concurrently. For the per core efficiency, we run the microbenchmark on a single-core at the highest and lowest frequency concurrently.

Considering the system power, a single-core at the highest DVFS state is 0.6× less power efficient than all cores at the minimum frequency, in terms of IPS per watt. This is because, the AMD processor does not implement C-States (or alike) to save power when the core is idle, and therefore the single-core is not as power efficient as it could have been. However, taking that all cores in a cluster, and assuming that all cores are fully utilised, four cores at the highest DVFS state are 1.5× more power efficient than four cores at the lowest DVFS state.

ARM platform. We characterise the heterogeneous platform (two big cores as big cluster, and four small cores as small cluster). We report the system power consumption as the sum of the big and small clusters and the rest of the system (including memory controllers, etc). For the per-cluster comparison, we run the microbenchmark on the big cluster and the small cluster concurrently. For a per core comparison, we run the microbenchmark either on a single big core or on a single small core.

Considering system power, a single big core is 52 % more power efficient than a single small core, in terms of IPS per watt. However, taking into account all cores in a cluster, and assuming that all cores can be fully utilised, a small cluster is 25 % more power efficient than a big cluster. This discrepancy is due to the rest of the system, outside the core clusters, which consumes about the same power as a big core at full utilisation (0.76 W). If we subtract the power of the rest of the system, a single small core is 2.3× more power efficient than a big core. We notice that small cores are attractive for throughput-oriented workloads, because of improved power efficiency. Big cores are, however, still needed for latency-critical workloads with tight QoS constraints, as a result of computationally-intensive single-threaded requests.

2.6 Workloads

In recent years, data centres [6, 22, 24, 28, 40, 50, 60, 62, 103] have seen an influx in a wide range of workloads which stress different microarchitectural components. This puts forth the need to evaluate an algorithm for different types of workload.

We evaluated our contributions, REPP and Hipster, using throughput-oriented/batch workloads [38, 58] and interactive/latency-critical workloads [22, 24, 28, 40, 103, 60]. These classes of workload are interesting due to the nature of the problem they exhibit and their interactions with the hardware. This thesis does not explore deployment of large-scale distributed workloads, and this is left out as future work.

2.6.1 Batch Workloads

We categorise the sequential batch workloads into single threaded batch workloads and multiprogrammed batch workloads. For the single threaded batch workloads, we ran 22 SPEC CPU 2006 [104, 105], nine PARSEC 3.0 [106], six NAS [107], 11 SPLASH2x [108]. These workloads exhibit both memory and computational bound phases. The number of benchmarks in a multiprogrammed workload, when evaluating REPP, is equal to the number of cores in each architecture, thereby, having 35 workloads of four benchmarks on AMD and Intel, and ten workloads of two benchmarks on ARM. When evaluating REPP, on the ARM processor, we use only the big cluster as the small cluster does not allow us to gather most of the counters. The methodology to build the workloads is described by Sanchez and Kozyrakis [109].

The benchmarks are divided into four categories, following the categorisation by Sanchez and Kozyrakis [109]. We run all the applications in isolation and gather PMCs at a sampling frequency of 250 ms until the end, and compute mean value of the ratio of number of kilo-IPS to kilo-LLC misses. For all workloads, we select the native input set size. Those which have ratio less 0.1 are considered as Thrashing or Streaming (**S**), applications that benefit cache size, that is, ratio between 0.1 and 0.2 are classified as Cache-Fitting (**T**). Then, applications with a ratio between than 0.2 and 1 are classified as Cache-Friendly (**F**). Finally, applications with ratio greater than 1 are classified as Insensitive (**N**). There are 35 possible combinations (with repetitions) of these four categories, each of which forms a group. In the multiprogrammed workloads consisting of four benchmarks (on Intel and AMD) from SPEC, PARSEC 3.0, NAS, and SPLASH2x suites, we have one mix per group. In the multiprogrammed workloads, each application is randomly selected from the category. This results in 35 workloads. The same methodology is followed on ARM, where there are 10 possible combinations

(with repetitions) of these four categories, each of which forms a group. In those multiprogrammed workloads consisting of two benchmarks from PARSEC 3.0, NAS, and SPLASH2x, we have one mix per group. Note that SPEC could not be compiled on ARM. Table 2.4 shows the categorisation of the workloads. This technique of selecting multiprogrammed workloads facilitates for significant difference in the size of the shared memory footprint and number of stall cycles of the processor. Tables 1.1 and 1.2 show the multiprogrammed workloads on ARM, AMD and Intel.

The threads in a multiprogrammed workload run continuously until the longest thread in execution finishes (other threads restart execution immediately as they finish). To ensure statistically significant results, we ran each workload multiple times and had a 95 % confidence with a very low error margin (less than 2 %). In the graphs that follow, we do not show the error margin to avoid visual clutter.

2.6.2 Latency-Critical Workloads

We evaluate the effectiveness of Hipster using two latency-critical workloads, Memcached [110], and Web-Search [111], which have distinct characteristics and impact on shared resources [40]. **Memcached** [110] is an open source implementation of an in-memory key-value store for data caching used in many services from Twitter, Facebook and Google [112–114]. The backend of **Web-Search** [27, 45, 112] is an instance of Elasticsearch [111], an open source implementation of a search engine used by many companies including Netflix and Facebook. The load generator (Faban [115]) for Memcached and Web-Search is adapted from CloudSuite 3.0 [116]. It is configured to model diurnal load changes, simulating a period of 36 hours [28]; each hour in the original workload corresponds to one minute in our experiments.

We also evaluate the effectiveness of Hipster by collocating a single interactive workload, and a mix of batch workloads. The objective of the collocation is to maximise the throughput of the batch workloads while satisfying QoS of the interactive workload. The number of batch workloads is equal to the number of cores not utilised by the interactive workload. We report the system throughput by aggregating the IPS of all batch programs.

When implementing Hipster, the ARM processor is used when collocating a latency-critical workload with a batch workload, and running a latency-critical workload in isolation; The workload generator runs on another machine: XGene2.

Assessment metric for latency-critical applications. Quality of Service for social-networking sites like Twitter and Google require fast rendering of user-content

Table 2.3 Latency-critical workload configuration. Workload configurations, maximum load while meeting the target tail latency with two big cores.

App	Workload Configuration	Max. Load	Target Tail latency (ms)
Memcached	Twitter caching server of 1.3 GB	36 000 RPS	10 (95%ile)
Web-Search	English Wikipedia Zipfian distribution	44 QPS | think-time of 2sec[115]	500 (90%ile)

with strict SLA requirements such as requests per second. In addition, to statistically compute the average latency, QoS often focuses on tail latency which is defined as the upper bound of latencies experienced by 95^{th} or 99^{th} percentile of flows (for example, TCP flows, HTTP requests, RPCs, etc.) to improve interactivity [41].

Tail latency requirements. For Memcached, we define the tail latency to be the 95^{th} percentile request latency, with a target of 10 ms [103, 116]; for Web-Search, we define it to be the 90^{th} percentile query latency, with a target of 500 ms [116]. The load for Memcached and Web-Search are measured in terms RPS and QPS, respectively. Table 2.3 lists the two latency-critical applications, their configurations, maximum loads, and target tail latency in milliseconds. For each latency-critical workload, the maximum load is chosen so that the platform is able to meet the tail latency when running on the big cores at maximum DVFS.

2.7 Conclusion

In this chapter, we have introduced the architectures, the power meters, the performance monitoring tools, the type of workloads our results have been evaluated on. In a nutshell, the work has been carried out on three 64-bit architectures: Intel SandyBridge, AMD Phenom II, and ARM Juno. Each of these architectures is equipped with a power meter to gather power statistics. Similarly, we install a performance monitoring tool to gather performance statistics per application. Finally, the single threaded workloads are taken from four different benchmark suites: SPEC CPU 2006, PARSEC, SPLASH, and NAS. The multiprogrammed workloads were designed based on the methodology described by Sanchez and Kozyrakis [109].

Table 2.4 Categorisation of workloads based on methodology described in [109]. The label, and description for each benchmark from SPEC, NAS, PARSEC, and SPLASH2x. The labels are read as follows: N as Insensitive, F as Cache-Friendly, T as Cache-Fitting, and S as Streaming/Thrashing

		Platform		
Benchmark	*Suite*	*Intel*	*AMD*	*ARM*
Bzip2	SPEC	N	N	-
Calculix	SPEC	N	N	-
Gobmk	SPEC	N	N	-
Gromacs	SPEC	N	N	-
H264ref	SPEC	N	N	-
Hmmer	SPEC	N	N	-
Namd	SPEC	N	N	-
Omnetpp	SPEC	N	N	-
Povray	SPEC	N	N	-
Tonto	SPEC	N	N	-
CactusADM	SPEC	F	F	-
Lbm	SPEC	F	S	-
Libquantum	SPEC	F	S	-
Sjeng	SPEC	F	N	-
Zeusmp	SPEC	F	F	-
Wrf	SPEC	F	F	-
Astar	SPEC	T	T	-
Bwaves	SPEC	T	T	-
Soplex	SPEC	T	T	-
GemsFDTD	SPEC	T	S	-
Mcf	SPEC	S	S	-
Milc	SPEC	S	S	-
Xalancbmk	SPEC	S	S	-
Radiosity	SPLASH2x	N	N	F
Raytrace	SPLASH2x	N	N	T
Water_spatial	SPLASH2x	F	N	T

		Platform		
Benchmark	*Suite*	*Intel*	*AMD*	*ARM*
Barnes	SPLASH2x	F	T	T
Fft	SPLASH2x	N	N	N
Fmm	SPLASH2x	F	F	S
lu_cb	SPLASH2x	N	N	N
lu_ncb	SPLASH2x	F	F	N
ocean_cp	SPLASH2x	F	F	T
Radix	SPLASH2x	T	T	T
Volrend	SPLASH2x	N	N	F
Water_nsquared	SPLASH2x	N	N	F
Blackscholes	PARSEC	N	N	N
Bodytrack	PARSEC	N	F	S
Dedup	PARSEC	N	T	S
Canneal	PARSEC	T	F	T
Facesim	PARSEC	N	N	F
Fluidanimate	PARSEC	N	N	N
Freqmine	PARSEC	N	N	N
Raytrace	PARSEC	N	N	T
Streamcluster	PARSEC	S	S	F
Vips	PARSEC	N	F	T
x264	PARSEC	F	F	T
ep.C	NAS	N	N	N
is.C	NAS	N	N	N
cg.C	NAS	F	F	S
lu.C	NAS	S	S	S
mg.C	NAS	S	S	S
sp.C	NAS	S	S	S

This page is intentionally left blank.

Part II:

Modelling and Prediction Technique

Survival of the Fittest.

This quote is often associated with Charles Darwin as he gave us the evolutionary theory, but it was Herbert Spencer who phrased it first.

THIS part dives into the topic of power and performance management in data centre environments. Modern and dynamic data centres continuously update and replace commodity servers to deliver a higher throughput (performance) or to reduce the power consumption or to improve the Power Usage Effectiveness (PUE) [97, 117]. To ensure these constraints are met, administrators need to understand how the applications' phases, external perturbation, current hardware settings, etc., impact the (single thread) performance and power. However, operating system scheduling algorithms on multicore systems, are often targeted to maximising performance based on average CPU utilisation in a sampling interval (typically in the order of 10^{-6} s) while neglecting the system power consumption and the scalability of the application [77]. This is a shortcoming with the OS scheduling strategy, as applications that are memory bounded, do not benefit from the high compute performance of a high DVFS state as they stall for memory, thus using a lower DVFS state might be sufficient to satisfy the performance constraint while saving power. At the same time, the converse is true for compute bounded workloads.

In addition, current server systems are intrinsically power inefficient at low utilisation [24, 28, 40, 103, 118] and therefore major high-performance computing (HPC) vendors (like Amazon EC2 [11] and IBM Soft Layer [119]) offer high-performance computing at a small fraction of the server costs. This helps vendors to maximise utilisation, improve power efficiency and drive capital investment. While "renting" servers maximises utilisation and improves power efficiency, it is challenging to estimate server power and performance because (a) Server vendors are agnostic to the users applications, thereby it is challenging to profile each application offline to meet the power constraints and performance guarantees. (b) Since each user has different constraints, it is necessary to understand applications' power and performance at runtime and select hardware settings that satisfy each users requirement while minimising the server power consumption. In such scenarios, to improve energy efficiency at a low cost, modern data centre architectures incidentally or intentionally, have to deal with server architecture heterogeneity [38, 67, 120] i.e., different homogeneous processors on a single server system. With server architecture heterogeneity come some exciting challenges: (1) How to map applications across multiple architectures? (2) How to provide a service that is reliable to meet constraints or requirements? (3) How to provide a computing service on demand? (4) How to deliver computing service at an attractive cost to end-users?

Given such interesting challenges in current data centres, a service that can be provided using native hardware capabilities across architectures – making a service uniform – may be represented in various forms. For instance, minimising energy

consumption subject to a performance target and peak power constraint, or maximising performance subject to a peak power constraint or to provide energy efficient cluster management in data centres. Irrespective of the how the service is formulated, any solution, whether optimal or heuristic-based, requires a fast and accurate model to predict how a potential change in a DPM feature [76] or other low-level power features will translate into thread performance and power demand.

A modelling solution with currently available DPM features on modern operating systems makes it difficult to understand the combinatorial complexity they bring regarding service comprehension. Generally, administrators of large-scale data centres deal with metrics such as performance (IPS), power and efficiency levels (e.g., performance per watt or PUE), which do not directly correspond to DVFS state, Cl-States or core shutdown. With an increase in the number of DPM features, a unique combination of these features generates a different power and performance level, thereby making the combinatorial complexity of these parameters polynomial.

This complexity in hardware explains why current solutions (like Intel Intelligent Power Node Manager [121], the OpenStack framework [122] or the IBM Tivoli framework [119]) operate at a coarse granularity, at the node level, while the hardware features work at a core level, with low overheads. Understanding the behaviour at per core or per thread level allows for precise control over performance and power contribution and their corresponding effects. What is necessary is an application agnostic prediction approach for heterogeneous data centres to meet a constraint, irrespective of how it is formulated.

Traditionally such complexity has been met with iterative algorithms which are DVFS state based: they monitor the application behaviour history periodically [32] and set the DVFS configuration for the next quantum. However, these approaches require multiple iterations before power and performance criteria are satisfied and can incur massive violations in power and performance budgets for data centres [67, 123].

In contrast to traditional iterative algorithms, what is crucial is a fast and reactive prediction based algorithm that can provide a quick response and reaction for a small fraction of the computational costs. Reactive based algorithms determine phase changes in behaviour, i.e., from compute-bound to memory-bound and vice-versa, using existing PMCs to select the appropriate DVFS state based on the external constraints.

Using PMCs, we demonstrate that our prediction algorithm can determine if the application is scalable with a combination of DPM features at runtime and can select the most appropriate configuration given a constraint.

The remainder of Part II is organised as follows.

Chapter 3 introduces *Runtime Estimation of Performance and Power*, **REPP**, a methodology to build *models* and estimate performance and power for a *single-core* [124] and *multicore* processors [125] parametrised by DVFS states and Cl-States. REPP is a method to generate fast, and accurate performance and power predictions. Our real-machine experimental evaluation shows that the models are accurate enough to capture the workload behaviour and are driven by existing performance counters. Since the computational complexity at runtime is low, it can be used for fine-grain power and performance management.

Chapter 4 extends REPP to predict power and performance with core consolidation (**REPP-C**). REPP-C (*Runtime Estimation of Performance and Power with core consolidation*) [126] estimates and controls power and performance on server architecture with core consolidation.

The evaluation of REPP and REPP-C, on each architecture, was carried out with numerous single threaded and multiprogrammed applications. We demonstrate the effectiveness of the modelling technique by ensuring that the power constraints and performance guarantees are satisfied with multiple constraints on three different architectures.

CHAPTER 3

Multicore Power and Performance Modelling

In this chapter, we introduce the single-core and multicore modelling methodology and prediction technique (REPP), and evaluate on three major architectures: Intel, ARM and AMD. REPP is modelled using multi linear regression models constructed in two steps: *offline* modelling and *online* modelling.

Offline modelling. In the offline modelling technique, a subset of applications from representative benchmarks suites are profiled to build power models and performance models offline for a single-core. The models built can predict power and performance at the available DVFS states and Cl-States with high accuracy. The offline design phase is attractive because (1) it eliminates the overhead of learning and tuning the performance and power model at various DVFS states and Cl-States at runtime; (2) it does not rely on using power sensors/meters to estimate power and performance at runtime; (3) it is a one-time effort.

Online modelling. In the online modelling technique, we extend our models to the multicore environment, by *aggregating* the per core contributions. The models when exposed to a multicore system need to tackle with shared resource contention, which is modelled using simple multi linear regression technique. Finally, we describe a technique for power and performance control across the multicore environment.

The remainder of this chapter is structured as follows. Sections 3.1 and 3.2 describe the design of single-core and multicore modelling technique, respectively. Sections 3.3.2 and 3.3.3 focus on the offline and online validation of the single-core models. Section 3.4.1 evaluates REPP without considering the shared resource contention. Sections 3.4.2 and 3.4.3 demonstrate the results of the multicore modelling technique without and with external user specified constraints. Section 3.5 concludes the chapter. The notations used hence-forth for the rest of the chapter are detailed in Table 3.1.

Table 3.1 Summary and description of the symbolic notations

Notation	Description
P_c	Current DVFS state.
P_i	Intermediate DVFS state.
P_f	Future DVFS state.
P_c	Current frequency for DVFS state.
P_i	Intermediate frequency for DVFS state.
P_f	Future frequency for DVFS state.
Cl_c	Current Cl-State.
Cl_i	Intermediate Cl-State.
Cl_f	Future Cl-State.
α_c	Current configuration, that is, (P_c, Cl_c).
α_f	Future configuration, that is, (P_f, Cl_f).
η_c	Performance (IPS) at current configuration.
η_f	Performance (IPS) at future configuration.
ρ_c	Power at current configuration.
ρ_f	Power at future configuration.

3.1 Single-Core Offline Modelling

We predict performance and power in a different DVFS state and Cl-State, based on the activity recorded at the current DVFS state and Cl-State in the microarchitectural components floating point units (FP), integer units (INT), front end (FE), branch predictor unit (BPU), private L1 cache (L1), private L2 cache (L2), last level cache (LLC), and the memory sub-system (MEM). Since these components do not have PMCs that directly record their activity, we use the available PMCs to calculate each component's **Activity Ratio** (AR). The activity ratio is defined as the component's average number of µops (micro-ops) per cycle (µops/cycle).

Table 3.2 summarises the microarchitectural components and modelled components on the Intel processor [78]. Similar components are modelled on AMD [82] and ARM [83]. Table 3.3 summarises the microarchitectural components, activity formulas for Intel, AMD, and ARM; and their respective raw `perf` event registers [78, 82, 83]. To compute the activity ratio, the activity in each component, for each architecture, is divided by `cpu_clk_unhalted (r076)` on Intel, `cpu_cycles (cycles)` on ARM and `cpu_clk_unhalted:0x01 (r13c)` on AMD, respectively.

In building the multi linear regression models, we specifically profile the activity in the aforementioned microarchitectural components because our results using microbenchmarks, on each architecture, have shown that these microarchitectural components have a high dynamic power. We define dynamic power as the difference be-

Table 3.2 Component definitions for Intel Core i7

Component	Modelled components
FE (IPC)	L1_ITLB, L1_ICACHE, PREDECODE, FETCH_UNIT, µCODE ROM, µOP BUFFER, LSD, SPT, RAT, ROB, RETIRE
INT	Integer arithmetic units
FP	Floating point arithmetic units
BPU	BPU and branch execution
L1	LD/ST execution, MOB, L1, L1 DTLB, L2 DTLB
L2	L2
LLC	LLC
MEM	Memory and Front Side Bus (FSB)
IPS	Instructions per second
LLC misses	Last level cache misses

tween the current power consumption and power consumption when idle. The activity formulas were built using carefully selected PMCs that are highly correlated with the dynamic power and performance. For instance, on the Intel platforms' pipeline functionality, the scheduler which is a part of the out-of-order engine has six issue ports, out of which ports 0, 1 and 5 are shared for integer, branch instructions and floating point instructions. However, there are counters for each port, branch instructions and floating point instructions. Therefore, we subtract the instructions issued using ports 0, 1 and 5 and the branch instructions and number of floating point instructions to get the total number of integer instructions. By contrast, on AMD and ARM, the microarchitectural components have unique counters for a total number of integer instructions.

Table 3.3 Formula to compute activity ratio. Events and *perf* raw event numbers used on Intel (top), AMD (middle), ARM (bottom) machines

	Component	Intel	Intel (*perf* raw event)
INTEL	FE	UOPS_RETIRED:ANY	R1C2
	INT	(UOPS_DISPATCHED_PORT:(PORT_0 + PORT_1 + PORT_5) - FP_COMP_OPS_EXE:X87 - BR_INST_RETIRED)	R1A1 + R2A1 + R80A1 - R110 - R0C4
	FP	FP_COMP_OPS_EXE:X87	R110
	BPU	BR_INST_EXECUTED	RFF88
	L1	PERF_COUNT_HW_CACHE_L1D	-
	L2	L2_RQSTS:0XC0	RC024
	L3	LAST_LEVEL_CACHE_REFERENCES	R4F2E
	MEM	PERF_COUNT_HW_BUS_CYCLES	BUS-CYCLES
	IPS	INSTRUCTIONS	INSTRUCTIONS
	L3 MISSES	CACHE-MISSES	CACHE-MISSES

	Component	AMD	AMD (*perf* raw event)
AMD	FE	RETIRED_UOPS	R5000C1:U
	INT	DISPATCH_STALL_FOR_INT_SCHED_QUEUE_FULL	R0D6
	FP	RETIRED_MMX_AND_FP_INSTRUCTIONS:X87	R5001CB:U
	BPU	BRANCH_RETIRED	R5000C3:U
	L1	PERF_COUNT_HW_CACHE_L1D	-
	L2	REQUESTS_TO_L2	R037D
	L3	PERF_COUNT_HW_CACHE_REFERENCES	R080
	MEM	PERF_COUNT_HW_BUS_CYCLES	R0062
	IPS	INSTRUCTIONS	INSTRUCTIONS
	L3 MISSES	CACHE-MISSES	CACHE-MISSES

	Component	ARM	ARM (*perf* raw event)
ARM	FE	-	-
	INT	INST_SPEC_EXEC_SIMD	R074
	FP	INST_SPEC_EXEC_VFP	R075
	BPU	INST_SPEC_EXEC_SOFT_PC	BRANCHES
	L1	L1D_CACHE	R04
	L2	L2D_CACHE	R16
	L3	NO L3	NO L3
	MEM	PERF_COUNT_HW_BUS_CYCLES	BUS-CYCLES
	IPS	INSTRUCTIONS	INSTRUCTIONS
	L3 MISSES	CACHE-MISSES	CACHE-MISSES

3.1.1 Modelling Power

Power can be modelled based on the PMCs [127–132]. Multi linear regression models allow for power prediction with high accuracy (less than 5 % error rate).

Predicting power in the current configuration. We build on the technique proposed by Bertran *et al.* [133] to predict power at the current configuration (ρ_c), and is computed based on the activity recorded in FP, FE, BPU, L1 cache, L2 cache, LLC cache and MEM.

$$\rho_c = \sum_{i=0}^{comps} (\Delta_i \times AR_i) + constant \tag{3.1}$$

Equation 3.1 represents the multi linear regression model for predicting ρ_c, where Δ_i and *constant* represents the coefficients to be learned and AR_i represents the activity ratio of the individual components. We build one model for each available DVFS state with Cl-State zero.

Predicting power across DVFS states. Intuitively, one of the biggest challenges to predict performance or power across DVFS states, while keeping the Cl-States fixed, is the need to interpolate the component activity ratios with the higher or lower DVFS state. Contrary to our intuition, our findings show that the changes in activity ratio of the microarchitectural components are minimal.

For example, Table 3.4 shows the average activity ratio at the minimum and maximum DVFS state for the microarchitectural components INT, LLC, and FP for 16 SPEC benchmarks [104] for the first 100 million instructions of execution. We represent the data into parts as the activity ratio in FP varies in three orders of magnitude depending on the benchmark type (integer benchmark or floating point benchmark), with a few exceptions. We focus on the memory intensive floating point benchmark *Lbm* and a visual representation of the activity in FP, INT, LLC and FE at each DVFS state are shown in Figure 3.1. Observe, albeit having a 67.4 % change in activity ratio for FP from 0.8 GHz to 2.4 GHz (Figure 3.1a), the absolute change in activity is negligible. To ensure the statistical significance of this finding, we ran the workloads multiple times and had a 95 % confidence with a low error margin (less than 2 %). This behaviour is consistently observed across any benchmark for any microarchitectural component, as it is a microarchitectural property [39].

On the contrary, having negligible absolute difference in activity ratio across DVFS states does not imply that the power across DVFS states for a given Cl-State is constant,

Table 3.4 Microarchitectural component activity range for SPEC CPU 2006. The activity computed over 100 million instructions at minimum and maximum DVFS states

Benchmark	INT/FP	INT	LLC ($\times 10^{-2}$)	**FP** ($\times 10^{-1}$)
Xalancbmk	INT	0.6835-0.5124	2.1426-1.7193	0.0021-0.0012
Calculix	FP	1.7852-1.8618	0.2043-0.2074	0.0139-0.0127
GemsFDTD	FP	1.3728-1.2165	2.9210-2.6255	0.0156-0.0168
Wrf	FP	1.0714-1.0526	0.3263-0.2859	0.1673-0.0941
Tonto	FP	1.2196-1.2258	0.1295-0.4179	0.0957-0.2169
Povray	FP	1.0132-1.0290	0.0808-0.0764	0.4823-0.4958

Benchmark	INT/FP	INT	LLC ($\times 10^{-2}$)	**FP** ($\times 10^{-5}$)
CactusADM	FP	1.0972-0.8945	0.5738-0.4455	1.6002-0.5251
Lbm	FP	1.0231-1.0199	1.1000-1.7063	1.6121-0.5255
Hmmer	INT	1.3572-1.3945	0.1099-0.0862	3.1287-1.6042
H264ref	INT	1.2756-1.3068	0.2752-0.2565	0.0344-0.0353
Gobmk	INT	0.6718-0.6745	0.3105-0.2924	0.0502-0.0446
Bzip2	INT	0.9543-0.9525	0.9027-0.9270	1.6428-0.5573
Astar	INT	0.8335-0.7726	2.1482-1.5707	1.8260-0.7140
Mcf	INT	0.3398-0.2619	2.6239-2.8608	1.6472-0.5575

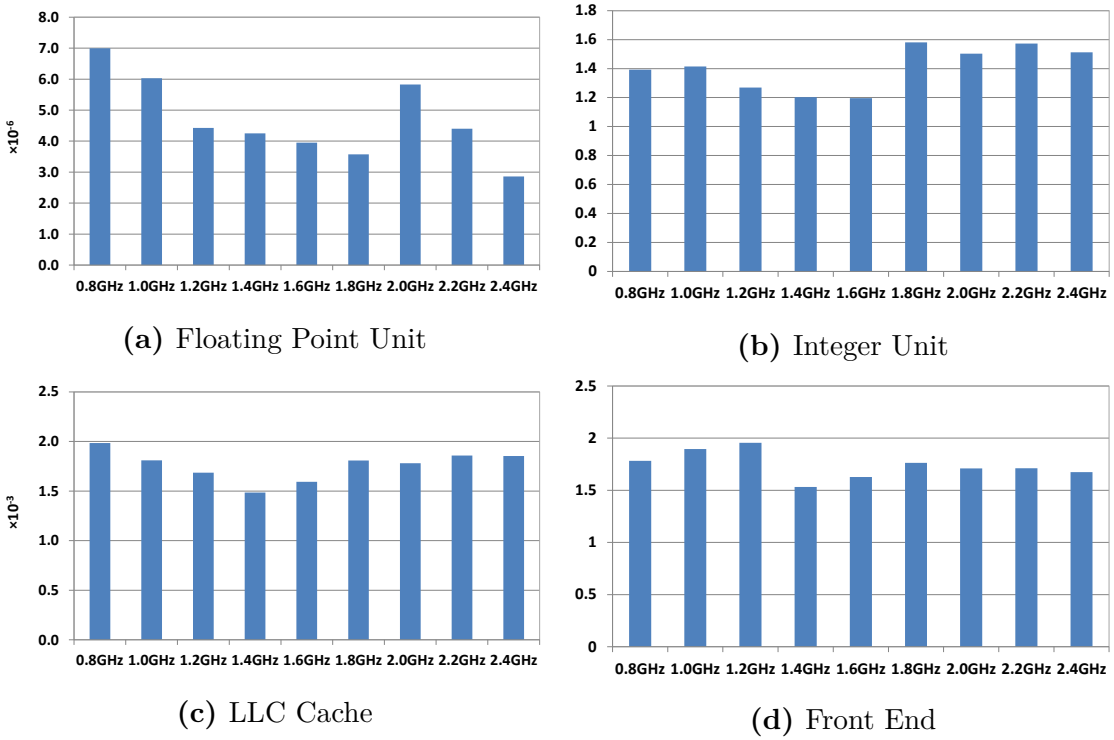
but instead the difference in the real activity ratio across DVFS state is reflected using the coefficients derived in Equation 3.1.

Therefore, the PMCs read at the current configuration can be used to predict power or performance at future DVFS state (P_f) while the Cl-State remains constant.

Predicting power across Cl-States. To predict power (ρ_f) across Cl-States, moving from current Cl-State (Cl_c) to future Cl-State (Cl_f), we build a multi linear regression model based on the ratio of the Cl-States and predicted power at the current DVFS state (ρ_c).

$$\rho_f = (\Delta \times \rho_c + (\beta \times \frac{Cl_f}{Cl_c})) + constant \tag{3.2}$$

Equation 3.2 represents the multi linear regression model for predicting ρ_f, where Δ, β and *constant* represent the multi linear regression coefficients. We build one model at each DVFS state.



(a) Floating Point Unit

(b) Integer Unit

(c) LLC Cache

(d) Front End

Figure 3.1 Microarchitectural component activity ratio for a SPEC CPU 2006 benchmark. The activity measured over 100 million instructions at all DVFS states for *lbm* from the SPEC CPU 2006 benchmark suite.

3.1.2 Modelling Performance

Modelling performance is based on three major constraints which are: (1) the inherent instruction-level parallelism of the execution flow; (2) number of stall cycles caused by misses in the cache hierarchy; (3) the number and latency of functional units such as INT and FP. The relationship between these constraints is tough to predict. Therefore, the most accurate way to estimate performance is by implementing a full-scale simulator [134, 135]. However, this increases the complexity of predicting performance in runtime and suffers a higher latency. On the other hand, linear regression techniques allow faster predictions and are easier to implement although they suffer from a relatively high error rate compared with simulation techniques.

Reporting performance at current configuration. The PMCs available in the current hardware subsystems allow precise measurement of the number of instructions retired (`instructions`). We take advantage of this counter and report the number of instructions retired (in millions) per second (MIPS), that is, η_c.

Predicting performance across DVFS states. We predict the performance in MIPS (η_f) of a thread using multi linear regression models. The components modelled are MIPS, IPC, private L1, private L2 and LLC and the ratio of change of DVFS state moving from P_c to P_f.

$$\eta_f = \sum_{i=0}^{comps} (\Delta_i \times AR_i) + (\beta \times \frac{P_f}{P_c}) + constant \tag{3.3}$$

Equation 3.3 represents the multi linear regression model for predicting η_f, where Δ_i, β and *constant* represent the coefficients to be learned and AR_i represents the activity ratio of the individual components. We build one model for each available DVFS state with no idle cycles.

Predicting performance across Cl-States. To predict performance (η_f) across Cl-States, moving from Cl_c to Cl_f, we build multi linear regression models while keeping the DVFS state fixed and moving across the Cl-States. The components modelled are MIPS, IPC, private L1, private L2 and LLC cache and the ratio of change of Cl-State.

$$\eta_f = \sum_{i=0}^{comps} (\Delta_i \times AR_i) + (\beta \times \frac{Cl_f}{Cl_c}) + constant \tag{3.4}$$

Equation 3.4 represents the multi linear regression model for predicting η_f, where Δ_i, β and *constant* represent the coefficients to be modelled and AR_i represents the activity ratio of the individual components. We build one model for every DVFS state available.

3.1.3 Single-Core Algorithm

The single-core algorithm predicts performance and power for a given DVFS State and Cl-State in a single thread using multi linear regression models.

1. Read the PMCs at current configuration and compute the activity ratios.
2. Predict power and report performance at current configuration.
3. Predict power and performance for current Cl-State and future DVFS state.
4. Predict power and performance for current DVFS state and future Cl-State.

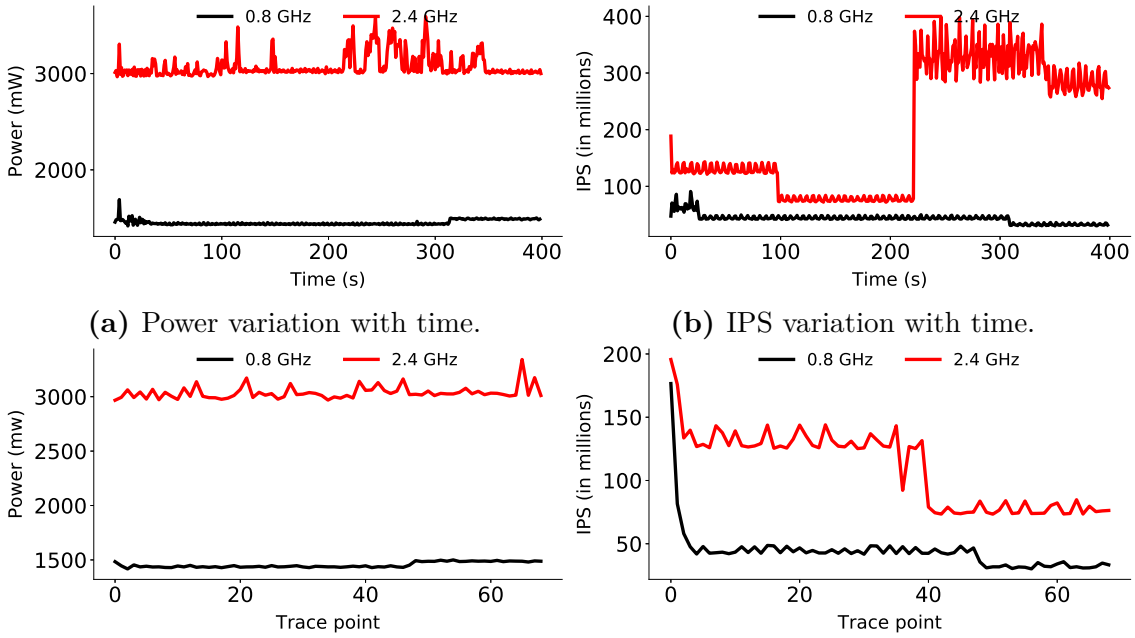
3.1.4 Training the Models

Our performance and power models for a single-core are built using a random training set from representative benchmark suites (training data set).[1] These benchmarks are executed to gather the PMCs values during a period of 100 seconds with a sampling period of 250 ms, at all DVFS states and a subset of the Cl-States.

The models are trained using a data set which consists of three floating point and three integer benchmarks, which are chosen at random, and run at all DVFS states and five Cl-States.[2] This results in 270 experiments on Intel, 24 experiments on AMD, and 18 experiments on ARM. Recollect that AMD and ARM do not have Cl-States; moreover only the Cortex A57 processor provides performance statistics on the ARM processor. The Cl-States are chosen such that there is at least 10 % variation (specifically the Cl-States are 10, 20, 30, 40, 50). The models were validated in two steps: offline and online. The *offline validation* of the models was carried extensively on the Intel processor as a proof of concept using the SPEC CPU 2006 benchmark suite (Section 3.3.2). Next, to validate the resulting single-core models *online*, we use the remainder of the benchmarks from SPEC CPU 2006, PARSEC 3.0, NAS, and SPLASH2x (validation set).

To build models offline, we collect traces containing PMCs samples of individual components per application. The traces, obtained at different DVFS states, need to be realigned to compare the activity ratios at similar points of execution (Section 3.1.5). This realigning is done with respect to the total instruction and exclude the trace points for which the difference in total instructions retired between two trace points for a given application exceeds ten million instructions. This process of realigning is to ensure that the trace points are at the same point of execution and makes the activity ratio comparable.

We empirically select a sampling period of 250 ms to collect PMCs and power using the RAPL register to build offline models as it increases the total number of trace points for a given application without exceeding ten million instructions. We have explored mapping intervals of 50 ms, 100 ms, and up to 950 ms, and 1000 ms, but 250 ms was chosen empirically as most SPEC benchmarks have less than 1 % change in performance or power at a particular DVFS state and allowed us to predict in the same phase.



(a) Power variation with time.

(b) IPS variation with time.

(c) Power traces realigned with IPS.

(d) IPS traces realigned with IPS.

Figure 3.2 Traces to build multi linear regression models. From left-to-right, power and performance *recorded* using the power meter and PMCs (on top), and traces *realigned* based on IPS (on bottom) for the SPEC benchmark astar.

3.1.5 Trace Files

The models are built offline using the statistical information gathered and stored in trace files. The trace files contain activity ratio for individual components for each benchmark. The traces, obtained at different DVFS states, Cl-States need to be realigned such that the activity ratios are comparable at similar points of execution.

The performance statistics for each benchmark in the training dataset are gathered by setting a thread affinity to a core using the Linux system call `sched_setaffinity`. Assigning an affinity [136] for a thread bypasses the Linux Completely Fair Scheduler (CFS) and helps avoid excessive thread migration and provides a more controlled environment.

Realigning traces gathered at DVFS state, Cl-State to model offline provides a unique challenge as the correlation between throughput, DVFS state, and time is non-linear. For instance, Figures 3.2a and 3.2b show power consumption (in mW) and throughput (in MIPS), respectively, for SPEC benchmark *astar* at 0.8 GHz (in black) and 2.4 GHz (in red) with Cl-State zero for the first 400 seconds of execution. For e.g., observe that the throughput obtained over the first 300 seconds at 0.8 GHz, and

[1]The models built are exclusive to a given architecture and are rebuilt for each architecture.
[2]The training workloads are consistent across architectures.

first 100-seconds at 2.4 GHz is approximately same. This shows that DVFS state and throughput do not have a one-to-one mapping. For this reason, prediction models can not be built using traces gathered based on a time varying demand of workloads. This raises the need to realign traces with respect to performance, thereby allowing (some) correlation between two different core frequency and performance.

As indicated before, a buffer of ten million instructions is used to compare traces for each benchmark, as most SPEC benchmarks exhibit less then 2 % difference and to ensure that the boundary of error is fixed (!). Figures 3.2c and 3.2d shows the traces realigned with respect to total performance for *astar* at 0.8 GHz and 2.4 GHz. Each trace gathered refers to (approximately) similar points of thread execution. This methodology is followed for every DVFS state and Cl-State combination, and such traces here-forth are referred to as ***realigned traces***.

3.2 Multicore Modelling

To predict total performance or power in multicore architectures, we *aggregate* the results from each of the *single-core* models (Section 3.1.3).

Single-core models when exposed to multicore prediction techniques are bound to suffer from an error due to the contention for shared resources [32, 30, 137–144]. To model the contention when predicting performance and power, we use four different types of multiprogrammed workloads with variations in memory footprint and validate the models, by switching between combinations of DVFS states and Cl-States.

It impossible to be at two DVFS states or Cl-States at the same time. Therefore for training and validating the models, we generate multiple (one thousand) random tuples of DVFS state, Cl-State combinations per core within the minimum and maximum DVFS state and Cl-State ranges.

The estimated increase in power consumption and degradation in performance due to the shared resource contention is modelled offline using multi linear regression techniques. We build one model for performance and another model for power. We demonstrate the process of building models for one core. This process is replicated across all cores simultaneously.

1. Read the 1000 random tuples of DVFS state and Cl-State per core.
2. Set current configuration for core to the current tuple and future configuration for the same core to next tuples.
3. Spawn training workloads consecutively.

④ Apply the single-core algorithm (Section 3.1.3) for the thread running on the core to predict power and performance in the future configuration.

⑤ Switch to future configuration after 250 ms.

⑥ Report power and performance at future configuration using RAPL and PMCs respectively.

⑦ Using the PMCs read at current configuration, compute activity ratios for private L1, private L2, LLC and MEM for the thread.

⑧ Compute difference between the PMCs read (or RAPL register) and prediction (using REPP) for performance (or power) per thread in the workload models the error due to shared resource contention.

The aforementioned method is followed for all random tuples generated for all cores for a period of 300 seconds.

$$Error = \sum_{i=0}^{comps} (\Delta_i \times AR_i) + constant \tag{3.5}$$

Equation 3.5 represents the multi linear regression model to predict the error due to shared resource contention. Where Δ_i represents the coefficient to be learnt and AR_i represents the activity ratio of the individual components.

Intuitively, in a multicore architecture as the number of threads increases, the performance per thread decreases and the total power consumption increases. Since the single-core models do not include contention in shared environments, the total power consumed will be lower and the performance higher. Therefore, the modelled error for power and performance is added and subtracted for every prediction on a per thread level respectively.

Then, we evaluate REPP across a wide range of performance and power constraints. Contrary to prior works [39, 70, 123, 135, 145], which predict power and performance at a system level, our work predicts at a system level on a per core basis.

With the inclusion of the shared resource contention model (Equation 3.5), the architecture of REPP for multicore systems is complete. Figure 3.3 shows a high-level view of REPP. REPP is a technique that can be implemented across any multicore server system running an operating system that allows to gather performance statistics of workloads (e.g., perf, on all our multicore machines). Such performance statistics from each thread are fed to REPP to compute the activity ratio's which are fed to the

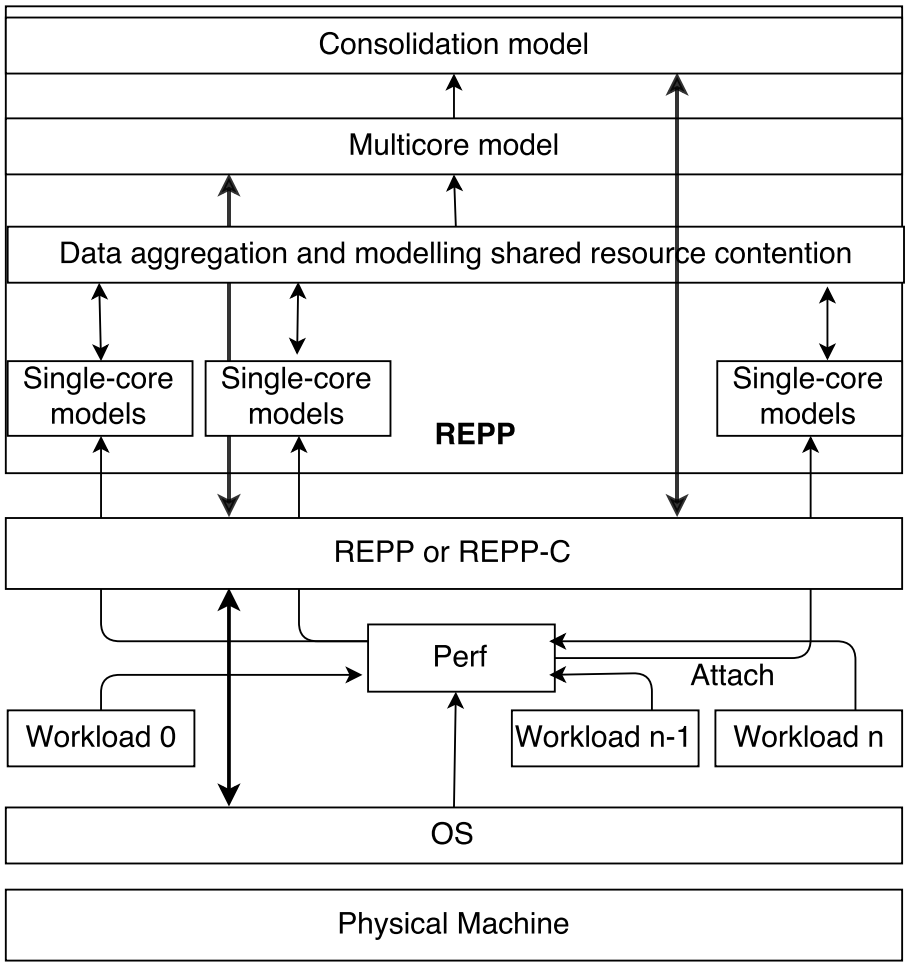


Figure 3.3 High-level view of REPP runtime system

single-core models to estimate the power and performance across a wide range of DVFS states and Cl-States. Using equation 3.5, we model the shared resource contention as the single-core models ignore it. Then, the results from each of these single-core models are *aggregated* to estimate the power for a multicore system.

3.3 Single-Core Model Evaluation

In this section, we first introduce the model assessment metric, then we evaluate the models offline by predicting performance and power using the traces gathered. Next, we evaluate the single-core models when predicting performance and power at runtime/online. Both the evaluations are carried by predicting performance and power from the current DVFS state, Cl-State to numerous DVFS state and Cl-State combinations.

3.3.1 Model Assessment

The models are evaluated in the remainder of this chapter in terms of Percentage Absolute Average Error (PAAE), and the Standard Deviation (STDEV) for each data point over a period of 300 seconds:

$$PAAE = \frac{1}{N}\Big(\sum_{i=1}^{N}\frac{|M_i - m_i|}{m_i}\Big) \tag{3.6}$$

Where N is the number of data points for each benchmark, M is the predicted value, and m is the measured value (Reproduced from [146]).

The PAAE is a well-known metric [39, 133, 146, 147] to evaluate the predicted performance (or power) values against the actual values reported by PMCs for performance (or by power monitors for power consumption). The standard deviation over PAAE determines the variability of PAAE from the actual value.

3.3.2 Offline Evaluation

The offline evaluation of the single-core models is carried out over a subset of DVFS states and Cl-States. The subset is chosen to represent the spectrum of error variability. Specifically, the DVFS states chosen are 0.8 GHz, 1.6 GHz, and 2.4 GHz; and the Cl-States chosen are 0, 30 and 50.

The PAAE is computed as the error between the predicted performance and power at the future DVFS state and Cl-State using the PMCs obtained at the current DVFS state and Cl-State for a given number of instructions retired. The PMCs and power statistics for a given workload at each DVFS state, Cl-State are obtained from the realigned traces. The error is computed over a period of 300 seconds.

Table 3.5 shows the power prediction error, in terms of PAAE and STDEV, when switching from lower to higher DVFS state (left-hand side of the table) and higher to

lower DVFS state (right-hand side of the table) for SPEC workloads. Our results have shown that the PAAE over the subset of switches in DVFS states is less than 2.30 % (mean), while the maximum error is 5.78 % for *soplex*.

Table 3.6 shows the performance prediction error, in terms of PAAE and STDEV, when switching from lower to higher DVFS state (left-hand side of the table) and higher to lower DVFS state (right-hand side of the table) for SPEC workloads. The worst-case scenario PAAE is 33.99 % for the memory-intensive benchmark *xalancbmk*. Also, note that there exists a collinearity between PAAE and the ratio of change in DVFS state. For instance, observe there is a linear increase (or decrease) in PAAE for benchmark *astar* when switching from lower-to-higher (or higher to lower) DVFS state.

Table 3.7 shows the power prediction error, in terms of PAAE and STDEV, when switching from lower to higher Cl-State (left-hand side of the table) and higher to lower Cl-State (right-hand side of the table) at 1.8 GHz. Our results have shown that the PAAE over the subset of switches in Cl-State is less than 4.30 % (mean), with a maximum error of 10.45 % for *xalancbmk*.

Table 3.8 shows the performance prediction error, in terms of PAAE and STDEV, when switching from lower to higher Cl-State (left-hand side of the table) and higher to lower Cl-State (right-hand side of the table) at 1.8 GHz. The mean error observed when switching between the Cl-States is 15.38 % (with a maximum error of 37.77 % for *calculix*).

Our results demonstrate that the error across numerous switches, for both DVFS states, and Cl-States, in an offline modelling technique is "acceptable" [30, 39, 124–126, 147], to determine the real behaviour of the application.

Figure 3.4 represents the predicted performance (y-axis), and power (x-axis) for three workloads: *astar* (mid memory intensive; top row, subfigures (a), (b)), *mcf* (memory intensive; middle row, subfigures (c), (d)), and *calculix* (compute intensive; bottom row, subfigures (e), (f)). In the left column, the colour map corresponds to the maximum PAAE between performance, and power ($max_{PAAE} = max(PAAE_{power}, PAAE_{performance})$); where each point in the graph represents a unique hardware configuration. By contrast, in the right column, we represent the data plotted in the left column in terms of maximum error per sub-grid for each benchmark, where a configuration exists. The initial configuration is set to 0.8 GHz and Cl-State zero. We evaluate the models offline over nine DVFS states and six Cl-States. These graphs conclude that there exists at least one prediction with less than 15 % error. This behaviour was observed across all SPEC benchmarks.

Table 3.5 Power prediction error across DVFS states for SPEC benchmarks. Error shown in terms of PAAE and STDEV. The highest PAAE when predicting across DVFS states is **boldfaced**.

	Lower-Higher						**Higher-Lower**					
DVFS state (GHz)	0.80 - 1.00		0.80 - 1.60		0.80 - 2.40		1.00 - 0.80		1.60 - 0.80		2.40 - 0.80	
Benchmark	*PAAE*	*STDEV*	*PAAE*	*STDEV*	*PAAE*	*STDEV*	*PAAE*	*STDEV*	*PAAE*	*STDEV*	*PAAE*	*STDEV*
Astar	1.77	1.28	1.49	0.97	1.17	0.78	1.74	0.67	1.72	0.71	2.25	2.58
Bzip2	1.93	0.95	1.92	2.48	1.40	0.98	1.14	0.68	0.82	0.53	2.05	1.73
Calculix	1.27	1.20	1.27	1.13	3.48	2.10	1.11	0.57	2.37	2.25	2.36	2.04
GemsFDTD	3.88	2.77	3.54	2.13	4.24	2.30	3.74	2.95	3.78	2.24	3.36	1.70
Gobmk	1.05	0.94	1.34	1.61	2.41	3.20	0.86	0.65	0.70	0.47	0.67	0.57
Hmmer	2.60	0.53	2.52	0.86	3.06	1.45	2.74	0.62	2.70	0.60	2.31	0.93
Lbm	0.85	0.69	1.82	1.10	2.16	1.38	0.84	0.70	1.43	0.83	1.47	0.77
Leslie3d	1.10	0.86	1.32	1.28	2.04	1.20	1.24	0.96	1.17	0.93	1.38	1.23
Libquantum	2.68	3.91	1.64	1.26	2.44	1.95	3.33	2.77	1.96	1.50	1.85	1.85
Mcf	3.17	2.36	3.79	2.68	5.67	2.25	2.42	2.66	2.29	2.23	2.42	2.72
Povray	1.14	0.62	2.05	1.09	1.90	3.25	1.23	0.51	1.40	0.55	1.06	0.47
Soplex	5.13	1.72	2.76	1.63	2.82	1.16	**5.78**	1.99	5.69	1.88	4.14	1.71
Xalancbmk	2.52	1.59	3.63	1.58	5.81	1.46	1.99	1.36	1.62	1.15	1.40	1.24
Mean	2.24	1.49	2.24	1.52	2.97	1.81	2.17	1.31	2.13	1.22	2.05	1.50

Table 3.6 Performance prediction error across DVFS states for SPEC benchmarks. Error shown in terms of PAAE and STDEV. The highest PAAE when predicting across DVFS states is **boldfaced**.

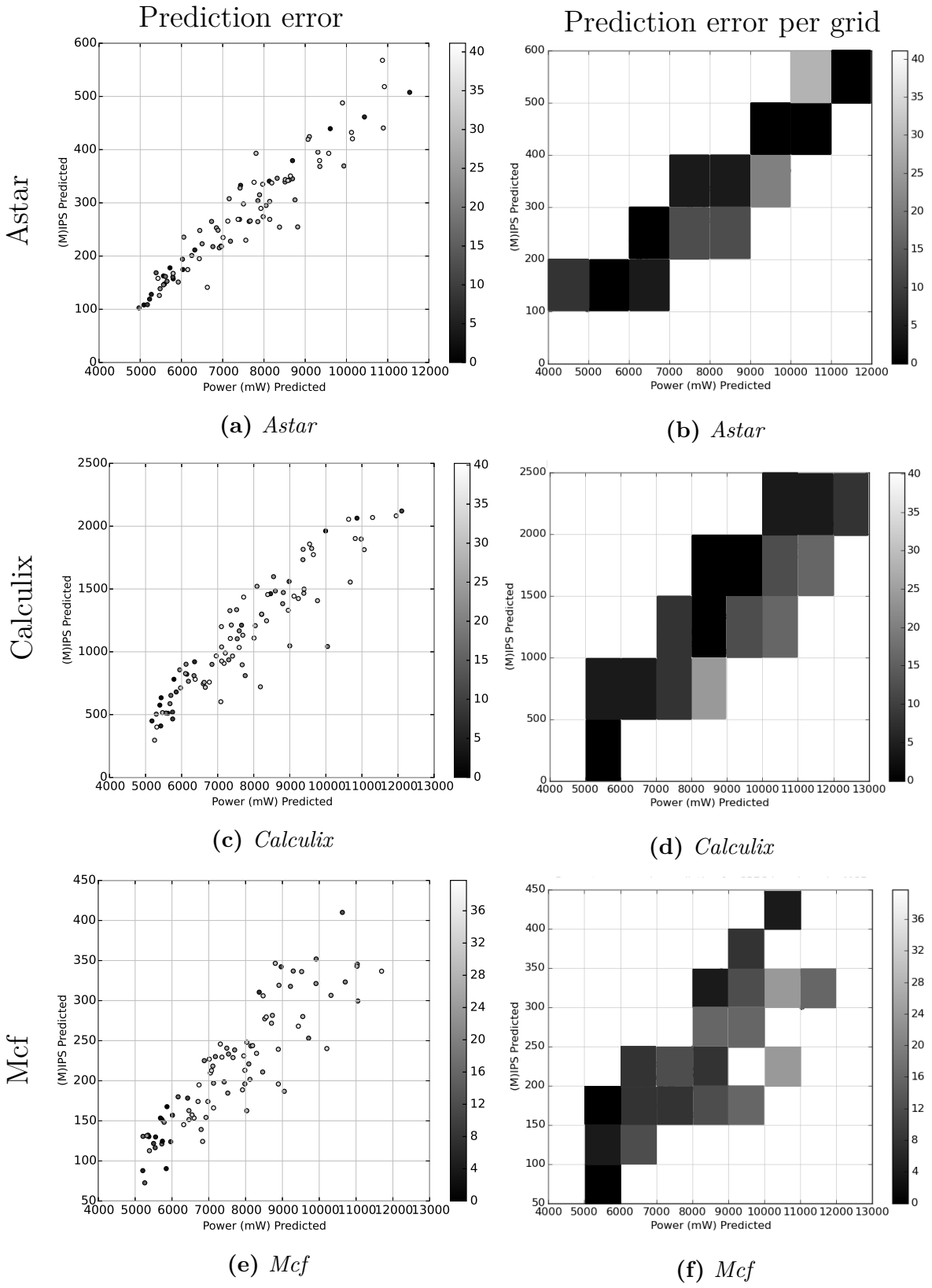
	Lower-Higher						**Higher-Lower**					
DVFS state (GHz)	0.80 - 1.00		0.80 - 1.60		0.80 - 2.40		1.00 - 0.80		1.60 - 0.80		2.40 - 0.80	
Benchmark	*PAAE*	*STDEV*	*PAAE*	*STDEV*	*PAAE*	*STDEV*	*PAAE*	*STDEV*	*PAAE*	*STDEV*	*PAAE*	*STDEV*
Astar	9.41	16.75	11.17	14.05	24.08	9.52	7.92	8.65	9.13	9.41	16.24	14.44
Bzip2	13.16	11.36	17.28	11.75	22.98	15.54	13.06	10.50	19.80	17.59	22.66	13.48
Calculix	22.19	13.44	19.24	15.12	26.97	20.41	17.22	27.00	18.17	24.18	19.89	23.93
GemsFDTD	21.69	15.48	23.88	14.50	15.65	14.65	21.71	15.58	22.43	12.32	12.71	9.62
Gobmk	8.97	7.82	8.74	8.69	9.24	7.16	8.85	7.61	8.89	9.12	9.61	8.05
H264ref	6.10	4.69	9.18	14.61	1.89	1.35	6.12	4.71	7.71	8.51	1.94	1.42
Hmmer	5.70	5.98	6.09	5.34	7.00	4.74	5.99	7.57	6.14	5.34	7.43	5.26
Lbm	6.93	5.10	5.82	4.12	5.77	3.87	6.84	4.98	5.73	4.01	5.63	3.77
Libquantum	8.60	6.36	10.30	7.10	12.58	10.28	8.64	6.52	9.55	6.45	10.54	7.70
Mcf	11.08	9.70	17.72	15.11	29.33	20.95	10.44	9.11	14.67	12.78	21.43	11.33
Povray	6.15	4.42	12.09	22.55	6.93	5.11	6.24	4.67	9.15	10.64	7.12	5.42
Soplex	8.83	6.74	10.34	8.56	16.78	13.80	8.19	5.76	9.04	6.55	13.37	8.67
Xalancbmk	9.65	8.57	19.31	13.98	**33.99**	16.75	9.70	9.43	15.46	9.20	25.17	11.01
Mean	10.65	8.95	13.17	11.96	16.40	11.09	10.07	9.39	11.99	10.47	13.37	9.55

Table 3.7 Power prediction error when switching across Cl-States at 1.80 GHz **for SPEC benchmarks.** Error shown in terms of PAAE and STDEV. The highest PAAE when predicting across Cl-States is **boldfaced**.

	Lower-Higher						**Higher-Lower**					
Cl-State	0 – 10		0 – 30		0 – 50		10 – 0		30 – 0		50 – 0	
Benchmark	*PAAE*	*STDEV*	*PAAE*	*STDEV*	*PAAE*	*STDEV*	*PAAE*	*STDEV*	*PAAE*	*STDEV*	*PAAE*	*STDEV*
Astar	3.19	2.31	2.69	1.75	2.11	1.40	3.12	1.21	3.09	1.28	4.04	4.64
Bzip2	3.47	1.70	3.46	4.47	2.52	1.77	2.05	1.22	1.48	0.95	3.70	3.11
Calculix	2.28	2.15	2.29	2.04	6.26	3.78	2.01	1.03	4.27	4.04	4.24	3.68
GemsFDTD	6.99	4.99	6.37	3.83	7.63	4.15	6.73	5.31	6.81	4.03	6.04	3.07
Gobmk	1.90	1.69	2.41	2.89	4.34	5.76	1.54	1.17	1.26	0.84	1.20	1.03
Hmmer	4.68	0.95	4.54	1.55	5.51	2.61	4.93	1.11	4.85	1.08	4.15	1.67
Lbm	1.53	1.24	3.28	1.98	3.88	2.48	1.51	1.26	2.58	1.49	2.65	1.39
Leslie3d	1.97	1.55	2.38	2.31	3.67	2.16	2.23	1.72	2.10	1.68	2.48	2.22
Libquantum	4.82	7.03	2.96	2.26	4.38	3.52	5.99	4.98	3.52	2.69	3.33	3.33
Mcf	5.71	4.24	6.81	4.82	10.20	4.05	4.35	4.78	4.13	4.01	4.36	4.90
Povray	2.06	1.11	3.69	1.97	3.42	5.85	2.22	0.93	2.51	0.98	1.91	0.84
Soplex	9.24	3.10	4.97	2.93	5.07	2.09	10.40	3.58	10.24	3.38	7.44	3.07
Xalancbmk	4.54	2.86	6.53	2.84	**10.45**	2.62	3.58	2.44	2.91	2.08	2.52	2.24
Mean	4.03	2.69	4.03	2.74	5.34	3.25	3.90	2.37	3.83	2.19	3.70	2.71

Table 3.8 Performance error when switching across Cl-States at 1.80 GHz **for SPEC benchmarks.** Error shown in terms of PAAE and STDEV. The highest PAAE when predicting across Cl-States is **boldfaced**

	Lower-Higher						**Higher-Lower**					
Cl-State	0 – 10		0 – 30		0 – 50		10 – 0		30 – 0		50 – 0	
Benchmark	*PAAE*	*STDEV*	*PAAE*	*STDEV*	*PAAE*	*STDEV*	*PAAE*	*STDEV*	*PAAE*	*STDEV*	*PAAE*	*STDEV*
Astar	9.84	9.80	12.44	8.44	23.71	11.46	9.39	7.75	11.46	8.60	13.59	11.94
Bzip2	12.49	8.75	19.40	12.96	28.11	16.54	12.85	10.34	19.68	11.79	23.97	19.87
Calculix	23.68	12.98	17.18	16.38	**37.77**	22.14	17.68	14.59	19.89	8.62	23.29	6.01
GemsFDTD	17.16	14.75	15.11	9.98	24.32	16.47	16.93	15.11	15.31	11.47	23.79	17.36
Gobmk	9.29	5.73	12.16	8.16	24.46	11.36	9.50	5.95	11.56	7.67	14.50	10.78
H264ref	10.30	5.41	10.65	10.86	25.19	12.03	10.67	6.16	12.77	8.66	15.13	9.91
Hmmer	7.41	4.30	10.47	6.96	24.09	12.73	7.38	4.37	8.19	6.06	16.06	10.18
Lbm	10.24	5.79	12.02	7.58	25.26	12.58	10.34	6.47	11.14	7.13	15.77	13.23
Libquantum	8.86	6.15	12.24	10.85	24.81	12.91	8.87	5.85	13.73	12.96	14.65	10.19
Mcf	9.69	7.40	15.26	10.72	27.50	13.71	10.65	9.18	14.19	18.81	19.42	21.16
Povray	8.50	5.06	9.75	8.52	23.42	10.50	8.72	5.41	11.30	8.31	12.90	6.86
Soplex	7.86	5.26	13.41	8.98	25.49	14.18	7.97	5.20	12.41	7.41	17.50	16.83
Xalancbmk	9.53	7.61	10.70	9.38	31.95	15.76	9.00	6.76	11.38	11.19	14.59	8.90
Mean	11.14	7.61	13.14	9.98	26.62	14.03	10.77	7.94	13.31	9.90	17.32	12.55



(a) *Astar*

(b) *Astar*

(c) *Calculix*

(d) *Calculix*

(e) *Mcf*

(f) *Mcf*

Figure 3.4 Percentage prediction error for SPEC benchmarks. The power and performance prediction error for Astar (mid memory intensive), Calculix (compute intensive), and Mcf (memory intensive).

3.3.3 Online Evaluation

We evaluate and summarise the single-core modelling technique at runtime for over 150 applications using K-Means clustering algorithm [148]. The evaluation is carried out by predicting performance and power across combinations of DVFS state and Cl-State at runtime. The PAAE on a single-core is computed using the error between the value measured from PMCs for performance (and power meters for power) and predicted performance (or power), and the error bars represent the STDEV.

First, we separate the benchmarks used in validation into four clusters, to present the results for over 150 single-threaded applications, using K-Means with parameters FE, INT, FP, MEM, BPU, L1, L2, and L3. The number of clusters (four) was chosen empirically based on the silhouette coefficient. We narrow the number of parameters to two using principal component analysis for keeping the most singular vectors to project the data in a lower dimensional space. Clusters are named with the architecture and cluster number, such as ARM-0, Intel-3, etc. Each cluster has results from all four categories: Insensitive, Cache-Friendly, Cache-Sensitive, and Thrashing. The benchmarks in each category are given in Table 2.4.

Figure 3.5 shows the average PAAE over-all applications in a cluster on a single-core and Figure 3.7 presents the results for each application in detail on a single-core. The results in the Figure 3.7 are organised as follows: Intel (top row (a)), AMD (middle row (b)), and ARM (bottom row (c)). We analyse the data points with higher error and also pointed out the sources of error below.

(a) Average PAAE when predicting performance for Intel-2 for *thrashing* benchmarks is 15.8 % because *mcf* has 22.5 % error as it is a pointer-chasing benchmark [104] and generates more than 41000 LLC misses per million instructions retired and the models

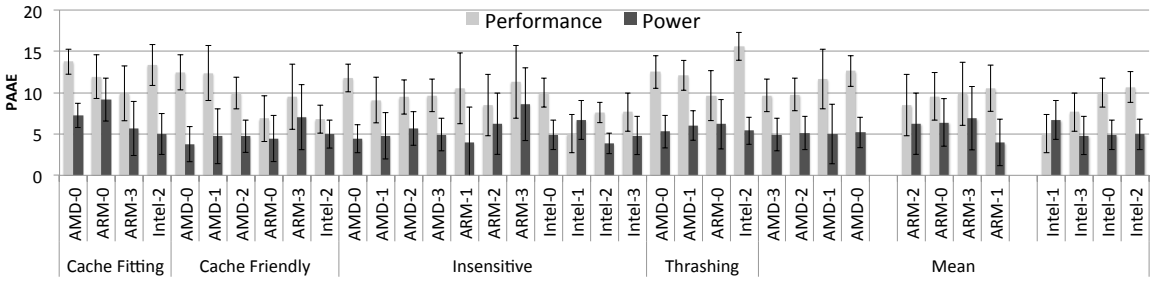


Figure 3.5 Average PAAE when predicting performance and power on a single-core. Average PAAE when predicting performance and power on a single-core for across a combination of DVFS states, Cl-States and architectures. The error bars represent the STDEV.

are not trained for that range. On the other hand, applications like *lbm* (a memory intensive floating point benchmark) generate 3000 LLC misses per million instruction retired has an error 11.2 %.

(b) The average PAAE for performance for *Cache-Friendly* on AMD-0 and AMD-1 are 12.4 % and 12.3 %, respectively; this is because both clusters contain applications such as *canneal* and *dedup*. The possible sources of error are: (1) Both applications have a high dynamic variability in application phases [149], which leads to erroneous counters due to PMCs multiplexing. (2) In contrast to the other applications across suites, these benchmarks have a shorter execution time. (3) Observe that canneal is a cache fitting benchmark on Intel, by contrast it is a cache-friendly benchmark on AMD. This is because of the aggressive hardware prefetcher on Intel causing a higher miss rate [150], thereby leading to fewer dynamic phase changes and relatively smaller error of 6.5 %.

We also observe that application *radix* is a cache-fitting, integer radix sorting algorithm, has very high activity in FE, across three different architectures, even though other benchmarks across four suites do not show this behaviour.

Figure 3.6 shows average PAAE over all applications in each benchmark suite across architectures, with error bars representing STDEV. Across architectures, we observe performance PAAE is higher for SPEC benchmarks, which have high variability at runtime, and low for NAS benchmarks, which have less variability after the initialisation phase. We conclude that the models to predict performance and power are accurate enough to capture the real behaviour, and since the computational complexity at runtime is low, they can be used for fine-grain power or performance management.

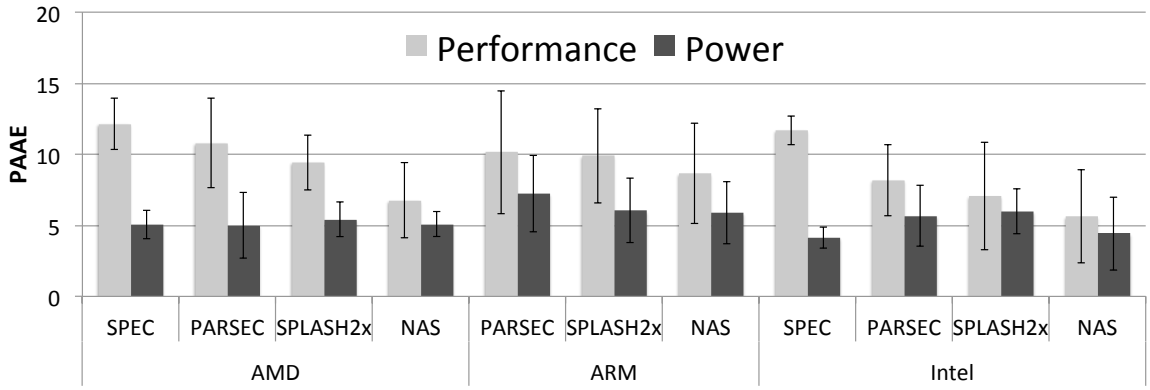


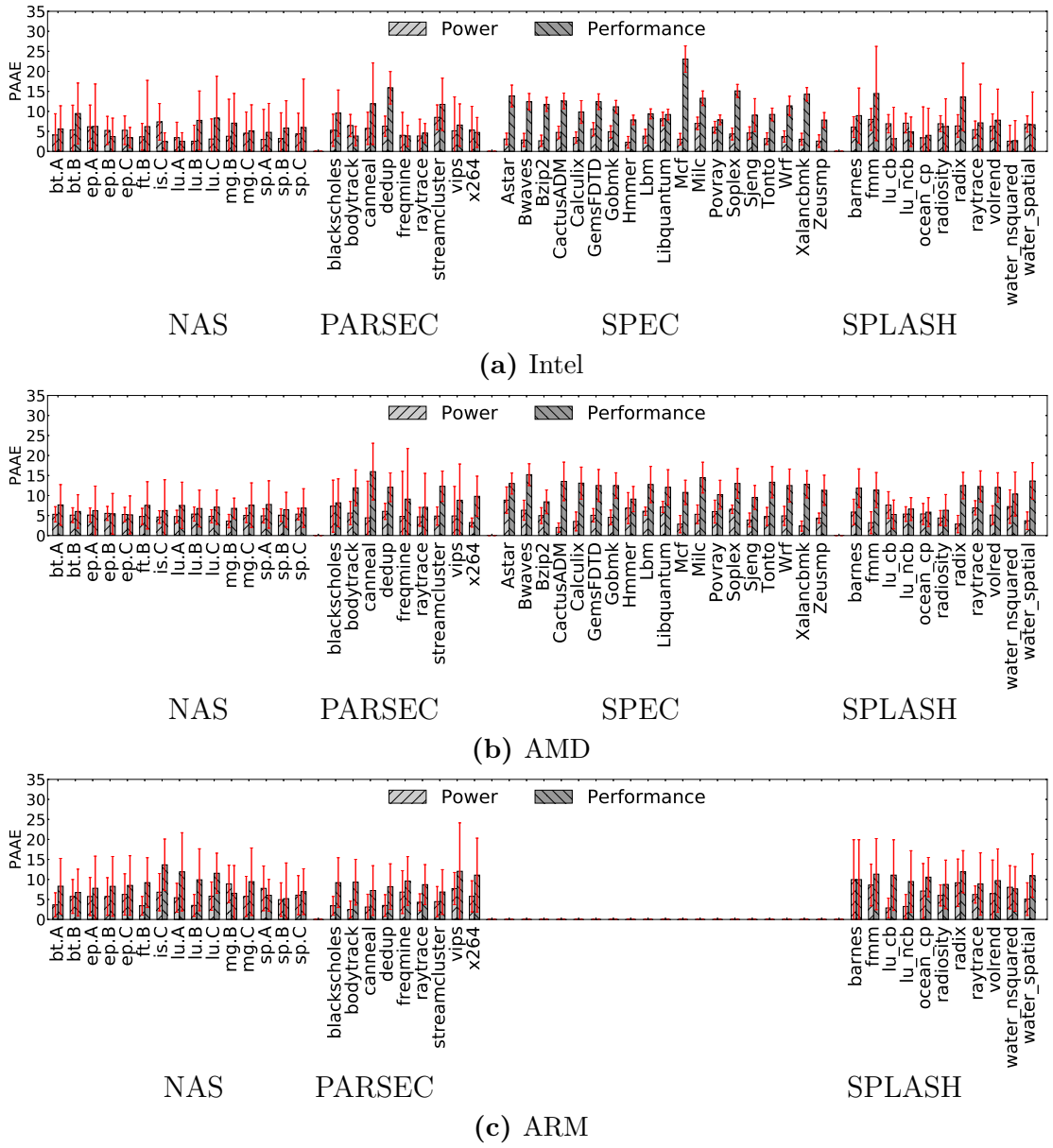
Figure 3.6 Average PAAE when predicting performance and power per benchmark on a single-core. The error bars represent the STDEV.

Table 3.9 PAAE over combinations of DVFS states and Cl-States. PAAE when predicting performance for every combination of switch in DVFS state and for a specific set of differences when switching between Cl-States for a single-core Intel architecture. Similar results were observed across AMD, and ARM.

Leap in DVFS State	1	2	3	4	5	6	7	8	-
DVFS state	14.14	16.75	23.37	15.37	13.78	57.17	18.49	27.49	-
Leap in Cl-State	5	10	15	20	25	30	35	40	45
Cl-State	24.86	26.25	29.14	26.36	23.25	22.07	29.30	31.94	34.56

The models to predict power can be built using standardised power meters and the models are built using PMCs that are available across architectures.

Also, we observe that when predicting performance at the future DVFS state on Intel processors, the error varies depending on the leap size between current DVFS state to future DVFS state or current Cl-State to future Cl-State. We compute PAAE when switching between every combination of DVFS states. For every switch in DVFS state, we calculate the PAAE when switching between every combination of DVFS states. Similarly, we calculate the PAAE for Cl-States when the difference in switches is a multiple of five for every DVFS state. As can be seen in Table 3.9, with a higher switch in hardware configurations from the current configuration, a greater error is observed. This is because, we train the models using a small subset of benchmarks, and use it for a broad range of threads which are not a part of the training set. Similar results were observed across ARM and AMD machines.



(a) Intel

(b) AMD

(c) ARM

Figure 3.7 Average PAAE when predicting performance and power per benchmark on a single-core. Average PAAE when predicting performance and power on a single-core for each benchmark from four benchmark suites across a combination of DVFS states, Cl-States and architectures. The error bars represent the STDEV. Results are shown for Intel (top), AMD (middle), and ARM (bottom). The benchmarks are organised as follows from left-to-right: NAS, PARSEC, SPEC, and SPLASH2x

.

3.4 Multicore Model Evaluation

The multicore evaluation of REPP is carried out in three parts. At first, we validate the error introduced by the single-core models on multicore architectures, that is, without considering shared resource contention (Equation 3.5 and Section 3.4.1). Thereafter, we introduce the model for shared resource contention and validate REPP without any performance or power constraint (Section 3.4.2). Finally, we validate REPP (with the contention model) by limiting the power usage or delivering a minimum performance for a wide spectrum of workloads on three different architectures (Section 3.4.3). The multiprogrammed workloads in all three cases are generated based on the methodology described in section 2.6.1. Irrespective of the how REPP is evaluated, the algorithm is invoked every 250 ms and each experiment was run multiple times. The standard deviation over multiple runs for each workload was low ($< 2\,\%$).

In the first two parts (sections 3.4.1 and 3.4.2), we validate REPP by switching across all combinations of DVFS states and Cl-States only on Intel processor. Since the total number of DVFS state and Cl-State combinations are 41 billion (450 per core and 4 cores). It is infeasible to validate for the entire spectrum. Therefore we generate 1000 random tuples of DVFS state, Cl-State combinations per core within the minimum and maximum DVFS state and Cl-State ranges. These random combinations are fixed across workloads. The randomisation was done using the `rand()` function. Moreover, the multiprogrammed workloads generated are taken from SPEC CPU 2006 and PARSEC 3.0 benchmark suites.

In the third part (section 3.4.3), we validate REPP on three different architectures with multiprogrammed workloads built from SPEC CPU 2006, PARSEC 3.0, NAS, and SPLASH2x. The process of delivering a minimum performance or limiting the power usage of the workload is done by selecting a configuration from the predicted values that satisfy the configuration.

The *training workloads* selected for building the contention model (refer section 3.2) in our case study are **SSSS**, **NNNN**, **TTTT**, and **FFFF** as they have a huge variation in memory footprint and CPU requirements.

3.4.1 Multicore Models ignoring Contention

Figure 3.8 shows PAAE in predicting power and performance across 1000 combinations of DVFS states and Cl-States per core in multicore architecture. We highlight two key points from this graph. First, observe that workloads **SSSS** (a memory intensive workload) and **NNNN** (a compute intensive workload) have the highest PAAE when

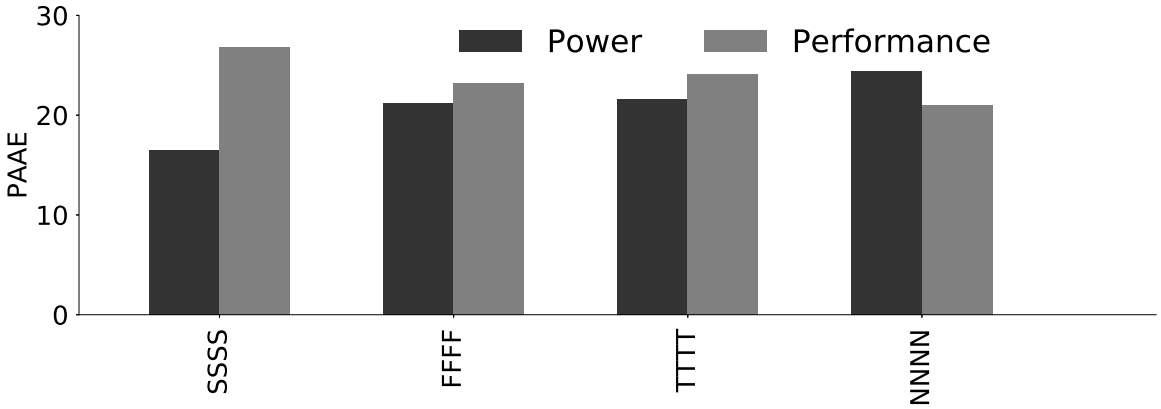


Figure 3.8 PAAE ignoring contention for shared resources. PAAE when predicting **power** and **performance** across 1000 combinations of DVFS states and Cl-States per core in multicore architectures for training workloads.

predicting performance and power, respectively, this is because the activity is predominant in the memory subsystem and processor, respectively. Second, observe that the error in predicting power increases as the compute intensiveness of the workloads increases, this is because the single-core models aggregate the results obtained from each core and do not account for the contention due to shared resources, thereby the compute intensive workloads which have lower activity in memory subsystem – compared to memory intensive workloads – are accounted for higher activity. Whereas the error when predicting performance increases as the memory boundedness of the workloads increase, this can be attributed to the fact that the activity generated per cycle in the components decreases, thereof the performance per thread. The PAAE and Absolute Average Error (AAE) when predicting power and performance across the training workloads are 20.93 % (430 mW) and 23.8 % (3316 MIPS).

3.4.2 Multicore Models including Contention without Constraints

Figure 3.9 shows an example of the power and performance prediction in runtime implemented on our system for the first 20 seconds of execution for the workload, SSSN. Specifically the workload SSSN consists of benchmarks *milc*, *milc*, *xalancbmk* and *blackscholes*. From top-to-bottom, the first (and second) graph represents the power (and performance) as measured using RAPL (and PMC) and the prediction made using REPP. The third and fourth graphs show the random combination of DVFS states and Cl-States generated for individual cores, respectively, for the first

20 seconds. We highlight two results. First, REPP does show the capability to adapt to workloads consisting of multiple thread phases (SPEC CPU 2006 and PARSEC 3.0 benchmarks have both memory and computational bound phases). For instance, observe at second 12, REPP makes a 11 mW error in predicting power, this is because of the huge changes in DVFS states and Cl-States. In this scenario, the DVFS states for core zero, one, two, three change from 0.8 GHz to 2.4 GHz, 0.8 GHz to 2.2 GHz, 0.8 GHz to 2.2 GHz and 1.2 GHz to 0.8 GHz respectively and the Cl-States change from 10 to 23, 1 to 31, 41 to 48 and 3 to 35. Observe that these errors only occur with huge changes in DVFS states and Cl-States in rapid intervals (for example, second four). Ozlem *et al.* [29] on the other hand, show that rapid changes in power or performance are seldom required in data centre environments. Second, REPP can predict power and performance per thread, which can not be accomplished using the in-built RAPL

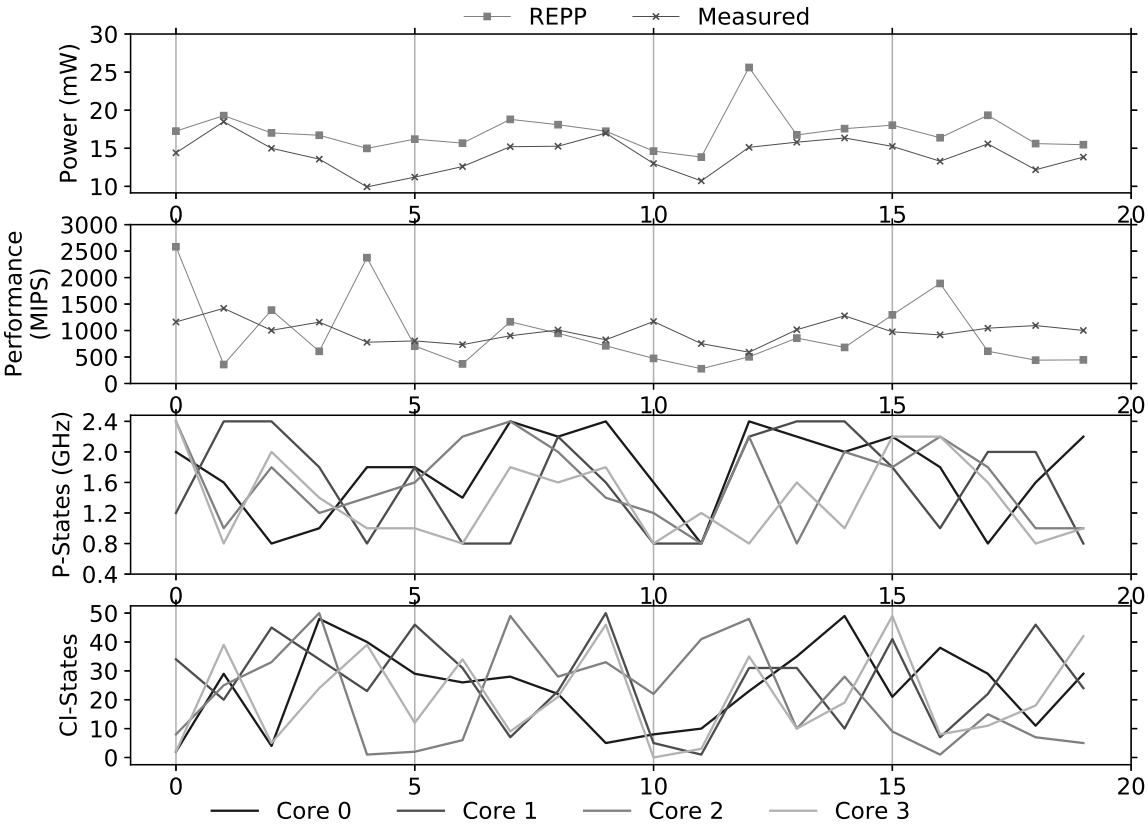


Figure 3.9 Power and performance prediction for workload SSSN. Runtime power and performance prediction over time (in seconds) for workload SSSN. The legend at the top is for the first two graphs, and at the bottom is for the last two graphs.

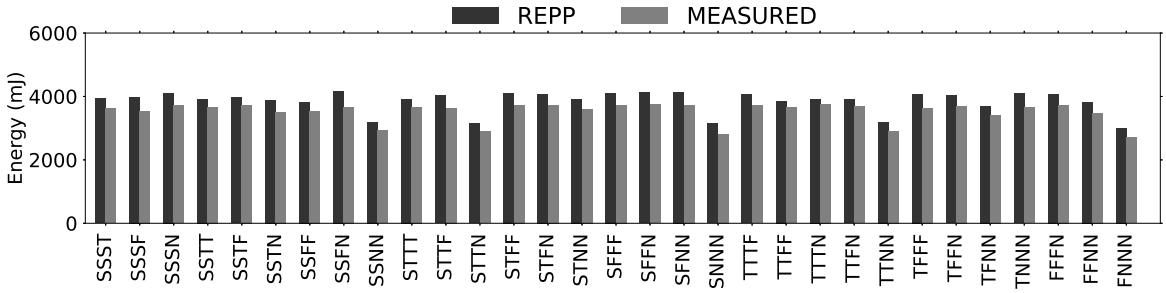


Figure 3.10 Power prediction for multiprogrammed workloads without constraints. *Energy consumed (*mJ*)* across all workloads on Intel processor. The y-axis is read as predicted (REPP) and measured (using RAPL).

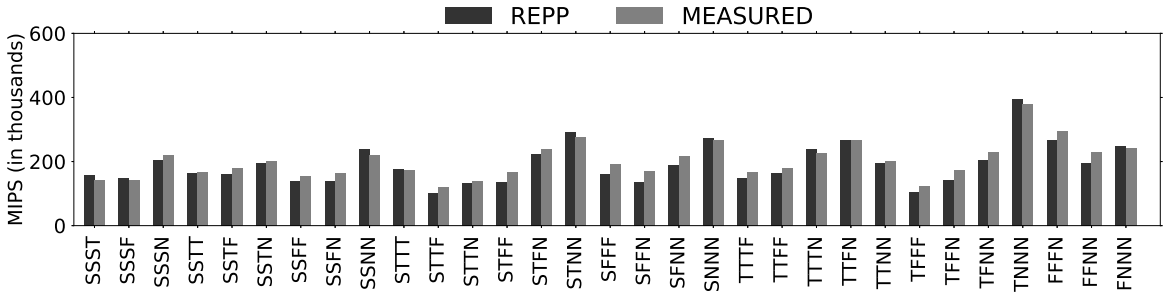


Figure 3.11 Performance prediction for multiprogrammed workloads without constraints. *Total performance (in thousands)* across all workloads on Intel processor. The y-axis is read as predicted (REPP) and measured (using PMCs).

register. In this particular workload, we make an error of 9.4 % (384 mJ) and 15.2% (1500 MIPS) when predicting power and performance over 300 seconds, respectively.

Figure 3.10 shows the energy consumption in millijoules (mJ) using our prediction technique, REPP and the power measured using native RAPL register for all workloads over a period of 300 seconds when switching across 1000 combinations of DVFS states and Cl-States. On average, workloads incur an error in prediction of 8.6 % or 332.31 mJ. Observe that the maximum error we incur is 12.1 % (507.36 mJ) in workload SSFN.

Figure 3.11 shows the total performance (MIPS) in thousands using our predicting technique (REPP), and the performance `measured` using PMCs for all workloads over a period of 300 seconds when switching across 1000 combinations of DVFS states and Cl-States. The average error we incur is 8.8 % (over the 1000 combinations on all cores) and the maximum error is 18.8 % in workload SFFN.

Table 3.10 shows the power and performance prediction error without constraints for the remaining multiprogrammed workloads when disregarding certain benchmark categories. It is observed that the power and prediction error for the remaining benchmarks even after disregarding a subset of benchmarks is (approximately) same.

Table 3.10 Performance and power prediction error without constraints while disregarding certain benchmark categories. Error is shown in terms of PAAE.

Label	Benchmark category	Power	Performance
N	Insensitive	8.2 %	10.2 %
F	Cache-Friendly	8.7 %	9.8 %
T	Cache-Fitting	9.6 %	9.9 %
S	Thrashing	8.2 %	7.5 %

Figure 3.10 and 3.11 demonstrate that REPP has been validated under multiple combinations of DVFS states and Cl-States across 35 multiprogrammed workloads. Moreover, each bar in the histogram represents a summary of the workload execution, similar to Figure 3.9.

3.4.3 Multicore Models including Contention with Constraints

In this section, we validate multicore models of REPP by predicting performance and power at the hardware settings DVFS states and Cl-States (Section 3.1.3), and then select a configuration to satisfy the user provided constraint. This constraint is defined as either delivering a minimum performance, or not violating a power target. Therefore, to satisfy either of these constraints, REPP provides a dynamic configuration selector.

3.4.3.1 REPP Configuration Selector

Modern data centres have power constraints (e.g., power capping) and run multiple application instances with different performance constraints. This requires an algorithm to select a configuration per core such that the performance and power constraints are met per core and/or per application, given by ω. To ensure that these constraints are met, REPP dynamically selects a configuration per core every 250 ms by performing a linear search for the DVFS state that is the nearest to the given constraint; next, REPP selects the Cl-State for the given DVFS state that satisfies the constraint. This selected configuration, DVFS state, Cl-State, is used to ensure that the constraint is met for each interval.

Figure 3.12 demonstrates a configuration selector to satisfy a user defined power or performance constraint of 40 %. We build one such model to predict performance or power per core. The single-core models predict performance and power for each

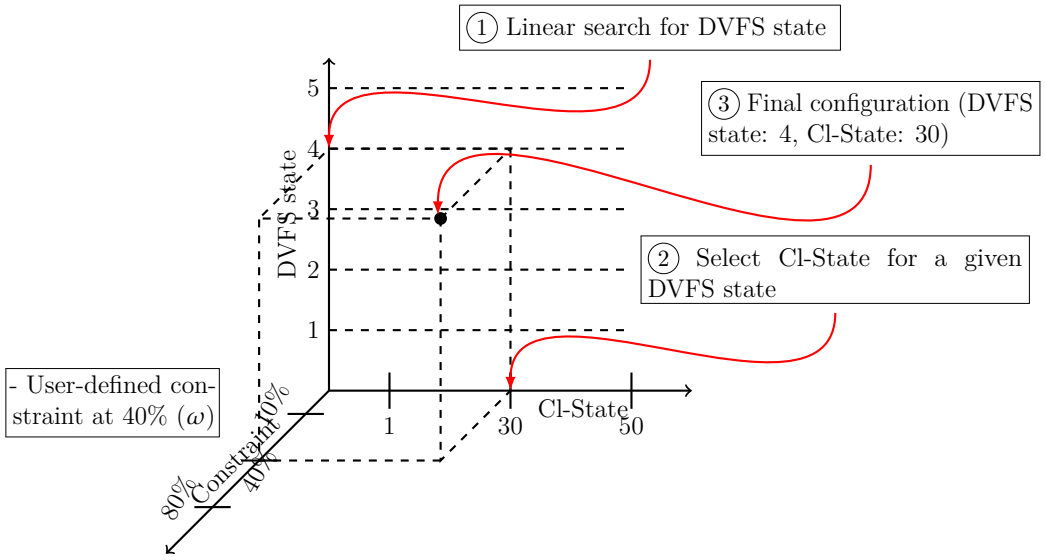


Figure 3.12 REPP configuration selector. An example of the configuration selector to satisfy a user-defined constraint of 40 % for either power or performance for a single-core.

configuration available in the 2-D space of DVFS state and Cl-State. We repeat the aforementioned process for each core at the same time.

In our study, the performance and power constraints are given at a system level. These constraints are distributed homogeneously across all cores. For example, if an AMD server can only consume 600 W, that power constraint is distributed 25 % per core, allowing each core to consume 150 W (it would be 50 % on ARM). In this aforementioned example, ω is 150 W. At runtime, for the spawned applications, REPP samples application behaviour periodically and uses the models built to predict performance and power at all configurations. REPP then selects the configuration for each interval to satisfy the local constraint.

3.4.3.2 Experiments

We perform two types of experiments: one for validating the power capping mechanism and the other for delivering a minimum performance. We define two input parameters: (a) frequency of change, and (b) χ, which represents load or power. The average load offered by the applications is constant between two load changes, which can occur every **load_change interval** (1, 6, or 9 seconds), based on a **change_factor**, as follows. Load starts at a minimum, and varies by multiplying load by change_factor τ until it reaches a maximum load χ_{max}; thereafter, the load is multiplied by the negative value of change_factor until it reaches the minimum χ_{min}.

The values of change_factor tested were 20 % (**Low**), 35 % (**Mid**), and 50 % (**High**). The minimum load is defined as the sum of smallest IPS for all four applications running at minimum frequency; similarly, maximum load is the sum of highest IPS for all four applications at maximum frequency. In another set of experiments, we change the power consumed by the workload similar to the load offered by the workload.

Mathematically, the experiment conducted for a load_change interval 1 can be represented as follows: In Equation 3.7, ψ represents the number of datapoints before the maximum load/power (χ_{max}), and in Equation 3.8, $\chi(\kappa)$ shows the datapoint κ in the sequence.

$$\psi = \left\lfloor \frac{\log(\chi_{max}/\chi_{min})}{\log(1+\tau)} \right\rfloor \tag{3.7}$$

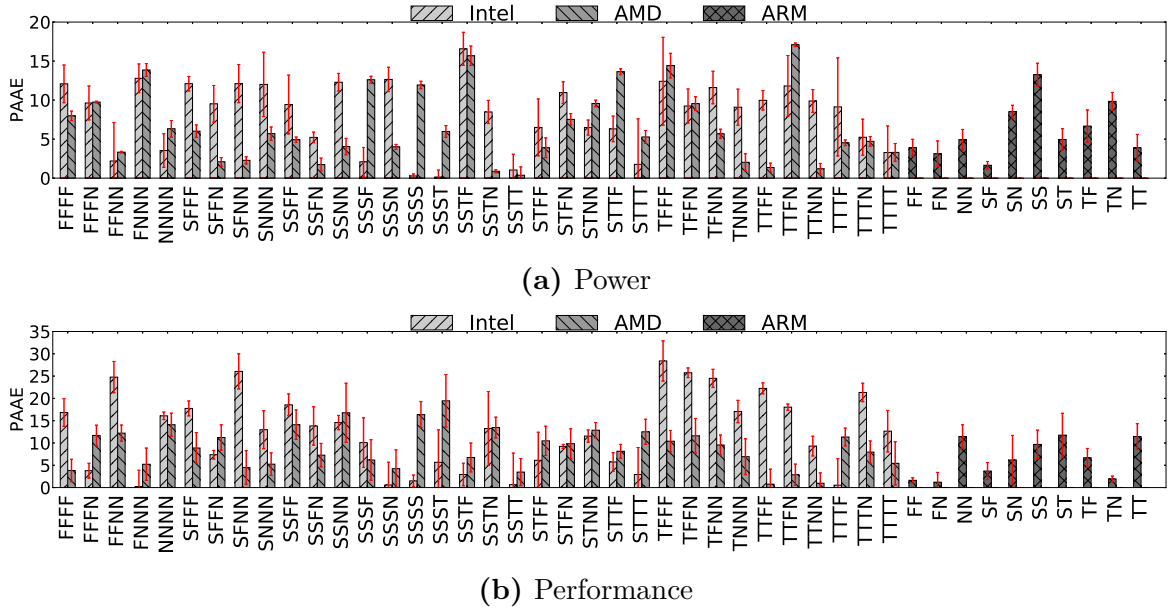
$$\chi(\kappa) = \begin{cases} \chi_{min} \times (1+\tau)^{n} & 0 \leq n \leq \psi \\ \chi_{max} \times (1+\tau)^{2\psi - n} & \psi < n \leq 2\psi \end{cases} \tag{3.8}$$

The error occurs when REPP selects a configuration that makes the application fall short of the minimum required performance (or exceed the maximum power requirement) in a given mapping interval.

We ran ten experiments for power and ten for performance. Nine of them come from the combinations of the two parameters (frequency of change and load/power) described above; the tenth comes from a **Random** setting within fixed boundaries of either power or performance. Selecting a broad spectrum of load (or power) and frequency of change allowed us to validate REPP across multiple combination of configurations at runtime.

Evaluation of the Multicore Models with Constraints

We present the results, and evaluate REPP when predicting performance and power on multicore processors by selecting a configuration in a single step to meet power and performance requirements for 35 multiprogrammed workloads across 20 experiments (ten for power and ten for performance). The prediction error is computed over a period of 400 seconds for all DVFS states and Cl-States. The error metric indicates that the constraint was violated by that amount, which occurs when REPP selects a configuration that makes the application fall short of the minimum required performance (or exceed the maximum power requirement) in a given mapping interval.

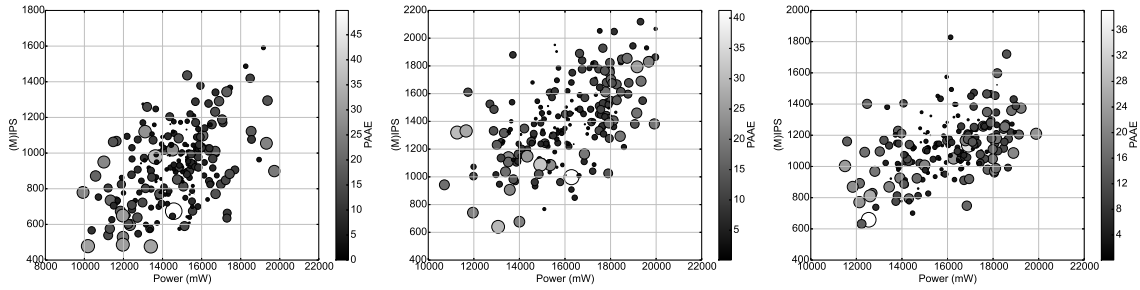


(a) Power

(b) Performance

Figure 3.13 Average PAAE for REPP with multiple constraints. Average PAAE when predicting power (3.13a) and performance (3.13b) for all workloads under Low, Mid, High and Random change_factors in multicore architectures. The error bars represent STDEV across change_factors. The x-axis shows multiprogrammed workloads.

Figure 3.13 shows the average PAAE for each workload when meeting the performance (ARM 7.1 %, AMD 9.02 %, and Intel 7.1 %) and power (ARM 6.0 %, AMD 6.6 %, and Intel 8.1 %) requirements in ten experiments across architectures. The error bars represent the STDEV across ten experiments for power and performance, which is less than 5.3 % for each workload. We focus on analysing those workloads with an error greater than 10 % in both architectures. When predicting power, we observe an error of 13.8 % (654.86 mJ) on AMD for workload `FNNN` that contains *ep.C*, which has very high activity ratio in BPU (4.2542) and FE (13.752). Similarly for workload `TTFN`, Intel has a higher error 11.8 % (11.34 mJ) than AMD, because on Intel we run two instances of *radix*, whereas on AMD we run a single one. Recall that the multiprogrammed are generated based on the methodology described in Section 2.6.1, therefore we can not control the number of instances in a given workload. When predicting performance, we observe an error of 26.0% for `SFNN` on Intel because we run *lu_ncb*, which has non contiguous blocks of memory and the activity ratio in LLC is higher relative to other benchmarks. Similarly, workloads `TTTF` on AMD and `TTFN` on Intel have a power prediction error of 11.3 % and 18.0 %, respectively, because *radix* is part of the multiprogrammed workloads. The maximum performance error on AMD, ARM and



(a) SSTN **(b)** FFFN **(c)** STFN

Figure 3.14 PAAE for workloads SSTN, FFFN and STFN on Intel. Performance and power prediction error with change_factor High.

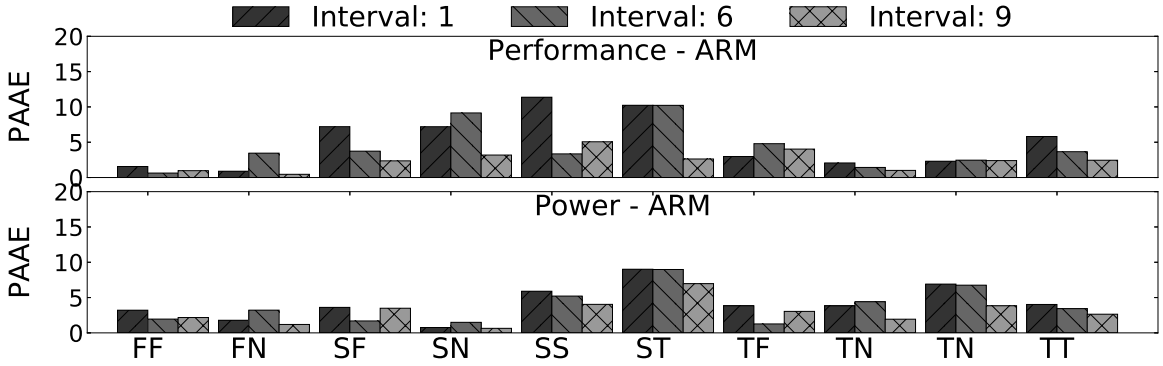
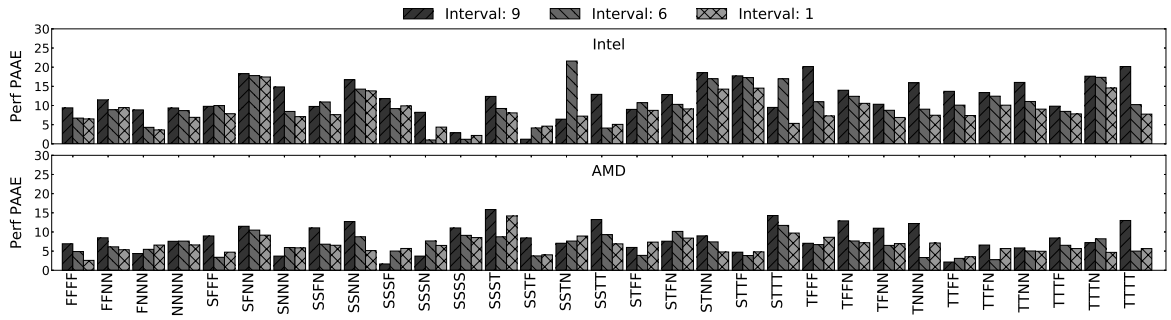


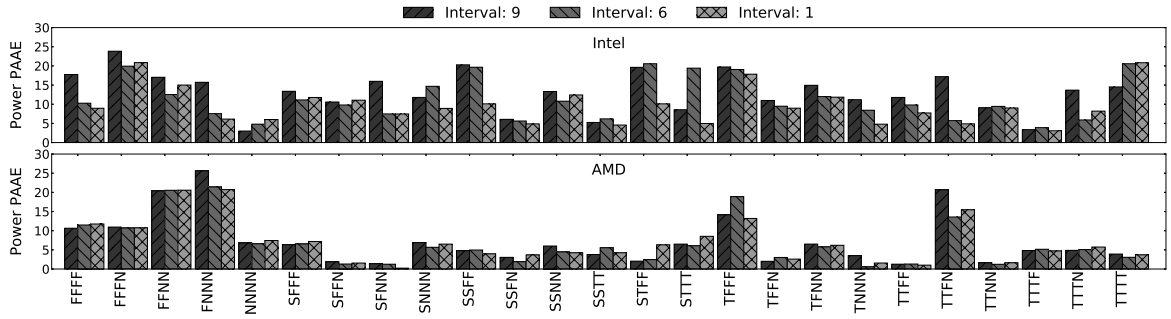
Figure 3.15 Average PAAE on ARM under different load_change intervals. Average PAAE when predicting power and performance for all workloads under different load_change `intervals` for change_factors Low, Mid and High. The x-axis shows multiprogrammed workloads.

Intel are 19.4% (769 MIPS for `SSST`), 11.7 % (5389 MIPS for `ST`) and 28.4 % (13611 MIPS for `TFFF`). Similarly the maximum power error AMD, ARM and Intel are 17.0 % (101.72 mJ for `TTFN`), 13.3 % (10 mJ for `SS`) and 16.6 % (37.24 mJ for `SSTF`), respectively.

Figure 3.14 represents the performance on y-axis and power on x-axis for multiprogrammed workloads SSTN, FFFN, and STFN on Intel architecture with *change_factor* High. The radius of each circle is the maximum prediction error, either power or performance (that is, $radius = max(\text{PAAE}_{\text{power}}, \text{PAAE}_{\text{perf}})$). There exists multiple grayscale grading from low PAAE (black) to high PAAE (white). Although the maximum PAAE shown is at the 50% mark (the grayscale in Figure 3.14a is up to 50%), the number of error predictions with PAAE greater than 30 % is less than ten. For SSTN, FFFN and STFN, the average error when predicting power (and performance) of 9.7 % (3.3 %), 8.7 % (9.4 %) and 8.4 % (6.7 %). This behaviour is observed across all workloads.



(a) Performance



(b) Power

Figure 3.16 Average PAAE on Intel and AMD under different load_change intervals. Average PAAE when predicting power and performance for all workloads under different load_change `intervals` for change_factors Low, Mid and High. The x-axis shows multiprogrammed workloads.

Figures 3.16 and 3.15 show the average PAAE for each workload under different load_change `intervals` on ARM and Intel/AMD when predicting performance (Figure 3.16a) and power (Figure 3.16b). Figures 3.15 and 3.16 are separated because ARM has different number of cores. Across the three architectures, faster (interval=1) load_changes have 3.5 % higher error compared to slower (interval=9) load_changes because of aggressive changes in load in short burst cause numerous changes in configuration, thus leading to a higher error. On the other hand, slower load_changes lead to fewer changes in configuration, and have stable phases in application behaviour. Fortunately, data centres seldom want to jump from 400 W to 600 W every second [73, 39]. Table 3.11 summarises the result obtained when predicting power and performance for load_change interval 1 s, 6 s and 9 s.

Enabling Power/Performance Capping. Determining a configuration to meet the minimum performance (or not exceeding power consumption threshold) is usually done in an iterative fashion (as in the RAPL driver on Intel processors [78]). A feedback

Table 3.11 Average PAAE for load_change intervals. Power and performance error for load_change intervals 1, 6 and 9 on Intel, AMD, and ARM.

	Power			*Performance*		
Interval	1 s	6 s	9 s	1 s	6 s	9 s
Intel	11.1 %	7.3 %	5.6 %	11.4 %	10.6 %	8.7 %
AMD	7.2 %	6.9 %	6.7 %	8.5 %	6.5 %	6.5 %
ARM	4.2 %	3.8 %	2.9 %	5.1 %	4.6 %	2.4 %

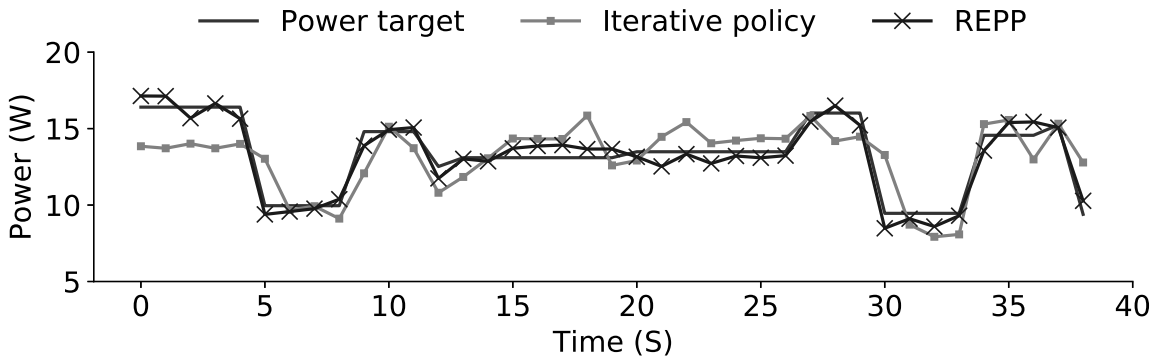


Figure 3.17 Responsiveness to power change on Intel. Responsiveness to power capping for workload SSTT with constraint Random.

control loop is often used to determine the configuration. If the power usage is above a certain threshold, the configuration is lowered. On the other hand, if the power usage is below a certain threshold, the configuration is increased to improve performance. In contrast, we provide a single-step mechanism to select configurations.

Figure 3.17 shows the responsiveness to (dynamic) power capping for workload `SSTT`, where the power capping limit is Random on Intel platform in the first 40 seconds. `SSTT` is composed of applications *streamcluster*, *lu.C*, *bwaves* and *soplex*. REPP changes DVFS states and meets the power target in 0.37 seconds on average, which is 3.6× faster than the iterative algorithm (used by Intel RAPL). This time includes sampling interval, latency to predict at all DVFS state and Cl-State and time to change DVFS state. Moreover REPP, provides 6 % higher prediction accuracy.

3.5 Conclusion

This chapter introduced REPP, a procedure to predict performance and power at runtime for single-core and multicore architectures using PMCs. The technique includes

a systematic method to build multi linear regression models parametrised by hardware settings DVFS states and Cl-States. The PMCs chosen to build the models are available across all major architectures.

We have shown that REPP is a scalable power and performance modelling, and prediction technique to meet various user-defined criteria. The models have a fast deployment time (less than 10 hours), involving a limited number of experiments on each architecture. REPP's single-core and multicore models have been validated on each architecture using benchmarks from SPEC CPU 2006, PARSEC 3.0, NAS and SPLASH2x. We perform real-machine experimentation and use existing hardware support to profile the behaviour of application's at runtime.

We validate REPP using several single threaded and multiprogrammed workloads with average errors, respectively, on ARM, AMD and Intel architectures of 7.1 %, 9.0 %, 7.1 % when predicting performance, and 6.0 %, 6.5 %, 8.1 % when predicting power consumption. Moreover, the single-step prediction technique provided by REPP is 3.6× faster than iterative algorithms. We argue that REPP can enable operators to better control power and performance in modern data centres that include server architectures with heterogeneous processing capabilities.

CHAPTER 4

Power and Performance Models with Consolidation

DYNAMIC power management features available in modern operating systems do not allow the current hardware to reduce the power below a minimum operating power consumption. One measure widely deployed across data centres to minimise energy consumption is to collocate workloads on a subset of cores such that the contention for shared resources is minimised [32, 38, 55, 56, 58, 70, 75, 123, 141]. To further minimise the energy consumption, data centres typically shutdown any unused cores that are inactive. This method of shutting down cores is referred to as core consolidation.

In Chapter 3 we presented REPP, a technique that used basic PMCs to estimate and control performance and power at control settings parametrised by DVFS states and Cl-States. The runtime of REPP is based on single-core models and a simple model to estimate the impact of shared resource contention in multicore environments. This makes REPP dynamic and scalable across multiple nodes quickly. Although REPP can ensure strict performance and power requirements are met, the hardware mechanisms deployed do not permit power to drop below a certain threshold when all cores are used. This is the case even in idle periods. To combat this problem, we introduce **REPP-C**: an estimator to predict performance and power parametrised by core consolidation, DVFS states and Cl-States. REPP-C allocates workloads from under-utilised cores to remaining cores using a scheduling technique to improve energy efficiency.

For example, existing contention-aware algorithms have shown significant improvements in power efficiency by spreading the thread contention across different shared resource levels/domains (e.g., core cache or memory controller [55]). Nevertheless, we observe that taking into account both memory demands and core's DVFS state improves energy efficiency of collocated threads significantly. To this end, we propose an algorithm for online thread assignment for homogeneous multicore systems called Frequency and Contention-Aware Thread Collocation Schema (FACTS).

The rest of the chapter is as follows: Section 4.1 introduces the scheduling algorithm, FACTS and Section 4.2 introduces the modelling and prediction technique REPP-C. Next, Section 4.3.2 evaluates FACTS against state-of-the-art schemes Distributed Intensity Online (DIO) [32] and CFS. Thereafter, Section 4.3.3 evaluates REPP-C by validating numerous performance and power constraints. Next, Section 4.4 and Section 4.4 demonstrate the implementation and scalability of REPP and REPP-C, respectively. Finally, Section 4.5 discusses the evaluation.

4.1 Energy Efficient Scheduler

FACTS (Frequency and Contention-Aware Collocation Schema) is a thread scheduler that uses basic PMCs available on most hardware and core's current DVFS state to optimise energy efficiency in multicore architectures by minimising contention for shared resource. The most important research problem to address when scheduling workloads (albeit challenging) at runtime is which threads to collocate and when (workload balancing). Such assignment of threads should take into account that threads could potentially be harmful to each other in the usage of shared resources (e.g., cache, memory).

The aforementioned challenges have been exhaustively addressed in prior works in terms of inter-core [32, 38, 56, 58] and intra-core [151, 30] contention-aware scheduling. Despite prior efforts, our research has shown that considering heterogeneous DVFS states and shared resources contention can improve energy efficiency even in homogeneous multicore processors. This is similar to operating with heterogeneous multicore platforms where threads with low performance demand and high memory access are mapped to the big cores, and high performance demand and few memory access are mapped to the small cores [30, 33, 152].

To optimise multicore systems for energy efficiency, the mapping of threads to core(s) is carried out periodically (reassignment interval) to cope with thread execution phases. The mapping of threads is determined by taking into account the memory bandwidth requirements and the current DVFS state of the core in a multicore system.

Specifically, we aim to minimise the difference in the total number of LLC misses per second in a given memory domain. A memory domain is defined as a set of cores sharing an LLC. To distribute the contention uniformly across memory domains, we assign the thread with the least number of LLC misses (high IPS and low stall cycles), with the thread with the highest number of cache misses (low IPS and high stall cycles). These two threads are collocated to maximise core utilisation, and therefore are allocated to the core with highest operating DVFS state.

4.1.1 Algorithm Description

We introduce the following notation. Let $\mathcal{S}$ be the total number of cores and $\mathcal{P}$ be total number of available DVFS states. $\mathcal{S}_j$ be the total number of cores running at $\mathcal{P}_{\mathrm{i}}$, $j \in \mathcal{S}$ and $i \in \mathcal{P}$. M be the set of all cores at different DVFS states $\mathrm{M} = \mathcal{S}_{1} \cup \mathcal{S}_{2} \cup \ldots \mathcal{S}_{\mathrm{n\text{-}1}} \cup \mathcal{S}_{\mathrm{n}}$.

Let K be the total number of benchmark threads running in the system. Each thread k running on core with DVFS state i has a memory bandwidth requirement of $LLC\_Misses_{\mathrm{i,\,k}}$, where i is the core allocated to the thread at a reassignment interval S.

The pseudo code of FACTS is shown in Algorithm 1. Note that FACTS runs periodically, with a period determined by the reassignment interval of 1 second (empirically determined). The initial thread mappings are assigned by the Linux scheduler, which allocates threads to cores to balance the workload among all available cores in the system [136]. After a reassignment interval, read the DVFS state ofeach core and FACTS has collected enough performance counter data (line 1-4) to determine the memory requirements of the thread.[1] Thereafter, sort the threads in the descending order of last level cache misses (line 5) and sort active core IDs based on the descending order of current DVFS state of the cores (line 6).[2] In lines 7-8, FACTS maps threads to cores at runtime (low cache misses to cores with higher DVFS state). Thereafter, the remaining threads, if any, are mapped to the cores in reverse order to the cores, that is, high cache misses to the core with high DVFS state and low cache misses (line 10-12). This mapping given by FACTS ensures that the total contention is spread uniformly amongst all cores.

Algorithm 1 FACTS for K threads

```
1  while n < S do                          ▷ Where S is total number of cores and n ∈ S
2      Read the DVFS state
3      for all threads mapped to that core do
4          Read performance counter                                  ▷ LLC misses
5  Sort threads in descending order of cache misses                  ▷ LLC_Misses array
6  Sort cores in the descending order of DVFS state                  ▷ COREID array
7  while n < S do
8      Assign thread with LLC_Misses[n] to core with COREID[n]
9  Set a = 0, c = 0;
10 for all thread not assigned to any core do                        ▷ that is, (K > S)
11     Assign thread with LLC_Misses[S-c] to core with COREID[a]
12     a = a+1; c = c+1;
```

[1] The core's DVFS state was fixed using the scaling governor userspace

[2] An active core is defined as a core with a benchmark thread running.

4.2 Multicore Modelling with Consolidation

In this section, we introduce REPP-C, a power and performance modelling and prediction technique parametrised by DVFS states, Cl-States and core consolidation. In building such models, we have identified four important research challenges: (1) How to consolidate cores? (2) Which thread should be collocated to minimise shared resource interference? (3) Determine the impact on power consumption, and performance degradation? (4) What is the prediction computational latency? We address the following issues separately.

Core consolidation. Traditionally core consolidation has been implemented by turning off a core [153]. Turning off or on a core can be accessed using the ACPI module in Linux, and these operation take a few microseconds to complete. Unfortunately, on our Intel architecture, invoking a function to restart cores at frequent intervals (250 ms) froze our system due to `mce_restart()` race with CPU hotplug operation [154]. Therefore, we implemented core consolidation by allowing a core to enter a deep sleep state mode (C-State). More information on C-States is given in Section 2.1. We verify a core's C-State residency by probing the `MSR` registers (Section 4.4) to ensure power measurements are accurate.

Impact on performance and power. We understand the impact of core consolidation on performance and power by running two experiments on an Intel architecture. First, we run four instances (let's call them App-0, App-1, App-2, and App-3) of the SPEC workload *astar* on four cores, where each instance is pinned to a separate core (that is, no time sharing). Next, we run four instances of the same workload on three cores, where the first two instances (let them be App-0 and App-1) are run on the first two cores (that is, no time sharing), and the remaining two threads timeshare the third core. The fourth core is idle, that is, without any external workload.

Intuitively, it is expected that core consolidation should impact of performance and power of the entire system. Contrary to our intuition, Figure 4.1 shows the impact on performance (the first four bars) and power (the last bar) due to core consolidation; This figure demonstrates that the influence of core consolidation on performance and power is only on the collocated threads, whereas the remainder of the threads show minimal changes.

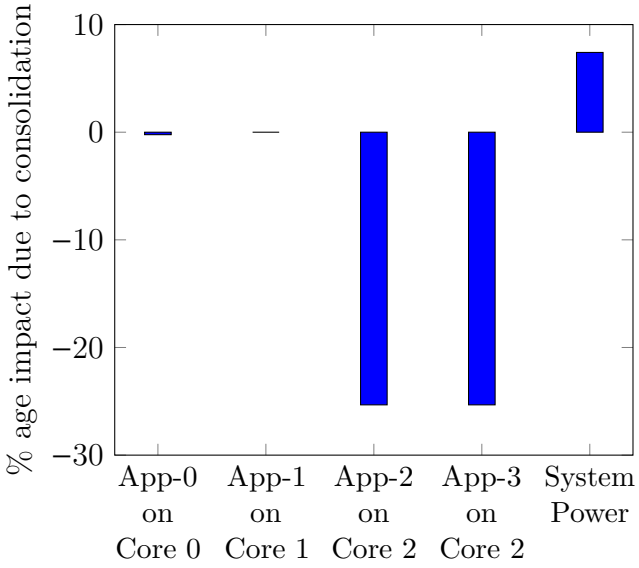


Figure 4.1 Impact on performance and power due to core consolidation. The first four bars show the performance loss and the latter shows the power gain when running four instances of the SPEC benchmark *astar* on four cores and three cores on an Intel architecture.

Modelling methodology. We model the impact of core consolidation as follows. At first, we deploy REPP to meet user-defined performance and power constraints for the training workloads, assuming there exists a inactive core. The workloads are scheduled using FACTS assuming there exists an inactive core, to mimic the impact of core consolidation for REPP. During this procedure, we gather performance statistics, power measurements and user-defined constraints every interval into a log-file.

To build the models, we use the traces collected in the log-file to compute the difference between the constraint at a given time and value obtained through PMCs (for performance) or power meter (for power). This difference in power or performance, is subtracted for each prediction on a per thread level for the pair of collocated threads. Finally, we build one model for performance and another for power.

(1) Set performance and power constraints exclusively for each workload, consecutively (Section 3.4.3.2).

(2) Spawn the training workloads consecutively using the scheduling policy FACTS (Section 4.3.1).

(3) Ensure there exists at least one inactive core. The core with the highest number of LLC misses is the inactive core.

④ REPP selects a DVFS state and Cl-State per core to satisfy the constraint (Section 3.4.3.1).

⑤ Using PMCs read at current configuration, compute activity ratios for private L1, private L2, LLC, and MEM for each thread.

⑥ Compute difference between the PMCs read (or RAPL register) and per thread performance (or power) limit for each thread that is collocated in the workload. This models the impact due to core consolidation on the co-runners.

The aforementioned method is followed for each workload for a period of 300 seconds.

$$core\_consolidation = \sum_{i=0}^{comps} (\Delta_i \times AC_i) + constant \tag{4.1}$$

Equation 4.1 represents the multi linear regression model to predict the reduction in performance or power consumed due to core consolidation, where Δ_i represents the coefficient to be learnt and AC_i represents the activity ratio of the individual components (specifically L1, L2, LLC, and MEM).

The initial thread-to-core assignment is carried out statically, and the cores are set to the lowest DVFS state. Then, given the constraint, REPP predicts performance and power, and determines the potential configuration that satisfies the constraint. Thereafter, FACTS selects the potential threads to be collocated on the system, assuming there exists at least one inactive core. In our case study, the core with the highest number of LLC misses is the potential inactive core. As shown in [32], collocating a thrashing workload with a non-thrashing workload reduces the impact on shared memory footprint/cache pollution. This is because, a thrashing workload has higher memory stalls cycles compared to non-thrashing workloads. Consequently, REPP-C predicts performance or power for each of the collocated threads by subtracting the value obtained in Equation 4.1 from the original prediction (REPP). Similar to REPP, REPP-C also selects a potential configuration. In section 4.2.1, we describe the rules to determine which configuration (REPP or REPP-C) to select.

4.2.1 Is consolidation worth it?

To ensure that the user-defined power or performance constraint is satisfied at runtime, a potential configuration is selected from either REPP or REPP-C periodically (250 ms). As a precursor to not violating performance guarantees and power budgets, we select a

configuration based on (a) Satisfying the system level constraint target; (b) Fulfilling the per thread constraint.

To determine if consolidation is worth it, we predefine two conditions, and the mechanism, that is used in the system is as follows:

Condition (**C1**): Compute the error between total predicted output and constraint for REPP and REPP-C.

Condition (**C2**): Compute the error between the per thread predicted output and the per thread constraint.

(1) Select the configuration that satisfies performance and power per thread for REPP and REPP-C.

(2) If both conditions (C1, and C2) are (approximately) zero for both REPP and REPP-C, then select the configuration as given by REPP. This is because it delivers a higher energy efficiency than REPP-C.

(3) If the constraint is met (approximately) with C1 for REPP, but C2 for REPP-C or otherwise. Then, select a configuration from either REPP-C or REPP for which the C2 has lesser error.

(4) Select DVFS state, Cl-State per core as given by REPP or REPP-C.

Table 4.1 FACTS workload composition. Composition of workloads from the SPEC CPU 2006 benchmark suite for validating FACTS. The workloads were generated based on the method described in Section 2.6.

Workload	Set of benchmarks	Categories (Section 2.6)
8C	Bzip2, Calculix, Gobmk, Soplex	NNNT
	CactusADM, Astar, Tonto, Povray	FTNN
2M6C	Astar, Calculix, CactusADM, Gobmk	TNFN
	Soplex, Povray, Tonto, Bzip2	TNNN
4M4C	Astar, Milc, CactusADM, Lbm	TSFF
	Gobmk, Tonto, Povray, Bzip2	NNNN
6M2C	Xalancbmk, Wrf, GemsFDTD, Astar	TSFF
	Lbm, Tonto, Calculix, CactusADM	FNNF
8M	Astar, Tonto, CactusADM, GemsFDTD	TNFT
	Lbm, Milc, Bzip2, Mcf	FSNS

4.3 Evaluation

This section describes the workloads used to train and validate REPP-C. Next, we evaluate FACTS against state-of-the-art schedulers and REPP-C.

4.3.1 Workloads

In validating FACTS, we run `twice` as many threads as cores available, to determine the effectiveness of the collocation technique. In evaluating REPP-C, we launch as many benchmark threads as cores available (as in prior works [39, 125, 155]).

For each evaluation, we run several different combinations of workloads concurrently; each workload is a mix of applications ranging from compute intensive to memory intensive [104]. As in prior work [30, 31, 39, 55–57, 141], we select a subset of benchmarks from the SPEC suite to represent the effectiveness of our technique. The process of generating a multiprogrammed workloads is described in section 2.6. However, for REPP-C, we categorise the workloads into two groups: the first group, compute intensive contains workloads from insensitive (N) and Cache-Friendly (F); the second group, memory intensive has workloads from Cache-Fitting (T) and Streaming/Thrashing (S).

Table 4.1 shows the different workloads used in our experiment to evaluate FACTS. In particular the workloads are: (8C) 8 compute intensive threads, (2M6C) two memory and six compute intensive, (4M4C) four memory and four processor intensive, (6M2C) six memory and four compute intensive and (8M) 8 memory intensive workloads.

Table 4.2 refers to the workloads used in our environment for validating REPP-C. In particular, the **boldfaced** workloads are used for training the model: (**4C**) four compute intensive, (**2M2C**) two memory-intensive and two compute-intensive, (**4M**) four memory intensive. The remaining workloads are used for validating: (3C1M) three compute- and one memory-intensive, (1C3M) one compute-intensive and three memory-intensive, and (4R1, 4R2) two randomly chosen set of workloads consisting of four benchmarks each.

4.3.2 Energy Efficient Scheduler

We evaluate FACTS against two state-of-the-art algorithms: DIO [32] and CFS. DIO is a thread scheduling technique directed towards reducing contention for shared resource in components like cache and memory. DIO dynamically schedules threads by spreading contention equally among different memory domains, taking into consideration the cache intensity of the threads. On the other hand, Linux scheduler time shares the

Table 4.2 REPP-C workload composition. Composition of **training** and validation workloads from the SPEC CPU 2006 benchmark suite. The boldfaced workloads used as training workloads. The workloads were generated based on the method described in Section 2.6.

Workload	Set of benchmarks	Categories (Section 2.6)
4C	Calculix, Gobmk, Bzip2, Tonto	NNNN
2M2C	Lbm, Astar, CactusADM, Calculix	FTFN
4M	Astar, Mcf, Lbm, Milc	TSFS
3C1M	Povray, Bzip2, Calculix, Soplex	NNNT
1C3M	Tonto, Lbm, Milc, Soplex	NFST
4R1	Astar, Bzip2, Mcf, Soplex	TNSS
4R2	Lbm, Bzip2, Calculix, GemsFDTD	FNNT

core uniformly among all application without considering the compute intensity or memory intensity of workloads.

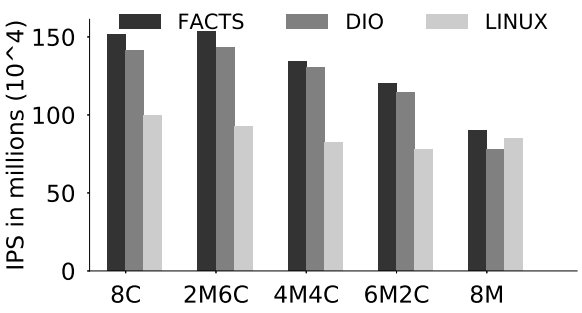
In evaluating FACTS, DIO and Linux scheduler, the active cores are set to all possible combination of DVFS states in consecutive runs for each workload available. This process is repeated across each scheduler consecutively. To compare the different scheduling policies, we analyse the results obtained for the total energy efficiency, activity in LLC and the total performance of all workloads for the first 1000 seconds of execution of the workload. Also, we study the pair of threads that are executed on a given core at a particular DVFS state.

Previous research [30–32, 55, 140] has shown that contention for shared resources in cache hierarchy can impact performance of the threads drastically. Miss penalty, measured in computational cycles, is defined as the ratio of memory access time to the inverse of frequency. Therefore, the power consumption increases as miss penalty to access the lower levels of caches increase.

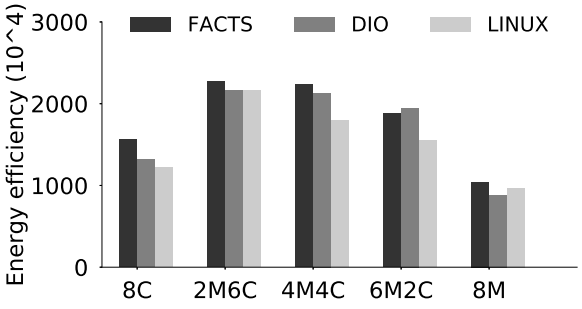
Specifically, we show three main reasons why FACTS improves over DIO and Linux in performance and energy efficiency when run for a fixed time quantum of 1000 s. First, one might intuitively assume FACTS runs slower because it has a higher number of thread migrations which might cause high overhead and make the workloads take longer when dynamically mapped between cores, thus having less performance. Towards that, Figure 4.2a shows that the total performance of FACTS improves over DIO by 8.25 % (mean) and Linux by 37.56 % (mean). Second, since FACTS runs threads faster than DIO, we assume that it might have a higher energy consumption. However, FACTS

schedules workloads based on the demand for memory bandwidth and core DVFS states. Therefore, the threads which stall for memory are migrated to cores with lower DVFS state, thus resulting in a small idle time for compute intensive threads and small power consumption while stalling for memory. Towards that, we looked at the energy efficiency (MIPS/watt – performance/watt) and show that FACTS improves over DIO by 6.2 % (mean) and Linux by 14.17 % (mean) (see Figure 4.2b). Third, since FACTS migrates threads which stall for memory to cores with lower DVFS state, and DIO does not, it improves the performance of the co-scheduled threads and reduces the LLC misses by 4.87 % (mean) over DIO and 41.70 % (mean) over Linux (see Figure 4.2c).

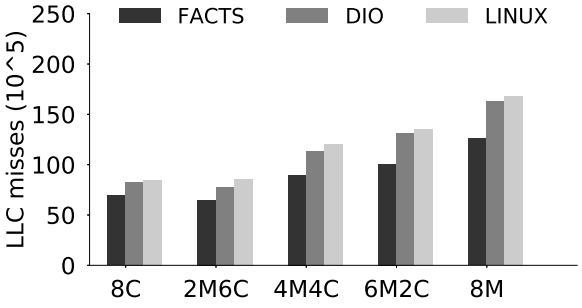
In addition, while executing workload 4M4C at DVFS states 1.6 GHz, 2.4 GHz, 0.8 GHz and 0.8 GHz on cores 0-3, respectively. DIO schedules *cactusADM* and *Tonto* on core with DVFS state 0.8 GHz for 82 % of the execution time. We note that *cactusADM* is a mid memory intensive, whereas, *Tonto* has low memory activity. On the other hand, FACTS schedules the pair of threads, *Astar* and *Bzip2* are allocated for 10.65 %; *Astar* and *cactusADM* for 15.2 %; *Bzip2* and *cactusADM* are allocated for 74.15 % of the execution time on the core with DVFS state 0.8 GHz. Specifically in this scenario, we improve over DIO by 6.84 % in performance.



(a) Performance (MIPS)



(b) Energy efficiency (MIPS/Watt)



(c) LLC misses

Figure 4.2 Performance evaluation for FACTS against DIO and Linux. Performance (4.2a), Energy efficiency (4.2b) and LLC misses (4.2c) for all workloads over all frequency set combinations for *FACTS*, *DIO* and Linux scheduler with eight threads.

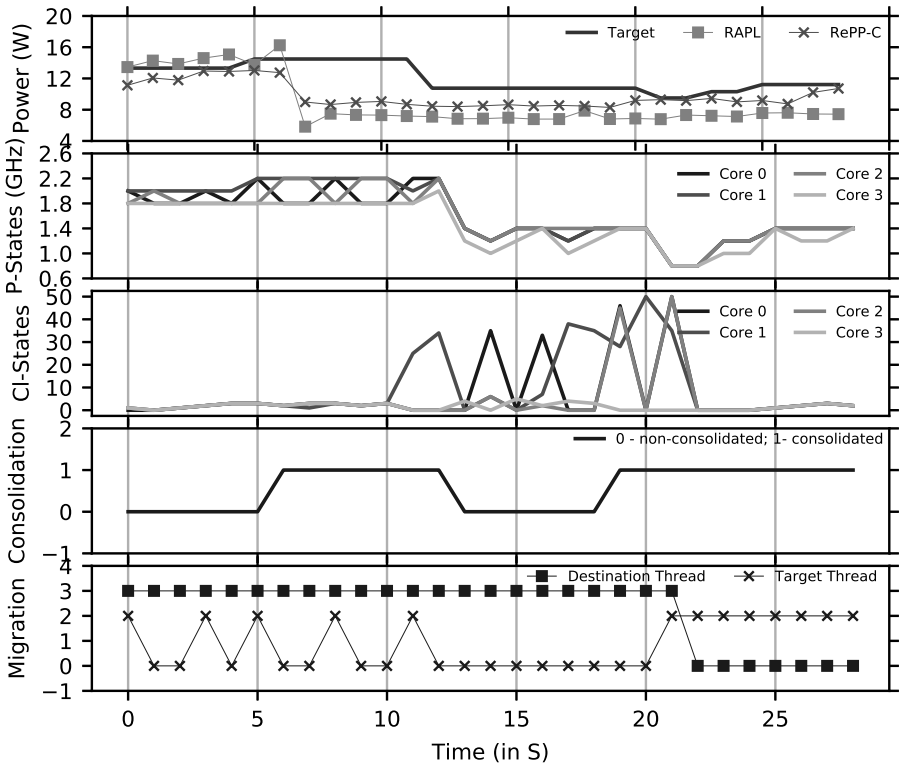


Figure 4.3 Power prediction for random workload. Runtime view of the power prediction for workload 4R1 under change_factor Randomised. In the bottom graph, destination and target threads refer to the thread with the least number and highest number of LLC misses, respectively. The numbers on the y-axis, in the last subplot, represent the threads in the same order as in Table 4.2.

4.3.3 Multicore Models including Consolidation with Constraints

In validating REPP-C, we launch the multiprogrammed workloads consecutively using FACTS, and show an example of the prediction technique to satisfy the constraints (see section 3.4.3.2). The constraints are satisfied by selecting a configuration from either REPP or REPP-C as described in section 4.2.1.

Figure 4.3 shows an example of the power prediction technique implemented on our system for the first 20 seconds of execution for the workload 4R1 with constraint Random. From top-to-bottom, the first graph represents the power as measured using RAPL, constraint (Target) and the prediction made using REPP-C. The second and third graph show the changes in DVFS states, Cl-States for all cores in response to changes in constraint along the course of the execution. The fourth graph shows, if REPP (non-consolidated) or REPP-C (consolidated) is deployed. Finally, the fifth graph

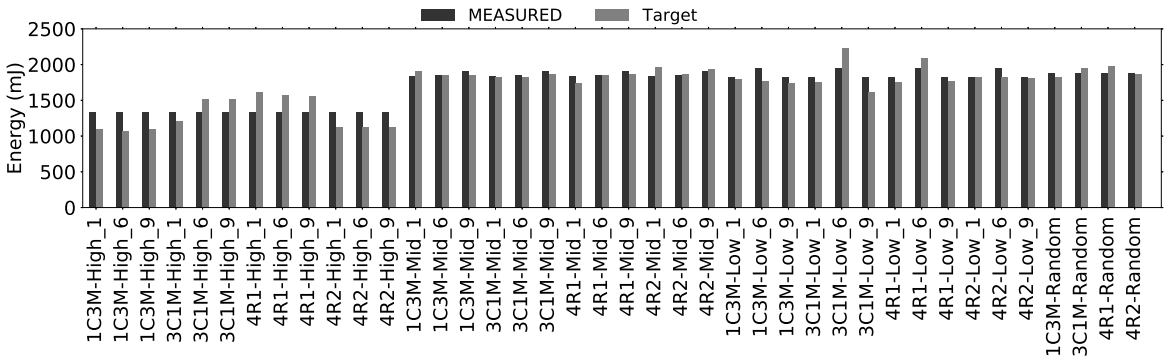


Figure 4.4 Power prediction for multiprogrammed workloads with REPP-C. Energy consumed (mJ) across all workloads under change_factors High, Mid, Low and Random. The y-axis is read as MEASURED *using RAPL* and constraint over time (*Target*)

.

shows the co-runners for the three active cores for REPP-C and four active cores for REPP. The numbers on the y-axis, in the last subplot, represent the threads in the same order as in Table 4.2. We highlight two results. First, REPP-C does show the capability to adapt to workloads consisting of multiple thread phases [104], and ensure constraints per-thread are met with low error. Despite the fact that there are phases where REPP-C have relatively higher error, for instance, in seconds 7-10 REPP-C makes a 3.6 W error, the average error is as low as 4.03%. Second, REPP-C and REPP can predict power per thread which native RAPL or PEPP [39] cannot achieve. This shows that the model is accurate enough to capture the real behaviour, is driven by existing performance counters, and, since the computational complexity at runtime is low, it can be used for fine-grain power management.

Figure 4.4 shows the energy consumption in millijoules (mJ) using the power measured using native RAPL register and constraint over time (Target) for four workloads under all constraints (for all load_change and change_factor) in period of 100 seconds. The x-axis represents the type of workload, change_factor and the load_change interval (in seconds). The configuration to satisfy the target is selected using REPP-C.

The energy consumption target is measured as the summation of $\chi(\kappa)$ (as given by Equation 3.8). On average, for all workloads, in under different constraint, REPP-C exhibits an error in prediction of 7.55% or 115.67 mJ. Observe, that the maximum error we incur is 286.4 mJ (in 3C1M-Low_6). Moreover, the error in achieving the given energy target for change_factors High, Mid, Low and Random are 217.22 mJ (16.23%), 41.63 mJ (2.24%), 107.80 mJ (5.69%) and 56.72 mJ (3.02%) respectively.

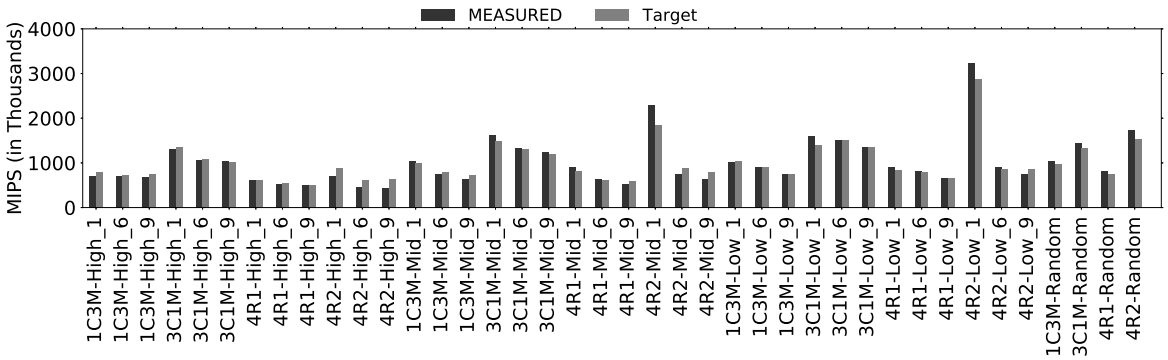


Figure 4.5 Performance prediction for multiprogrammed workloads with REPP-C. Total performance (in Thousands) across all workloads under change_factors High, Mid, Low and Random. The y-axis is read as MEASURED *using PMCs* and constraint over time (*Target*)

.

Figure 4.5 shows the total performance (MIPS) in thousands measured using PMCs and the constraint over time (Target) for four workloads under all constraints for a period of 100 seconds. The x-axis represents the type of workload, change_factor and the load_change interval (in seconds). The configuration to satisfy the target is selected using REPP-C. The total performance target is measured as the summation of $\chi(\kappa)$ (as given by Equation 3.8). We incur an average error of 8.96% and a maximum error of 46.91% (in 4R2-High_9) and a total of 206.19 MIPS. Similar to the energy consumption target, the total performance target is also measured.

Figure 4.4 and 4.5 demonstrate that REPP-C has been validated under multiple constraints, where each bar in the histogram represents a summary of the workload execution, similar to Figure 4.3.

4.4 Implementation of the Modelling and Prediction Technique

REPP and REPP-C are user-level processes running on Linux OS that consist of four main functions: (a) Gather performance statistics (Section 2.3), power meter readings (Section 2.4), and verify core's sleep state residency. (b) Bind tasks-to-cores. (c) Predict performance and power at each DVFS state and Cl-State (with and without core consolidation). (d) Given a constraint, set DVFS state and Cl-State.

We set all inactive cores to the lowest DVFS state and facilitate the core to enter the deepest sleep state. To verify the C-State residency (at sleep states C3, C5, and C7) during core consolidation, the MSR registers [78] MSR_PKG_C3_RESIDENCY, MSR_PKG_C6_RESIDENCY and MSR_PKG_C7_RESIDENCY are read every 250 ms on the Intel processor. The default time quantum to *enter* a C-State is 1024 ns.

In the process of generating the models, we bind tasks to cores via the Linux sched_setaffinity system call. Binding a task to a core ensures that the thread runs on the specified core. Assigning the affinity [136] for a thread bypasses Linux CFS and helps avoid excessive thread migration and provides a more controlled environment. The process of model generation is automatised using an R script in order to be applicable systematically. This is feasible because no manual training is required, showing that with a random set of benchmarks it is possible to generate accurate performance and power models.

The frequency scaling was performed by setting the *CPUFreq governor* to *userspace* and scaling the frequency for individual cores (Kasture *et al.* [60] show that changes in DVFS are in the order of microseconds). On the Intel processor, the MSR register MSR_IA32_PERF_CTL [76, 78] is used to reduce the latency to a single register move. Switching DVFS state using the MSR register overrides the Linux kernel driver acpi-cpufreq. Next, the Cl-State are set using the Intel powerclamp driver [81].

Algorithm Configuration. A sampling interval of 250 ms was empirically chosen for all phases of the algorithm i.e., to generate traces, build models and deploying REPP, REPP-C, and FACTS.

Scalability. Scalability of a prediction algorithm is determined by the computational latency to make predictions and the prediction accuracy. To visualise the scalability

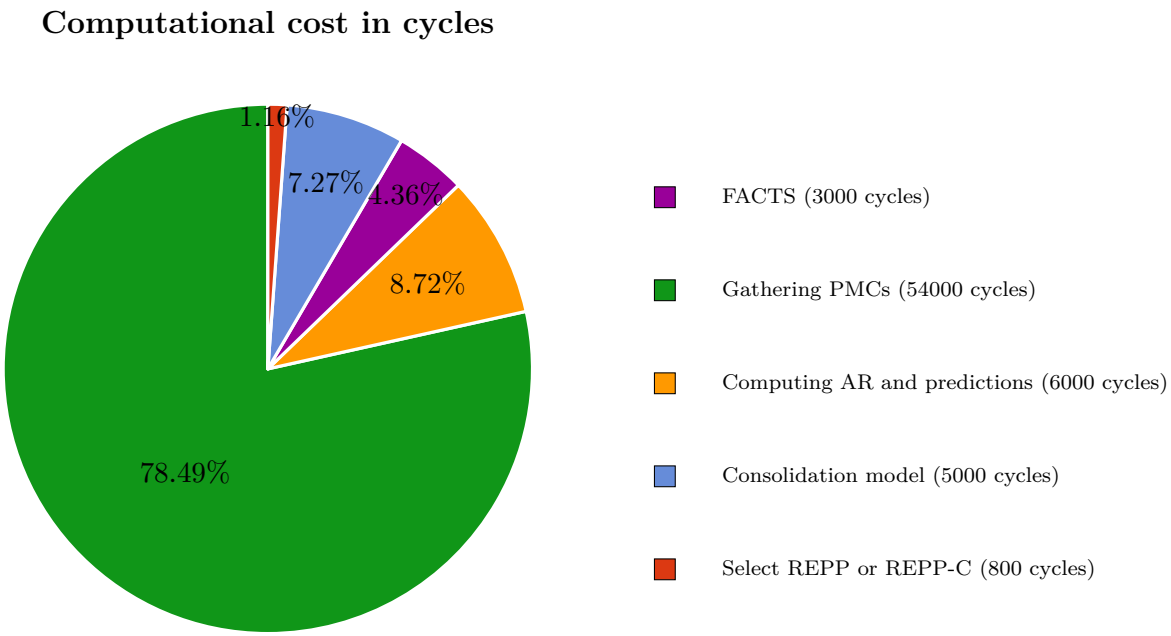


Figure 4.6 Scalability of REPP and REPP-C. The computational overhead measured in cycles (using RDTSC instruction) on the Intel processors is 68880 cycles.

impact, we breakdown the computational cost (in cycles) only for the Intel processor[3], using the RDTSC [156] instruction for the following computations: (1) Cost to read PMCs and compute ARs; (2) Predicting performance and power at all configuration without consolidation (REPP); (3) Predicting performance and power with consolidation (REPP-C); (4) Selecting REPP or REPP-C; (5) Moving to a given configuration. The estimates of computational overheads provided are recorded per evaluation, and is the mean over multiple iterations.

The possible number of configuration on a four core system are 80 billion (450 configuration per core for REPP and REPP-C). However, the overlap in performance and power prediction over these configurations leads to redundancy in computational resources and power. Therefore, we compute only 3600 configurations on a four core system running both REPP and REPP-C. Despite predicting only 0.5 % of the total number of configuration possible, our results have shown that the estimation technique can predict performance and power with an accuracy greater than 93 %.

Figure 4.6 shows the computational overhead (per computation) when predicting performance and power on a four core system running at 2.4 GHz. The total overhead is measured in cycles using the RDTSC instruction is 68800 cycles. The major overhead

[3]We do not expect the computational cost to have a large deviation across processors, as the technique involves only a few mathematical operations and one I/O operation

Table 4.3 Time complexity in predictions. Number of configurations possible per core given the number of cores and time for modelling

DVFS state	2.4 GHz			0.8 GHz		
Time(s)/#cores	1	2	4	1	2	4
0.500 s	900	230888	230888	900	76964	76964
0.250 s	900	115444	115444	900	38482	38482
0.125 s	900	57722	57722	900	19241	19241
0.063 s	900	29092	29092	900	9697	9697
0.031 s	900	14315	14315	900	4772	4772
0.016 s	900	7388	7388	900	2463	2463
0.008 s	900	3694	3694	900	1231	1231

(54000 cycles) is due to the low level operations to gather PMCs using perf. Next, computing the activity ratio and predicting performance and power (6000 cycles). Thereafter, selecting a potential corunners using FACTS (3000 cycles), and applying the consolidation model (5000 cycles). Finally, to select REPP or REPP-C and applying a given configuration (800 cycles). In summary, it takes under 100 cycles per prediction of performance or power.

Table 4.3 shows number of configurations that can be computed within a given time quantum (measured in seconds) at 2.4 GHz and 0.8 GHz on a machine with one, two, and four cores. For instance, the number of configurations on a two core system are 810000. Nevertheless, given a DVFS state (say 2.4 GHz) and a time constraint (0.125 s), the number of predictions that can be computed are 57722, instead of 810000. Also observe that on a machine with a single-core at 2.4 GHz and 0.8 GHz, the number of configurations are the same because, irrespective of the DVFS state the number of configurations are fixed at 450 per core for REPP and REPP-C.

Given the computational overhead, we suggest that our prediction methodology has minimal overhead and is fine-grained. This enables REPP to characterise workload at a per thread level using existing hardware counters to meet various constraints.

4.5 Conclusion

In this chapter, we presented REPP-C, a power and performance estimation model parametrised by the hardware DVFS states, Cl-States, and core consolidation. The technique includes a systematic method to build models to predict performance and power for each core, based on PMCs, and multi linear regression. In the multicore environment with consolidation, the multicore models are complemented by a correction to account for the impact of consolidation. REPP-C implements a scheduling technique to place the threads which were allocated on the consolidated core.

To allocate threads from the consolidated core to another one, we designed Frequency, and Contention-Aware Thread mapping Schema (FACTS). FACTS is an online frequency and contention-aware thread-to-core mapping scheduling policy for multicore systems. FACTS can be implemented on any multicore processor without knowledge of the workload or the target system. Therefore, it is quick and easy to implement user-level Linux scheduler. We compared FACTS with other state-of-the-art schedulers and our results show improvements over all SPEC CPU 2006 workloads. FACTS improves energy efficiency by 6.2 % and 14.2 % over DIO and native Linux scheduler, respectively.

FACTS is integrated in REPP-C to predict performance, and power with workload consolidation. REPP-C's predictions are used to meet user specified constraints for multiprogrammed workloads on multicore architectures. Our results show average errors of 7.55 % and 8.96 % (with 95 % confidence interval) are observed when predicting energy and performance, respectively for numerous multiprogrammed workloads.

This page is intentionally left blank.

Part III:

Addressing scheduling of interactive and batch workloads

There are other Annapurnas in the lives of men.
Maurice Herzog

CHAPTER 5

Hybrid Task Manager for Interactive Workloads

In the previous chapters, we introduced a framework to estimate performance and power consumption of batch workloads using a statistical modelling technique. By contrast, this chapter focuses on an application-level scheduling approach for latency-critical and batch workloads, by taking advantage of processor capabilities (like OS-level DVFS) [157–160] to improve energy efficiency or resource utilisation. Collocating latency-critical with batch workloads is interesting because the former is striving to deliver a service within a certain quantum [22], whereas the latter is aimed at maximising performance.

Many important cloud workload are latency-critical, and require strict levels of QoS to meet SLA requirements [40, 60, 103]. For instance, a web-request must complete within a certain time frame otherwise users are likely to leave. Recent study has shown that marginal delays in QoS (of hundreds of milliseconds) can impact revenue massively [17]. In particular it is important to meet QoS tail latency, such as 95^{th} or 99^{th} percentile of request latency distribution [41].

Recent works [24, 40, 60, 161–163] have shown that traditional power management practices and CPU utilisation measures are unsuitable to drive task management for data centre workloads. This is because prior schemes (like OS-level DVFS) work well to deliver long-term performance for batch workloads, but they can severely hurt the QoS of latency-critical data centre workloads.

As noticed in prior work [24, 40, 60, 161–163], workload management can be very challenging in heterogeneous server systems because an application can experience different behaviour in QoS and resource efficiency depending on specific resource allocation decisions. This requires careful resource characterisation of the running workloads to optimise resource usage. In many data centres, there also is a wish to run both latency-critical and batch workloads. Running both latency-critical and batch

workloads, in this way, increases cluster utilisation during periods of low demand, reducing operational cost and total energy consumption.

Precisely, we introduce **Hipster**, a scheme that manages heterogeneous multicore allocation and DVFS settings for latency-critical workloads given QoS constraints, while minimising system power consumption. In addition, Hipster allows collocation of latency-critical and batch workloads in shared data centres to maximise the data centre utilisation. In such scenarios, the resources allocated to latency-critical workloads are just enough to meet the QoS target, and the remaining resources are allocated to throttle the batch workloads.

The major contributions of this chapter are:

(1) We present **Hipster**, a hybrid management solution combining heuristic techniques and reinforcement learning to make resource-efficient allocation decisions, specifically deciding the best mapping of latency-critical and batch workloads on heterogeneous cores (big and small) and their DVFS setting (Section 5.2).

(2) Hipster is presented in two variants: **HipsterIn** and **HipsterCo**. HipsterIn (for interactive workloads) is targeted towards allocating resources to latency-critical workloads so that the system power consumption is minimised, whereas HipsterCo (for collocated workloads) enables running both latency-critical and batch workloads for improved server utilisation. Both variants ensure that QoS for latency-critical workloads is met.

(3) We carried out real measurement experiments on a 64-bit big-LITTLE (ARM Juno R1) platform along with back-end services such as Web-Search and Memcached. The request generator for each of the back-end services follows a diurnal load pattern typical of production data centres (Section 2.2).

(4) We evaluate Hipster against the only other heterogeneous platform-aware state-of-the-art scheme [24] that dynamically allocates heterogeneous cores to latency-critical workloads. Our results show that HipsterIn outperforms prior work, in energy consumption reduction by 13%, while achieving up to 99% QoS guarantees for the latency-critical workloads. In addition, our results for HipsterCo show that it improves performance by 2.3× compared to a static/conservative policy running batch workloads, while meeting QoS targets for latency-critical workloads (Section 5.3).

(5) We evaluate the ability of our scheme to adapt dynamically to changes in application at runtime. Our results show that HipsterIn can adapt to changes in application, while achieving up to 98% QoS guarantees for the latency-critical workloads. Finally, we present the source code written in Python for Hipster that can be used to deploy different cloud workloads such as Web-Search and Memcached.

5.1 Motivation

Typical web applications experience large variations in user traffic over time.[1] Figure 5.1, for example, shows how the number of queries per second during a web search, a typical load at Google data centre, varies between about 5 % and 80 % of maximum capacity [118, 28, 103]. Similarly, Facebook consistently sees diurnal load variations between 10 % and 95 % of maximum capacity, across multiple server clusters [29, 114].

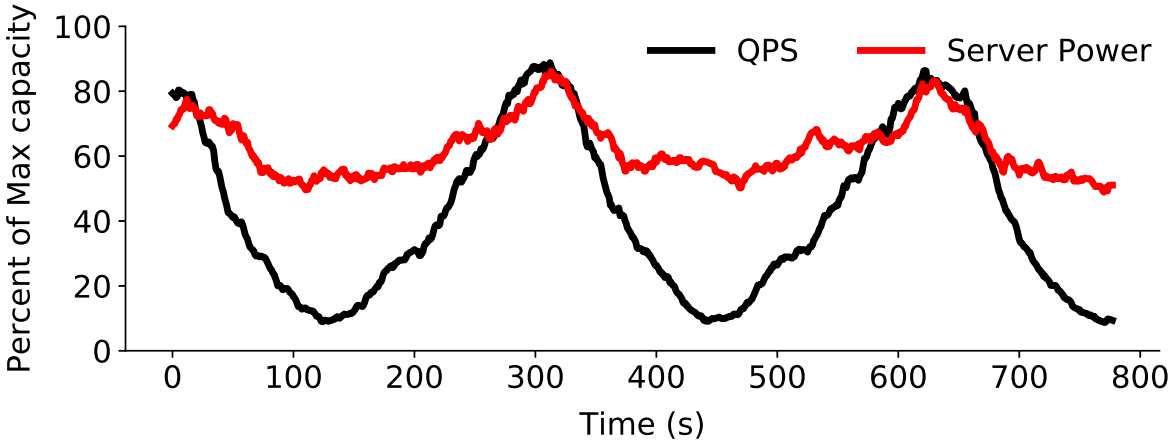


Figure 5.1 Power drawn for a diurnal load [28, 103, 118]. The power consumption is measured for Web-Search running on two big cores of the ARM Juno R1 (64-bit big.LITTLE) platform.

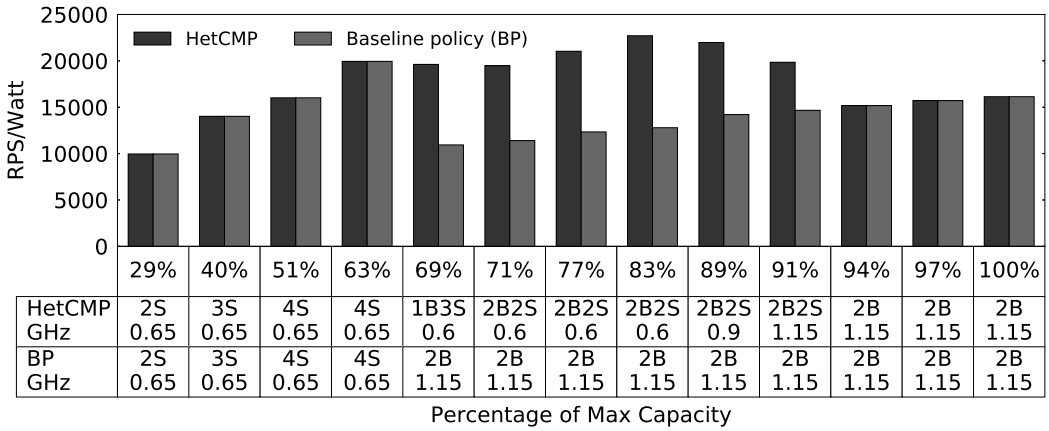
The periods of low server utilisation provide opportunity to reduce data centre energy consumption [24, 40, 161, 166]. As seen in Figure 5.1, although load drops dramatically, power consumption is always at 60 % or above. For this reason, both academia and industry are working towards better energy proportionality; i.e. that the system's power consumption is proportional to utilisation.

There are also opportunities to improve energy efficiency using heterogeneous servers combined with DVFS. Heterogeneous servers can minimise power consumption at low load by deploying small cores, and can provide maximum performance using big cores to meet the QoS target for latency-critical workloads [24, 45, 46].

Mixing different core types with DVFS. Figure 5.2 shows the energy efficiency in RPS/Watt and QPS/Watt when using a state-of-the-art baseline policy [24].[2] We explore a heterogeneous architecture mixing different core types and DVFS (HetCMP)

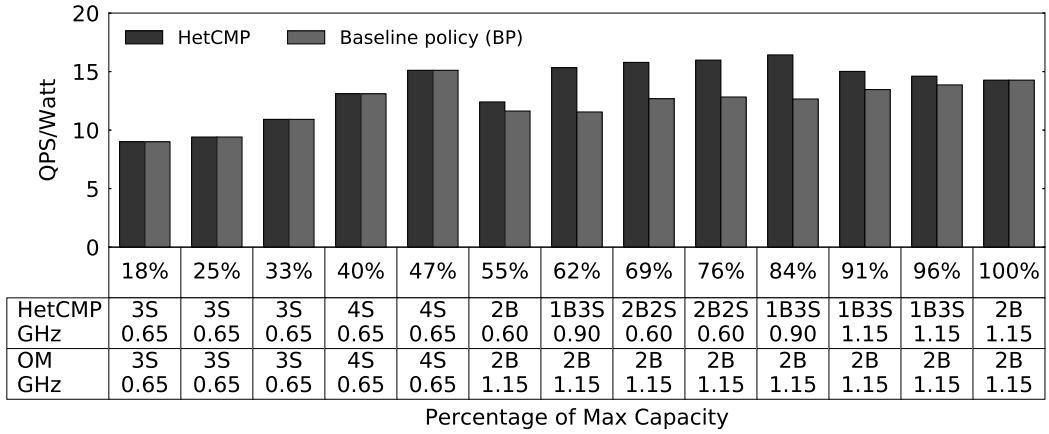
[1] Roy *et al.* [164] and Bronson *et al.* [165] show network bytes over the course of the day for a web-server and memcached [113] at Facebook.

[2] Recall, RPS refers to requests per second and QPS refers queries per second (see section 2.3)



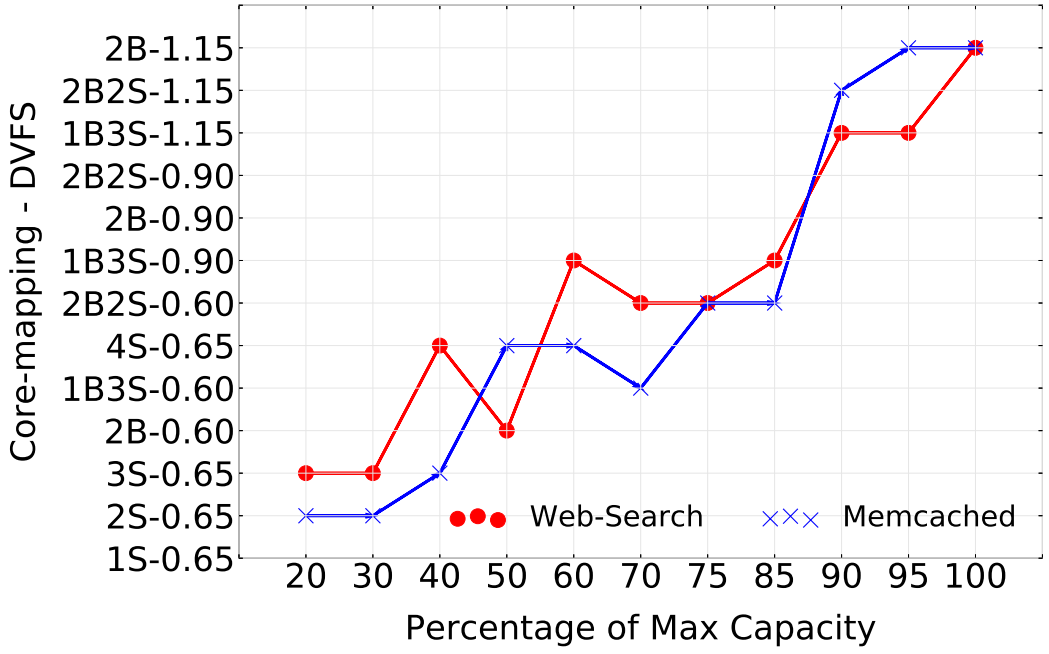
	29%	40%	51%	63%	69%	71%	77%	83%	89%	91%	94%	97%	100%
HetCMP GHz	2S 0.65	3S 0.65	4S 0.65	4S 0.65	1B3S 0.6	2B2S 0.6	2B2S 0.6	2B2S 0.6	2B2S 0.9	2B2S 1.15	2B 1.15	2B 1.15	2B 1.15
BP GHz	2S 0.65	3S 0.65	4S 0.65	4S 0.65	2B 1.15	2B 1.15	2B 1.15	2B 1.15	2B 1.15	2B 1.15	2B 1.15	2B 1.15	2B 1.15

(a) Memcached



	18%	25%	33%	40%	47%	55%	62%	69%	76%	84%	91%	96%	100%
HetCMP GHz	3S 0.65	3S 0.65	3S 0.65	4S 0.65	4S 0.65	2B 0.60	1B3S 0.90	2B2S 0.60	2B2S 0.60	1B3S 0.90	1B3S 1.15	1B3S 1.15	2B 1.15
OM GHz	3S 0.65	3S 0.65	3S 0.65	4S 0.65	4S 0.65	2B 1.15	2B 1.15	2B 1.15	2B 1.15	2B 1.15	2B 1.15	2B 1.15	2B 1.15

(b) Web-Search



(c) State Machine

Figure 5.2 Throughput per watt of baseline policy (BP) [24] vs heterogeneous platform with DVFS (HetCMP). Throughput per watt of Memcached (5.2a) and Web-Search (5.2b) with BP and HetCMP at different load levels along with their respective state machines (5.2c)

running Memcached (Figure 5.2a) and Web-Search (Figure 5.2b) at different load levels. For each policy, among the configurations where the QoS is met at each load level, the configuration with the least power consumption is selected. The table at the bottom of each subfigure shows the configuration selected by HetCMP and our baseline policy. In the configurations of the embedded table, *B* and *S* represent big and small cores, respectively. The configuration space for HetCMP consists of core-mappings (big and small cores) and DVFS combinations for a heterogeneous platform, whereas the baseline policy consists exclusively of either big or small cores at the highest DVFS. The configurations available for the baseline policy are therefore a subset of HetCMP. Experimental details of the ARM platform are given in Chapter 2.

Figure 5.2 raises two main concerns with current state-of-the-art heuristic algorithm [24]. First, Figure 5.2a demonstrates in periods of low load (less than 60 % of max capacity for Memcached), exclusive use of low performance cores at lower DVFS ensures QoS is met while reducing static-power, thus making it an excellent option to use for periods of low load. As the load increases, HetCMP transitions from low performance cores to a best combination of small and big cores at a given DVFS (for instance, 2 big and 2 small cores – 2B2S at 0.9 GHz at 89 % load) to deliver the required latency. On the other hand, the baseline policy transitions directly from low performance cores to high performance cores at highest DVFS to deliver the required latency, thereby increasing energy consumption by 27.74 % (mean). In periods of very high load (more than 90 % for Memcached), exclusive use of high performance cores at higher DVFS ensures QoS is met with better energy proportionality. Similarly results were observed for Web-Search (Figure 5.2b) with up to 25 % (mean) energy savings. The state-machine configuration for Memcached and Web-Search are represented by the blue and red line, respectively, in Figure 5.2c.

In summary, we show that small and big cores are an attractive option for periods of low load and very high load, respectively, while meeting QoS targets at a much lower cost. On the other hand, for intermediate loads, which are generally experienced by data centres during the day [162], harnessing HetCMP provides the opportunity for higher performance at a lower cost.

Exploring workload particularities. We note that prior work [24] relies on a general single heuristic to allocate exclusively big or small cores to workloads. By allowing an arbitrary allocation mix of big and small cores with DVFS, this kind of heuristic can be sub-optimal across diverse applications and architectures (evaluation details in Section 5.3); that is, a single state-machine management (as used in prior

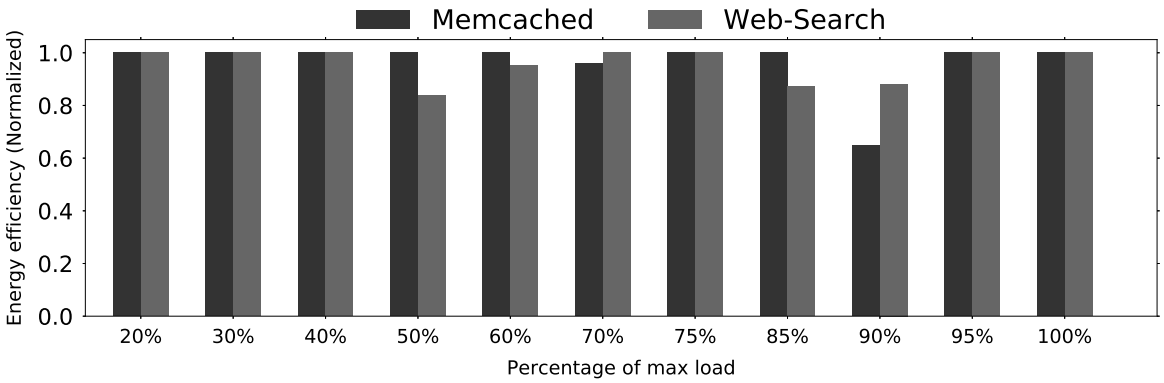


Figure 5.3 Energy efficiency of Memcached and Web-Search with state-machines exchanged. Energy efficiency at various load levels for Memcached while meeting QoS, using the state-machine of Web-Search normalised to the state-machine of Memcached (*lower is worse*); converse for Web-Search.

work) may fail to precisely satisfy the QoS targets given distinct workload characteristics of diverse applications. To illustrate this point, Figure 5.2c shows two distinct/unique state transition mappings that are optimal (throughput per watt) at different load capacities for Memcached and Web-Search.

Figure 5.3 shows the energy efficiency that would be neglected (ensuring QoS is met at different load levels) when using the state-machine built for Web-Search but used for the manage the Memcached workload, normalised to the energy efficiency using the state-machine built exclusively for Memcached; and vice-versa. The state-machine configuration for Memcached and Web-Search are represented in blue and red line, respectively, in Figure 5.2c. Figure 5.3 demonstrates that different latency-critical applications benefit from different state-transition mappings and show improvement in energy efficiency up to 35 % for Memcached (at 90 % load) and up to 19 % for Web-Search (at 50 % load). For instance, at low loads and at very high loads both applications use exclusively small cores (low static power) or big cores (high static power), respectively. However, for intermediate loads, the configurations in the state-transition for Web-Search are not present in Memcached and vice-versa, thus providing minimal to no energy optimisation.

In practical scenarios, each workload has a time-varying load [167] and a QoS target that needs to be met. As shown in Figure 5.2 and 5.3, there exists a unique configuration for each load that optimises energy efficiency. Moreover, the time-varying load presented in two forms: sudden load spikes [41] or gradual load changes [28, 118]. Both these forms present a challenge for a heuristic based approach as it jumps across multiple configurations to meet the QoS target, thereby leading to QoS violations due

to rampant core oscillations. Also, Kasture *et al.* [60] note that core-transitions are far more costly – relative to DVFS changes.

There is a need for application agnostic learning approach that can exploit the energy efficiency benefits of heterogeneous architectures and DVFS features, and can deal with sudden/gradual load changes across different levels. This is precisely what **Hipster** delivers.

5.2 Hipster

In this section, we introduce **Hipster**, a hybrid reinforcement learning (RL) approach coupled with a feedback controller that dynamically allocates workloads to heterogeneous cores while selecting optimised DVFS settings. We propose a variant, called HipsterIn, that is optimised for latency-critical workloads running solo in the system, adjusting the system configuration to reduce energy consumption. The HipsterCo variant, which supports collocation of latency-critical and batch workloads, and focuses on maximising the throughput of the batch workloads. Both variants of Hipster always ensure that the QoS requirements are met for the latency-critical workload.

5.2.1 Hipster Reinforcement Learning

The RL problem solved by Hipster is formulated as a Markov Decision Process (MDP) [168]. In an MDP, a decision-making process must learn the best course of action to maximise its total reward over time. At each discrete instant, n, the process can observe its current *"state"*, w_n, and it must choose an *"action"* c_n from a finite set of alternatives. Depending on the chosen action and current state (but nothing else), there is an unknown probability distribution controlling which state, w_{n+1}, it enters next and the reward, λ_n, that it receives. The problem is to maximise the total discounted reward, given by $\sum_{n=0}^{\infty} \gamma^n \lambda_n$, where γ is the discounting factor. The discounting factor γ should be positive and (slightly) less than one, in order to reflect a moderate preference for rewards in the near future.

The hybrid task management problem solved by Hipster is translated to an MDP as follows. The state w_n indicates the current load on the latency-critical workload, measured during the (prior) time interval t_{n-1} to t_n.[3] Hipster quantises the load into buckets. Specifically the latency-critical application provides a measurement of the

[3]Load is Chapter 5 (only) refers to load of the latency-critical workload measured in requests per second, or queries per second, whereas in Chapter 3 it is referred to as IPS.

percentage load during the time interval, in terms of requests per second, queries per second, or similar. The action, c_n, which is chosen by Hipster depending on the state, determines the configuration to be used in the (next) time interval, t_n to t_{n+1}; that is, the combination of cores and DVFS settings allocated to the latency-critical application. These settings are used for the upcoming interval, at the end of which, at time t_{n+1}, the reward λ_n is determined depending on the level of QoS relative to the target, given a metric of optimisation: either the system power consumption (HipsterIn) or the throughput of the batch workloads (HipsterCo). A precise definition of the calculation of the reward is given in Section 5.2.4.

RL is a type of unsupervised machine learning with a focus on online learning [169]. It solves an MDP by maintaining a table of values, $R(w, c)$, indexed on the possible states $w \in W$ and possible actions $c \in C$. The entry $R(w, c)$ estimates the total discounted reward that will be received, starting from state w, if the decision-making process starts by choosing next action c. Assuming that the **lookup table**, $R(w, c)$ has close to correct values, then, if the current state is w_n, the best action c_n is the one that gives the largest total discounted reward; i.e. $c_n = \arg\max_c R(w_n, c)$. The process chooses this value of c_n, then it updates $R(w_n, c_n)$ using a particular formula based on the old and new states, w_n and w_{n+1}, and the reward λ_n.[4] A classic problem in RL is known as the *exploitation–exploration dilemma*, which captures the need not only to exploit the best solution identified so far, but also to fully explore alternatives, which may or may not be better.

Hipster uses a hybrid RL approach [170], which combines reinforcement learning with a heuristic, to be used while the algorithm is still learning the optimal behaviour. For Hipster, the heuristic improves QoS at the beginning of the execution and it is also re-used after a change in the characteristics of the problem, e.g. the mix of batch workloads. A hybrid RL [170] has the potential to outperform pure RL schemes [171, 172] that only deal with the exploitation–exploration dilemma (e.g. Q-learning), for several reasons:

(1) During the learning phase, online unsupervised learning without a heuristic generates random decisions, which would produce an unacceptable number of QoS violations.

(2) As the complexity of the problem increases, in terms of workloads, number of cores, DVFS settings, and so on, it may take longer to learn the table R. In contrast, a hybrid RL can find acceptable solutions even during the learning phase.

[4] The update of $R(w_n, c_n)$ is on line 17 of Algorithm 2.

③ The exploration feature of many RL approaches is necessary to capture a global maximum, but it may cause extra QoS violations. Using a heuristic in the learning phase can reduce the need to explore configurations that clearly violate QoS.

5.2.2 Hipster Design

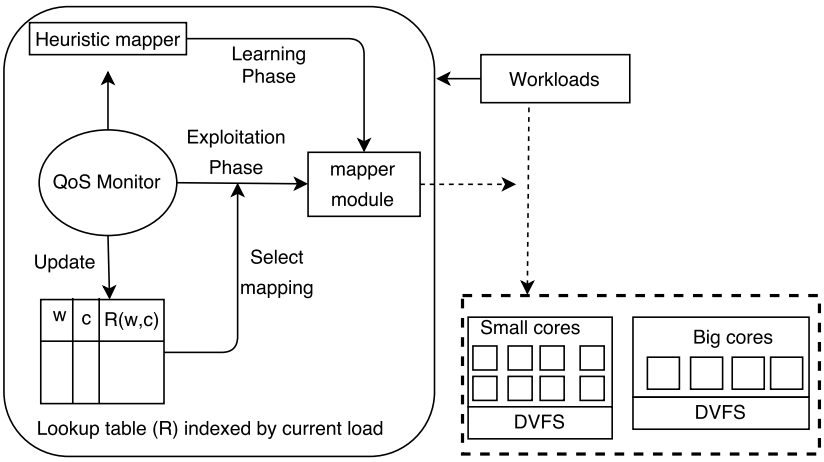


Figure 5.4 High-level view of Hipster runtime system

Figure 5.4 shows a high-level view of Hipster. Hipster includes a QoS Monitor, a heuristic mapper (used in the learning phase), an Exploitation Phase, and a Mapper Module (used in both learning and exploitation phase). Given a QoS target, an incoming load, and a metric to optimise for, Hipster learns the most adequate core configuration and DVFS settings by managing a lookup table that is used to map the workloads to the available hardware resources.

QoS Monitor. The QoS Monitor is responsible for periodically collecting the performance statistics from the latency-critical and batch workloads. For the latency-critical workload, Hipster gathers the appropriate application-level QoS metrics such as throughput (RPS or QPS) and latency (query tail latency). It also reads the current load on the latency-critical workload and quantises this value into discrete buckets between 0 and $T-1$, for (some) small value T. HipsterCo uses a profiling tool to measure the throughput of the batch workloads, using per core hardware performance counters, such as CPU utilisation, cache-misses and IPS.

Learning and exploitation phases. The data collected by the QoS monitor is used to make the thread-to-core mapping decisions. In the learning phase, Hipster

uses a feedback control loop based on heuristics to map the latency-critical workload to resources. Following the intuition from Section 5.1, when load is low, the mapper executes the latency-critical workload on small cores at lower DVFS states, and when load is high, it uses a combination of big and small cores at higher DVFS. Hipster also begins populating the lookup table so that each entry approximates the corresponding total discounted reward. Specifically, Hipster uses the reward mechanism (Section 5.2.4) to prefer core configurations that minimise system energy consumption or maximise batch workload throughput, while ensuring as well as possible that at least 95 % QoS guarantee is achieved [173].

In the exploitation phase, Hipster uses the lookup table to select the core mapping and DVFS settings, based on the load. It also continues to update the values in the lookup table, in order to continue to improve the mapping decisions. At runtime, Hipster determines when to dynamically switch between the learning and exploitation phases, based on a prefixed time quantum. At deployment stage, we ensure that the bucket size for each workload gives at least 95 % QoS guarantee [167] with minimal energy consumption.

5.2.3 Heuristic Mapper (Learning Phase)

The heuristic mapper is a state machine with a feedback control loop. The current state identifies the core configuration: the DVFS settings and number and type of cores to use for the latency-critical workload.[5] The choice of available states depends on the platform; that is, the total number and types of cores, and the DVFS settings. There is a predefined ordering of the states, approximately from highest to lowest power efficiency. This ordering is determined by measuring the power and performance of each state using a stress microbenchmark consisting of mathematical operations without memory accesses.

Whenever QoS is close to being violated, the state machine transitions into the next-higher power state. The QoS is quantified using the currently measured tail latency at the 95^{th} or 99^{th} percentile, denoted QoS_{curr}. The target tail latency is denoted by QoS_{target}. The state machine transitions to the next-higher state whenever the time interval ends in the so-called *danger zone* defined by:

$$QoS_{curr} > QoS_{target} \times QoS_D$$

[5] State machines and Markov Decision Processes use "state" with different meanings. In Section 5.2.3 (only), "state" refers to the core configuration, elsewhere it is the load.

where QoS_D is a parameter between 0 and 1 that defines the size of the danger zone. Whether such a state transition improves or degrades performance and whether it actually increases or decreases power depends on the characteristics of the platform and the particular workloads. The state machine may have to make several consecutive state transitions until the QoS is met.

In contrast, whenever the QoS is far from being violated, the state machine transitions into the next-lower power state. This happens whenever the time interval ends in the so-called *safe zone* defined by:

$$QoS_{curr} < QoS_{target} \times QoS_S$$

where QoS_S is a parameter between 0 and QoS_D that defines the size of the safe zone. The values of QoS_D and QoS_S are determined to avoid oscillations between adjacent states. In particular, QoS_D is empirically computed in the same way as for Octopus-Man [24, 173].

The heuristic proposed by Octopus-Man is attractive because of its simplicity but it can be sub-optimal (see Section 5.1, Figure 5.2c) because there is no common static ordering of configuration states that works for all workloads. Moreover, in practice, the state machine may respond slowly to rapid changes in load. Nevertheless, we found that such a state machine heuristic is suitable to accelerate the learning phase of the RL algorithm by exploring viable core configurations to quickly populate reasonable values into the lookup table.

5.2.4 Reward Calculation

During both the learning and exploitation phases, the values in the lookup table are dependent on the reward, which is calculated as defined in Algorithm 2. This reward calculation is invoked after each monitoring interval, and its definition was determined empirically (more details in Section 5.2.6). The reward λ_n has three parts: the QoS Reward, Stochastic Reward, and either the Power Reward (for HipsterIn) or the Throughput Reward (for HipsterCo):

QoS Reward. The ratio of the measured QoS to the QoS target is known as QoS_{reward}. If this value is less than one, then the QoS target has been met, and it quantifies how quick the response was as the **QoS earliness**. In this case, line 7 or 9 applies a positive reward that prefers configurations that approach the QoS target, which acts as a heuristic to reduce energy consumption or improve batch workload

Algorithm 2 Reward mechanism

▷ *Determine reward λ_n based on interval $t_n \dots t_{n+1}$*

1 Let QoS_{target} be the target QoS of the interactive workload.
2 $QoS_{\text{curr}} = QoSMonitorLatency$
3 $Power = QoSMonitorPower$
4 $QoS_{\text{reward}} = QoS_{\text{curr}}/QoS_{\text{target}}$
5 $Power_{\text{reward}} = TDP/Power$ ▷ *TDP (thermal design power)*
6 **if** $QoS_{\text{curr}} < QoS_{\text{target}} \times QoS_{\text{D}}$ **then**
7 $\quad \lambda_n = QoS_{\text{reward}} + 1$
8 **else if** $QoS_{\text{curr}} < QoS_{\text{target}}$ **then**
9 $\quad \lambda_n = QoS_{\text{reward}} + 1 - Random(0, 1)$
10 **else**
11 $\quad \lambda_n = -QoS_{\text{reward}} - 1$
12 **if** there exist batch jobs **then**
13 $\quad \lambda_n = \lambda_n + \frac{B_{\text{IPS}}+S_{\text{IPS}}}{maxIPS(B)+maxIPS(S)}$
14 **else**
15 $\quad \lambda_n = \lambda_n + Power_{\text{reward}}$
16 **if** $\nexists\, R(w_n, c_n)$ **then** $R(w_n, c_n) = 0$
17 $R(w_n, c_n) = R(w_n, c_n) + \xi\left(\lambda_n + \gamma \max_{d \in C} R(w_{n+1}, d) - R(w_n, c_n)\right)$

throughput. If QoS_{reward} is greater than one, then the QoS target has *not* been met, and it determines how intense the violation was as the **QoS tardiness**. In this case, line 10 applies a negative QoS reward.

Stochastic Reward. When the QoS is below the target, as defined in Section 5.2.3, but still over the danger zone, then a stochastic penalty is applied (line 9 of Algorithm 2). The stochastic penalty offers the possibility to continue to explore the configuration, but with a smaller probability. In future, other external influences for the latency-critical workload like noise, contention on shared resources, pending queue lengths, etc., may cause a QoS violation.

Throughput Reward (HipsterCo). Line 13 of Algorithm 2 calculates the Throughput Reward, which is approximately proportional to the total throughput of the batch workloads. Since HipsterCo does not require modifications to the batch workloads, it is only possible to measure their throughput in a generic way using performance counters. Specifically, the throughput is quantified in terms of IPS. The parameters B_{IPS} and S_{IPS} measure the total IPS of the big and small clusters running batch workloads, respectively. The denominator is constant given by the sum of $maxIPS(B)$

and $maxIPS(S)$, which measure the maximum IPS, at highest DVFS, for the big and small cores respectively. Platform specific details are given in Chapter 2.

Power Reward (HipsterIn). The ratio of the thermal design power (TDP) to the measured system power consumption is known as $Power_{reward}$ as shown in line 15. A smaller value of this term means that the system power consumption was lower, and it increases the reward.

Once the reward λ_n has been calculated, line 17 updates the value of $R(w_n, c_n)$ in the lookup table, and this is done in the same way during both the learning and exploitation phases. This update is controlled using two scalar parameters, both between zero and one: the learning rate, ξ, and, the discounting factor, γ.

Learning Factor, ξ. The ξ coefficient in line 17 of Algorithm 2 is the learning factor, which controls the rate at which the values in the lookup table $R(w, c)$ are updated. A large value of ξ close to one means that the algorithm learns quickly, favouring recent experience, but increasing susceptibility to noise. In contrast, a small value of ξ means that the algorithm learns slowly. In our experiments we used $\xi = 0.6$.

Discounting Factor, γ. The γ coefficient in line 17 of Algorithm 2 is the discounting factor, which quantifies the preference for short-term rewards [174]. Setting $\gamma = 0$ means that the algorithm only relies on immediate short-term rewards. To allow a balance between short-term and future rewards, we set $\gamma = 0.9$ (empirically determined). In other words, this methodology allows the optimization problem to also take into account future rewards.

5.2.5 Exploitation Phase

The exploitation phase of Hipster is defined by Algorithm 3. Line 6 determines the configuration, c_n, with the highest estimated total discounted reward. Lines 7 to 12 apply the configuration by mapping the workloads to the resources, as described below, depending on the specific variant of Hipster (HipsterIn or HipsterCo). Line 13 runs the workload for the next time interval, and line 15 calls Algorithm 2 to update the lookup table, based on the metrics obtained by the QoS Monitor during the time interval. Line 17 re-enters the learning phase when necessary. The mapping of workloads to resources is as follows:

Algorithm 3 Exploitation Phase

```
1  Let X be threshold on QoS guarantee to re-enter learning phase
2  Let w_n be observed load for interval t_{n-1} ... t_n
3  Let c_n be configuration for interval t_n ... t_{n+1}
4  Let n = 0
5  repeat
                                  ▷ At time t_n, choose configuration for t_n to t_{n+1}
6      Let c_n = max_{d∈C} R(w_n, d)
7      if there exist batch jobs then
8          Allocate remaining cores to batch jobs
9          if latency-critical jobs on a single core type then
10             Set highest DVFS for other core type
11     else
12         Set lowest DVFS for remaining cores
13     Sleep until t_{n+1}                  ▷ Run for interval t_n to t_{n+1}
14     Let w_{n+1} be the quantised load from the latency-critical workload
15     Call Algorithm 2                     ▷ Algorithm 2 updates R(w_n, c_n)
16     n = n + 1
17     if QoSGuarantee ≤ X then Learning phase
18 until Terminated
```

Reward Mechanism for HipsterIn. To minimise power consumption while meeting the QoS target for latency-critical workloads, the configuration with the highest reward is selected and then DVFS setting for the remaining cores is set to the lowest value (Lines 11 to 12 of Algorithm 3).

Reward Mechanism for HipsterCo. Corroborating the findings of prior work [40], we observed that collocating both latency-critical and batch workloads degrades QoS at higher loads due to shared resource contention. If the reward mechanism were not aware of such collocations, it may make decisions that violate QoS for the latency-critical workload and/or reduce the throughput of the batch workloads. As a precursor to this condition, we introduce the following mechanisms. First, to maximise the throughput of the throughput-oriented workloads while meeting QoS targets, all of the unused cores are allocated to the batch workloads (lines 7 to 8 in Algorithm 3). Second, in case the latency-critical job is allocated exclusively to a given core type, the other core type is set to the highest DVFS to accelerate the batch workloads (lines 9 to 12 in Algorithm 3). For instance, on a two-socket/cluster system with two cores per socket/cluster, if the latency-critical workload is running on two small cores, the big cores are allocated to the batch workloads at the highest DVFS.

5.2.6 Responsiveness and Stability

To ensure that QoS is met for latency-critical workloads, the scheduling policy must quickly respond to fluctuations in load and latency, either due to changes in core mapping, DVFS or any external influence. Therefore, the responsiveness and stability of Hipster is determined by (1) the computation latency in migrating cores and setting DVFS. (2) the reaction time of QoS between migrating an application from current mapping to future mapping, and (3) the granularity of monitoring for the latency-critical workload's QoS.

The computational latency for changes in core mapping and DVFS are negligible [68, 175, 176]. The default monitoring interval for Memcached and Web-Search is one second. Based on the aforementioned overheads, we determine the sampling interval as a sum of the monitoring interval for the latency-critical application, and the overhead to switch the core mapping and DVFS.

5.2.7 Hipster Implementation

Hipster is implemented in user space, and it uses minimal hardware support exposed by Linux. It consists of the QoS Monitor and Mapper Module, together with a lookup table, as shown in Figure 5.4.

QoS Monitor. Hipster uses a separate process to read the power measurements using native energy meters, at the sampling interval of the application. In addition to measuring energy, the QoS Monitor also gathers runtime statistics for the query/request latency of the latency-critical workload, using a logfile interface. In the case of HipsterCo, the batch workload aggregate IPS per core are measured using the performance monitoring tool, perf [85], specifically using the *perf_event* instructions [90, 91]. Alternatives to perf include the profiling tools [86, 87] supported by Docker, Kubernetes and LXC [89].

As discussed in Section 2.3, the perf bug on ARM processors generates garbage values for all cores whenever any core enters an idle state. Since performance statistics are required only necessary for the HipsterCo variant (where batch workloads are collocated with latency-critical workloads), we overcome this by disabling the CPUidle.

Mapper Module. The workloads are mapped to cores using the Linux sched_setaffinity call and DVFS is controlled using acpi-cpufreq. In addition, Hipster suspends and resumes the batch workloads using the relevant OS signals (SIGSTOP and SIGCONT in Linux).

Runtime overhead. Hipster has a simple algorithm, requiring few control flow statements and main memory accesses, so its runtime overhead is negligible. We measured the execution time overhead (implemented in Python and including I/O) to be < 2 ms, so triggering Hipster every second, as in our experiments, incurs an overhead < 0.2 %.

Lookup table. Each iteration of the RL algorithm accesses and modifies several entries in the lookup table. To ensure that these operations take negligible time, the computational complexity to access the table should be at most a few instructions. Therefore, in the prototype implementation of Hipster, the lookup table was implemented using a Python dictionary, which uses open addressing to resolve hash collisions, thereby having a computational complexity of $O(1)$ irrespective of the operation [177].

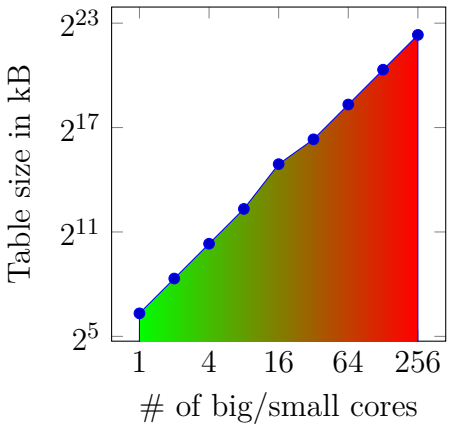


Figure 5.5 Lookup table size (log scale) with same number of small and big cores, using 10 different DVFS settings, and bucket size of 100.

We observe that the lookup table size can grow quickly (see Figure 5.5) with the number of big/small cores, DVFS settings, and the number of load buckets. Let us say there are B big cores and S small cores available, and B_f and S_f available DVFS settings for big and small cores, respectively, then the total number of actions in the table would be $C(B+B_f, B_f) \times C(S+S_f, S_f)$, where C(x, y) refers to the number of different ways to choose actions (y) for out of x distinct elements. This grows in the order of $O(B^{B_f} \times S^{S_f})$.

Figure 5.5 shows the unoptimised (or naïve) lookup table size in kB when the number of big cores is equal to number of small cores ($B = S$) with 10 DVFS states for big and small cores ($B_f = S_f = 10$) with 100 load buckets (w). For instance, 64 cores of each type leads to 40,960,000 combinations (in contrast to 56 configurations on ARM Juno), which gives a table size of approximately 327 MB for a whole node.

To reduce the size of the lookup table in Hipster, we store the configuration and reward only for the configurations which have been explored so far. The lookup table is still implemented using a Python dictionary as the complexity is $O(1)$ irrespective of the operation. Furthermore, in the exploitation phase, we ensure that each load bucket (w) contains *only* the top K (in our case, $K = 5$) configurations with highest discounted maximum rewards.

5.3 Evaluation

5.3.1 Algorithm Configuration

In deploying Octopus-Man, we first performed a sweep on the danger and safe thresholds, and picked the combination of thresholds with the highest QoS guarantee. For HipsterIn, we set the learning phase to be 500 seconds, except when quantifying the learning time, where we set it to 200 seconds. In addition, we quantify HipsterIn with changes in application at runtime, where we run each application for 1500 seconds.

5.3.2 HipsterIn Results

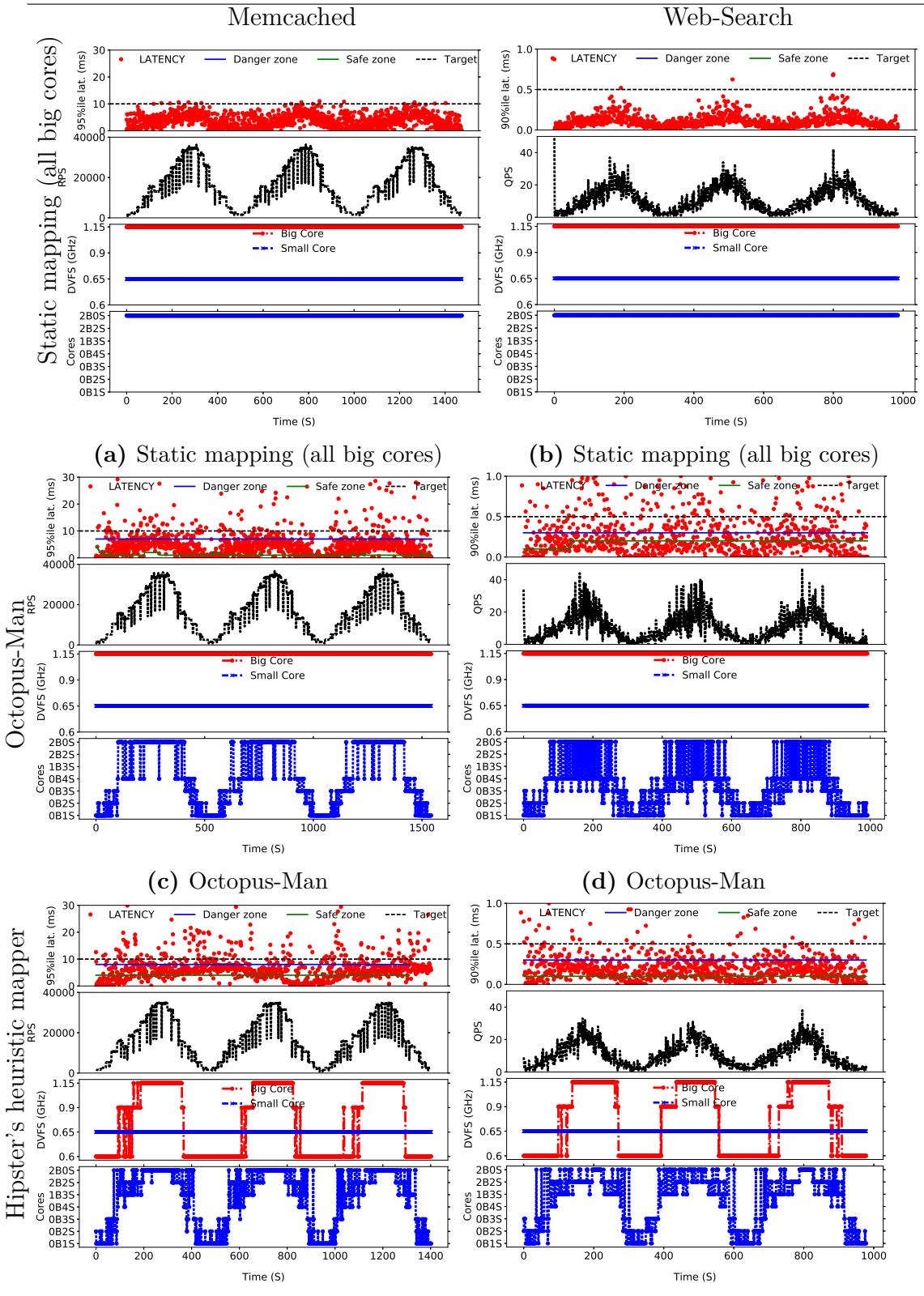
This section evaluates the effectiveness of HipsterIn, as a policy for managing a single interactive workload. The objective is to minimise system energy consumption while satisfying QoS.

5.3.2.1 Hipster's Heuristic Policy (interactive only)

We first evaluate the effectiveness of Hipster's heuristic policy alone, for mapping interactive workloads. Figure 5.6 shows the results for both workloads: Memcached (left-hand column , subfigures (a), (c) and (e)) and Web-search (right-hand column, subfigures (b), (d) and (f)). The rows, from top to bottom, correspond to static mapping, for which the interactive threads are mapped to the two big cores at highest DVFS of 1.15 GHz ((a) and (b)), Octopus-Man ((c) and (d)) and Hipster's heuristic policy ((e) and (f)). For each subfigure, from top to bottom, the first plot presents the tail latency (QoS), with the target marked with a dashed line. The second plot shows the achieved throughput in RPS (requests per second). The third plot presents the DVFS of the big and small cores, and the fourth plot represents the choice of core mapping.

Comparing the DVFS and core configuration subplots, we observe that Hipster's heuristic policy is successfully exploring the DVFS settings available on the Juno platform (bottom plots), and it is exploring all configurations including those that use both big and small cores at the same time (bottom plots). In contrast, Octopus-Man does not adjust the DVFS settings and it uses either the big or small cores, but not both at once.

Both Octopus-Man and Hipster's heuristic policy frequently oscillate between consecutive core configurations. In the case of Octopus-Man, there are clear oscillations between two big cores and four small cores, for example between the 600^{th} and 800^{th} seconds. Using two big cores satisfies QoS but since it is within the safe zone, Octopus-



(a) Static mapping (all big cores)

(b) Static mapping (all big cores)

(c) Octopus-Man

(d) Octopus-Man

(e) Hipster's heuristic mapper

(f) Hipster's heuristic mapper

Figure 5.6 Comparison of heuristic policies. Static mapping (top), Octopus-Man (middle), and Hipster's heuristic policy (bottom). Results are shown for Memcached (left-hand column) and Web-Search (right-hand column) on ARM Juno R1.

Man switches to four small cores, which enters the danger zone, generating an alert provoking a return to two big cores. Such oscillations between cores in different clusters leads to severe QoS degradation of up to 20 %. As expected, the static mapping (all big cores) has the least number of violations.

In summary, although Hipster's heuristic policy alone improves over Octopus-Man by exploiting a wider search space, it still suffers from an unacceptable number of QoS violations.

5.3.2.2 HipsterIn: Memcached Results

Figure 5.7 shows the results using HipsterIn for Memcached. After completing the learning phase, the oscillatory effect between core mappings is greatly reduced (by 8.3 %), and overall the QoS guarantee is improved by 24 % compared with the learning phase. HipsterIn performs well because it moves directly to the appropriate core configuration for a given load that satisfies QoS.

In addition to switching between a combination of different cores, HipsterIn also explores more fine-grained DVFS adaptations, which has lower overheads (of microseconds) compared with migrations between cores (order of milliseconds) [60].

5.3.2.3 HipsterIn: Web-Search Results

Figure 5.8 shows the results using HipsterIn for Web-Search. In contrast to the heuristic policies (Octopus-Man and Hipster's heuristic), during the exploitation phase, HipsterIn monitors the QoS and dynamically adjusts the core mapping and DVFS

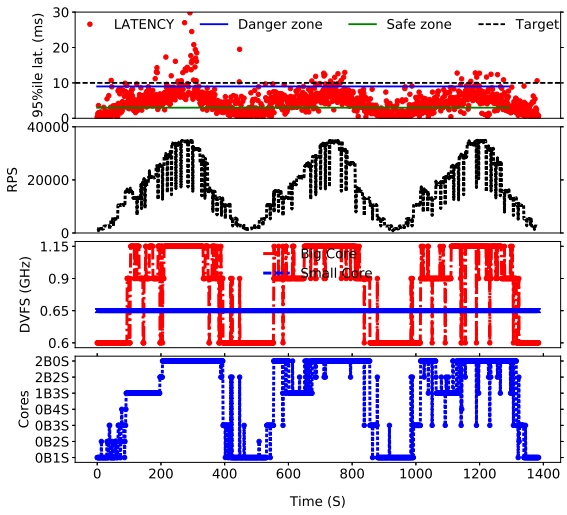


Figure 5.7 HipsterIn on Memcached

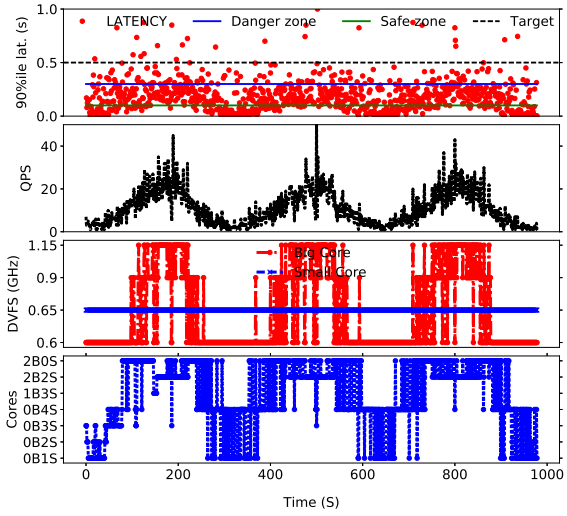


Figure 5.8 HipsterIn on Web-Search

settings to adapt to load fluctuations. Both Hipster's heuristic and Octopus-Man perform aggressive changes to core mappings to reduce energy, leading to a negative impact on QoS. On the other hand, HipsterIn shows a more balanced behaviour, performing 4.7× fewer task migrations than Octopus-Man for Web-Search, while improving QoS up to 16 % and reducing energy consumption by 13.5 %.

5.3.2.4 HipsterIn: Swapping applications

We also evaluate the effectiveness of HipsterIn to adapt to changes in application at runtime and deliver real-time performance guarantees even for the new incoming application.

Web-Search to Memcached Figure 5.10 shows the results using HipsterIn when swapping from Web-Search (represented in red) to Memcached (represented in cyan) after 1500 s of execution. "NA" in the last two subplots on y-axis represents the period when Web-Search or Memcached are running exclusively. HipsterIn shows a more balanced behaviour, performing 22% fewer violations than learning phase, set at 500-seconds, when running Web-Search. Thereafter, when swapping Web-Search with Memcached observe that HipsterIn has a far more QoS violations but quickly adapts to changes in application by learning the lookup table suitable for Memcached.

Memcached to Web-Search Figure 5.9 shows the results using HipsterIn when swapping from Memcached (represented in cyan) to Web-Search (represented in red) after 1500 s of execution. "NA" in the last two subplots on the y-axis represents the period when Memcached or Web-Search are running exclusively. When application is swapped at runtime from Memcached to Web-Search, HipsterIn dynamically updates the lookup table to adapt to changes in application behaviour. For instance, observe at 1500 s, HipsterIn has large number of violations (by 7.1 %) but quickly drops after it adapts the lookup table and the overall QoS guarantee is improved by 38 %. The QoS guarantee of HipsterIn for Web-Search is 95%, which is similar to QoS guarantee when running Web-Search from the start (Table 5.1)

5.3.2.5 HipsterIn Summary

Table 5.1 summarises the QoS guarantee, QoS tardiness and energy reduction for Memcached and Web-Search for different policies: Static (all big cores), Static (all small cores), Hipster's heuristic mapper, Octopus-Man and HipsterIn. We compare the energy consumption of each mapping schema against Static (all big cores). We quantify

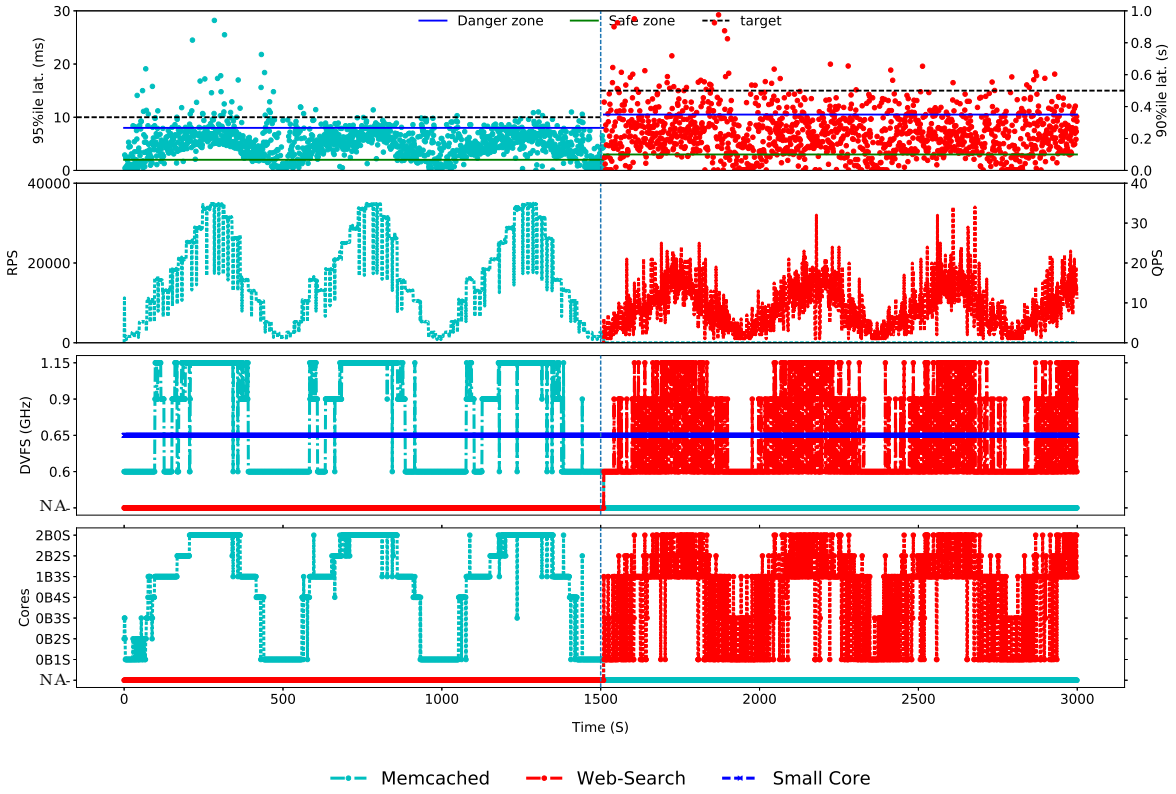


Figure 5.9 HipsterIn on Memcached swapping to Web-Search

the QoS behaviour at each sampling interval by assessing the measured QoS using two metrics: QoS guarantee and QoS tardiness.[6] The QoS Guarantee is the percentage of samples for which the measured QoS did not violate the target (100 %-QoS violations%). The QoS tardiness in the table is the average (mean) of the QoS tardiness, including only the samples that violated the QoS target.

As shown in Table 5.1, for Web-Search and Memcached, static (all small cores) cannot meet the required QoS. On the one hand, the heuristic policies reduce energy marginally, but violate QoS due to excessive core migrations. On the other hand, HipsterIn meets QoS at 99.4 % and 96.4 % for Memcached and Web-Search, while having energy savings of 14.3 % and 17.8 %, respectively.

5.3.2.6 HipsterIn Analysis

Rapid adaptation to load changes. Hipster can respond to rapid changes in load by directly mapping to a configuration that satisfies the QoS. Figure 5.11 shows how HipsterIn (during the exploitation phase) and Octopus-Man respond to changes in load.

[6] QoS Tardiness is QoS_{curr}/QoS_{target}, using the definitions from Section 5.2.4.

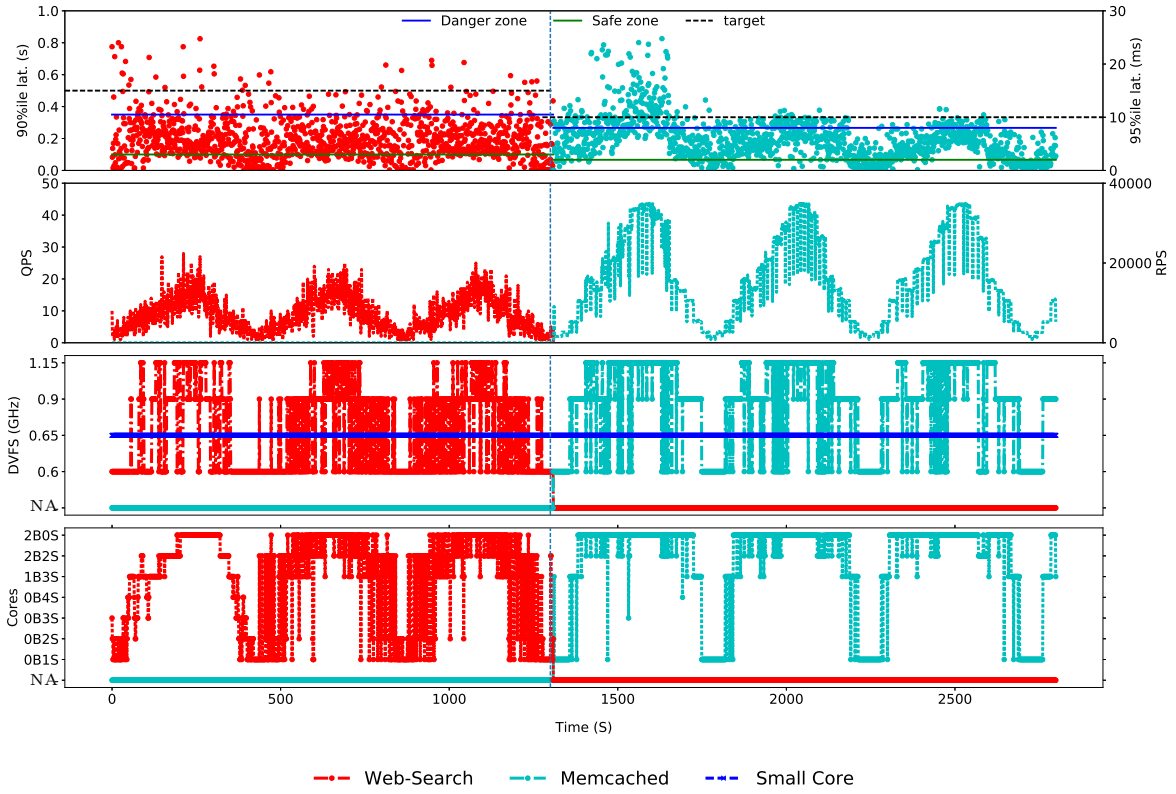


Figure 5.10 HipsterIn on Web-Search swapping to Memcached

From top to bottom, we express the input load in terms of the percentage of maximum load, where it increases from 50 % to 100 % over a period of 175 seconds for Memcached. In the second graph, we express the 95^{th} percentile tail latency as QoS Tardiness. A QoS violation has occurred if the QoS Tardiness is above 1, otherwise QoS is satisfied. We find that Octopus-Man violates QoS due to aggressive core mappings to minimise energy consumption. By contrast, HipsterIn achieves more stable tail latency even at higher load (80 %). Note that, from 75 % to 90 % of the load, the QoS tardiness (extent of violation) experienced by HipsterIn is 3.7× (mean) lower than Octopus-Man.

Impact of learning time. HipsterIn aims to deliver the best balance between QoS guarantee and energy reduction compared to heuristic policies (Table 5.1). In practice, to best optimise for energy efficiency, and to improve QoS, HipsterIn needs a short learning phase. Figure 5.12 shows the QoS guarantee and energy distribution smoothed over 100 s intervals using the Savitzky–Golay filter [178] for Web-Search, for both HipsterIn and Octopus-Man. Each data point in the graph refers to a 100-second interval. The learning phase is set to 200 s. As can be seen, HipsterIn quickly learns during the heuristic phase, which improves QoS guarantees. On the other hand, for

Table 5.1 Summary of HipsterIn for Memcached (Mem) and Web-Search (Web). A summary of QoS guarantees, tardiness and energy savings

	QoS Guarantee		*QoS Tardiness*		*Energy Reduction*	
	Mem	*Web*	*Mem*	*Web*	*Mem*	*Web*
Static (all big cores)	99.5%	99.5%	1.1	1.3	-	-
Static (all small cores)	85.8%	78.4%	1.4	2.0	48.0%	31.0%
Hipster's Heuristic	89.9%	95.3%	1.8	1.9	18.7%	13.6%
OctopusMan	92.0%	80.0%	2.2	2.1	17.2%	4.3%
HipsterIn	**99.4%**	**96.5%**	1.4	2.0	**14.3%**	**17.8%**

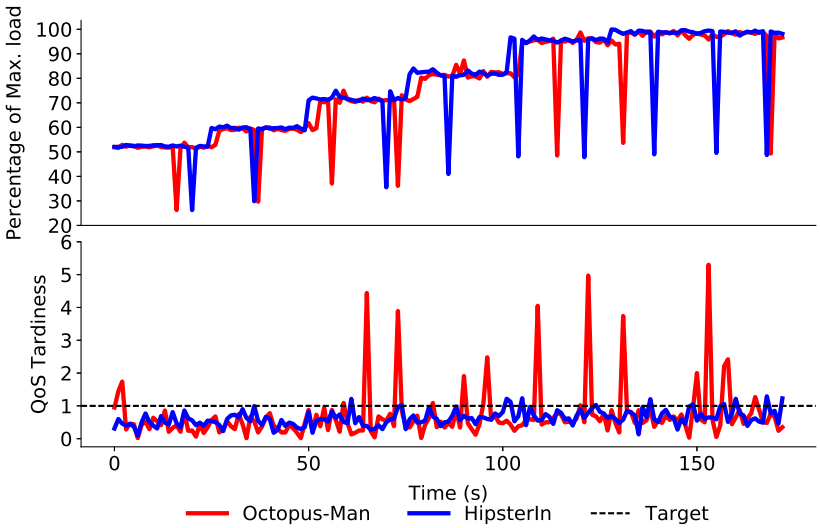


Figure 5.11 Responsiveness to load change of HipsterIn and Octopus-Man. Percentage of Max. load, and Tail latency (QoS Tardiness) running Memcached with HipsterIn and Octopus-Man.

Octopus-Man, the QoS guarantees are consistently around the 80 % mark, since it does not use past decisions and their associated effects to improve the future decisions.

Impact of bucket sizes. Figure 5.13 shows the impact on QoS and energy savings when varying the load bucket sizes in Hipster. The x-axis represents the bucket size, expressed as the percentage of maximum load. Each bar in the figure (y-axis) represents the QoS violations and energy reductions normalised to Static (all big cores). Using a large bucket size forces Hipster to use the same core configuration across a wide range of loads, whereas using a small bucket size allows fine-grained control. A small bucket size therefore improves the energy savings, but it tends to cause rapid changes in core configuration for small changes in load, and doing so incurs a larger number of

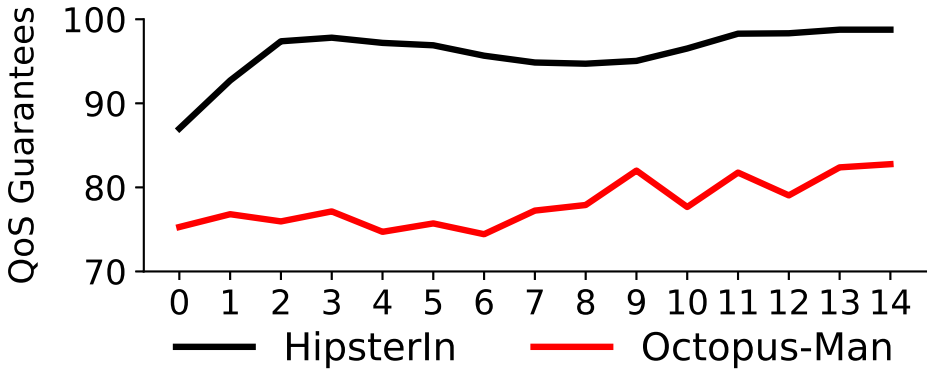


Figure 5.12 QoS Guarantees of HipsterIn and Octopus-Man. Each data point represents the QoS guarantees over 100 s intervals.

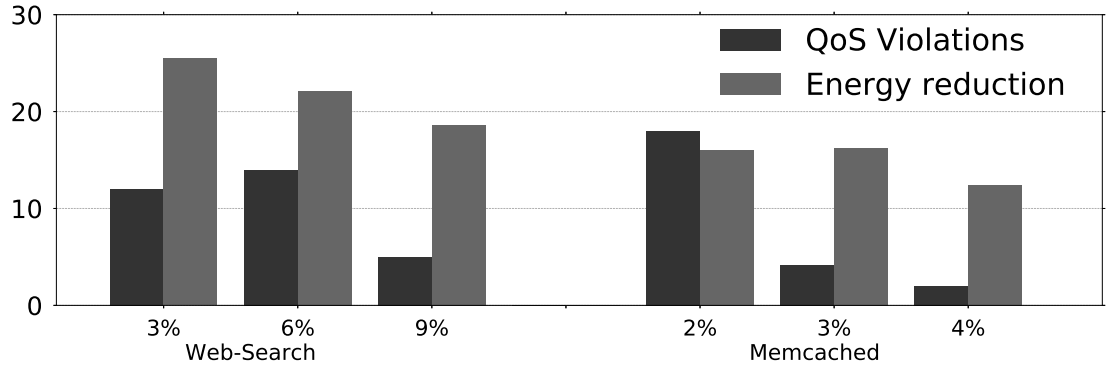


Figure 5.13 Impact of bucket size. QoS guarantees and energy savings normalised to static (all big cores) on Web-Search and Memcached.

QoS violations. On the other hand, larger bucket sizes provide better QoS guarantee but lower energy savings, because they categorise large variations in load into a single load bucket. Therefore, in tuning Hipster, we empirically determine the bucket size to maximise energy savings subject to at least 98 % QoS guarantee.

Optimising the lookup table size Table 5.2 summarises the QoS guarantee, number of configurations per load bucket, and number of configurations for Memcached and Web-Search with unoptimised and optimised lookup tables. We quantify the reduction in lookup table size by assessing the number of configurations stored for each load bucket at the end of the execution. As shown in Table 5.2, the optimised lookup table saves 90 % of the memory footprint while providing similar QoS guarantee to unoptimised lookup table because, it only stores those configurations that have been explored in the learning phase and in the exploitation phase, the size of the table is further reduced by keeping only the top five configurations with the highest reward.

Table 5.2 Results of the table size optimisation in Hipster

	Lookup Table	Memcached	Web-Search
QoS Guarantee (%)	Unoptimised	99.4	96.5
	Optimised	99.2	96.0
# of config./load bucket	Unoptimised	13	13
	Optimised	5	5
Table size (elements)	Unoptimised	1344 (0% savings)	392 (0% savings)
	Optimised	120 (91.1% savings)	35 (91.1% savings)

This is possible, as the configurations explored at a given load level for a particular application are chosen from only a subset of the configurations in the learning phase instead of random solutions as in Q-Learning.

5.3.3 HipsterCo Results

This section evaluates the effectiveness of HipsterCo, as a policy for collocating a single latency-critical workload and a mix of batch workloads. The objective is to maximise the throughput of the batch workloads while satisfying QoS of the interactive workloads.

Figure 5.14 shows the QoS guarantee (top), throughput (middle) and energy consumption (bottom) for Web-Search collocated with batch workloads, managed by Octopus-Man and HipsterCo. All figures are normalised to a static mapping that allocates the latency-critical workload to the two big cores and the batch workloads to the four small cores. The number of running batch workloads is equal to the number of cores not utilised by Web-Search. We report the system throughput by aggregating the IPS of all batch programs.

As shown in the top plot of Figure 5.14, HipsterCo consistently delivers 94% QoS guarantees, whereas Octopus-Man has much lower QoS guarantees of 76 %. This is because Hipster learns from the QoS behaviour and performance history and is able to jump directly to a core mapping and DVFS state that satisfies QoS. As a result, it incurs fewer core migrations compared with Octopus-Man (see Section 5.3.2.3), so it achieves superior QoS guarantees.

As shown in the middle plot of Figure 5.14, for all benchmarks, Hipster and Octopus-Man deliver much higher throughput compared to static mapping, with an average of 2.3× and 2.6× improvement, respectively. Both task managers improve performance compared with the static mapping because they migrate the latency-critical workload to

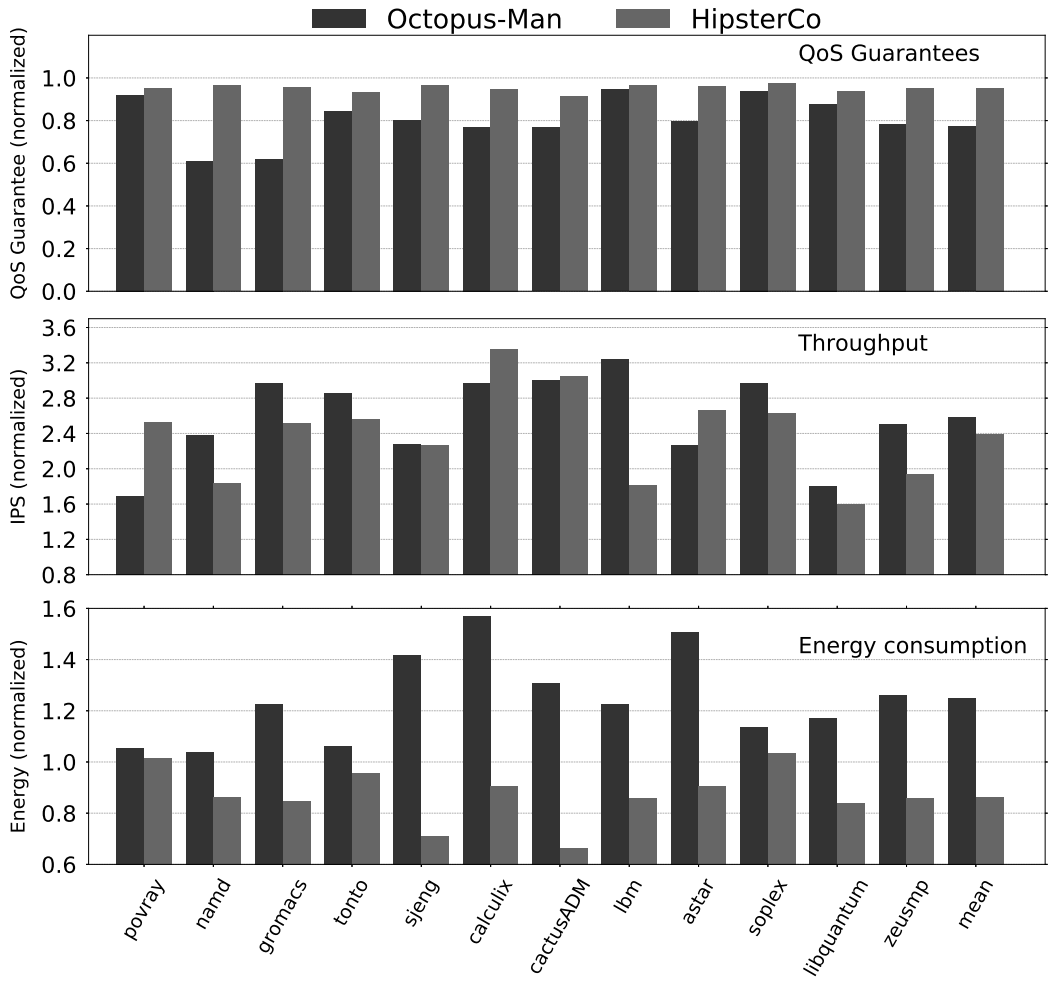


Figure 5.14 Collocation of interactive and batch workloads with HipsterCo. QoS guarantee (top), Throughput improvement (middle) and Energy consumption (bottom) when Web-Search is collocated with batch workloads. The results are normalised to static all big cores.

small cores during periods of low load, allowing the batch workloads to run on big cores (which can be 2.6× more powerful than small cores). For *Calculix*, a compute-bound application, HipsterCo achieves the highest throughput improvement over static of 3.35×, and for *libquantum*, a memory-bound program, the least improvement is still 1.6×.

As shown in the bottom plot of Figure 5.14, HipsterCo reduces the energy consumption to an average of 80 % of static, whereas Octopus-Man increases energy to an average of 1.2 times that of static. This is because, as shown in Figure 5.2, Hipster explores a wider range of core configurations, including DVFS settings and mixing core types. In contrast, Octopus-Man only allows the latency-critical workload to occupy a single cluster and each cluster is set to the highest DVFS.

HipsterCo sometimes chooses a different performance–energy trade-off than Octopus-Man. An example is *lbm*, a memory bound workload, for which HipsterCo delivers 40 % the throughput of Octopus-Man, but 31 % lower energy. There are two main reasons for this. Firstly, when HipsterCo uses DVFS for the latency-critical workloads, this DVFS setting also applies to batch workloads running in the same cluster, reducing both batch throughput and system energy. Secondly, HipsterCo sometimes uses a larger number of cores at lower DVFS, leaving fewer resources available for the batch workloads. As a result, on average, HipsterCo marginally reduces performance (by 7 %) but it delivers energy savings of 33 %, both compared with Octopus-Man.

5.4 Deployment Methodology for New Workloads

Hipster is a user-level daemon written in Python that is designed to implement a hybrid reinforcement learning approach under Linux running on multicore machines. It is released under General Public License (GPL) v3 and its source is available for download.[7] To deploy Hipster at runtime, we initiate a one-time system profiling to have minimal hardware understanding (similar to Octopus-Man [24]), and then the Hipster runtime is responsible to manage a given workload on a physical machine.

5.4.1 System profiling

We profile the system by running the stress microbenchmark (with no memory access) at different core mappings and DVFS settings to gather the performance (such as IPS) and power statistics.[8] The performance and power statistics can be obtained using performance monitoring tools (see section 5.2.7) and power meters (see section 2.4), respectively. Then, sort the configurations in the increasing order of the system's mean power efficiency (performance per watt of power consumed) and only the order is of interest to us. The system's TDP is obtained by running the microbenchmark on all cores at the highest DVFS setting.

For each (new) latency-critical workload, we quantify the maximum load (measured in terms of queries/requests per second or similar) the system can manage on the most power efficient configuration while satisfying the QoS target (measured in ms or s).

[7] https://bitbucket.org/rnishtala/hipster.git

[8] The stress microbenchmark is executed on each of the available cores simultaneously and is given by "stress_cpu.c" in the source code.

5.4.2 Hipster runtime

Hipster is invoked periodically, with the period determined by the sampling interval of the latency-critical workload. Hipster takes as input the *system power consumption*, *QoS target*, *load*, *maximum load* and *latency* (measured in ms or s) of the latency-critical workload and finally, the *performance* (measured in IPS) of the throughput-oriented workload (if any). The performance metrics such as QoS target, load and latency can be obtained either from the application directly (as in our case) or an external profiling tool such as Google Wide Profiler (GWP) [86].

To make the best use of Hipster, we suggest tuning the upper threshold (QoS_D) and lower threshold (QoS_S), learning factor (ξ), discounting factor (γ) empirically. Nevertheless, in the current implementation, we predefine the upper threshold and lower threshold to 80% and 20% of the QoS target, respectively; and the learning factor and discounting factor are set to 0.6 and 0.9, respectively.

Our implementation assumes that there exists an ACPI governor [77] that allows external changes to DVFS state. The governor selected on our ARM processor is userspace. The deployment stage of Hipster needs permission to access Ring 0 or root as the ACPI module writes changes in DVFS state to the sysfs. We provide an example of Hipster's deployment method.

Hipster schedules the workloads using Hipster's heuristic algorithm and reinforcement learning mechanism. For a multicore server, we use Hipster to dynamically select the core mapping and DVFS state to "just meet" the QoS based on the load for an interactive workload and any remaining resources are allocated to throughput-oriented workloads (if any).

5.5 Conclusion

We propose Hipster, a hybrid scheme that combines heuristics and reinforcement learning to manage heterogeneous cores with DVFS control for improved energy efficiency. We show that Hipster performs well across workloads and interactively adapts the system by learning from the QoS/power/performance history to best map workloads to the heterogeneous cores and adjust their DVFS settings. When only latency-critical workloads are running in the system, Hipster reduces energy consumption by 13 % in comparison to prior work. In addition, to improve resource efficiency in shared data centres by running both latency-critical and batch workloads on the same

system, Hipster improves batch workload throughput by 2.3× compared to a static and conservative policy, while meeting the QoS targets for the latency-critical workloads.

Part IV:

Epilogue

The questions are always more important than the answers.

Randy Pausch

CHAPTER 6

Related Work

In this chapter, we discuss the work related to this dissertation. We begin the chapter by discussing the topic of generating performance and power models for multicore systems (section 6.1). Prior work primarily focused on the problem of building statistical models to detect applications compute and memory phases to determine the power consumption and the scalability of the application. In this context, statistical modelling refers to generating time series, linear regressions, multivariate models, etc., using PMCs available [33, 39, 127, 128, 145, 179–182] on most commercially processors. It has been documented in previous studies that a thread's power consumption, performance and energy consumption can vastly depend on the DVFS of the core/socket and contention for shared resource (amongst others). The use of such statistics to determine the causal effect of performance and power is crucial in data centres to deploy a power cap or to ensure performance guarantees are met. For instance, the Intel RAPL [183] module was designed based on such models to provide the end-users with the capability to control systems power consumption.

Section 6.2 discusses the topic of contention-aware scheduling aimed at maximising energy efficiency in multicore systems. Prior work [30–32, 55, 184–187] has focused predominantly on the problem of cache contention since this was assumed to the primary, if not the only source of performance degradation. In this context, the aforementioned research works propose using either statistical modelling or heuristics to deal with inter-core or intra-core contention, and to determine thread's behaviour (compute bound or memory bound) at runtime, thereby minimising contention for shared resources.

In our work, we proposed a unified framework to generate models that estimate the performance and power consumption of workloads at multiple hardware settings. These models are deployed at runtime to ensure power constraints or performance

guarantees are met. After extensive experimentation, we suggest that the proposed models can estimate performance and power with minimal hardware support for any batch application and can be deployed on an architecture as they use basic PMCs. None of the works cited in Sections 6.1 and 6.2 investigated an architecture and application-agnostic approach.

Nevertheless, as the nature of the workloads executing on data centres moving towards user-centric jobs (i.e., jobs requiring fast response times, typically in the order of milliseconds measured as tail latency) rather than throughput-oriented (i.e., where response time of the application does not impact user experience), the plethora of work addressing scheduling of batch workloads are ineffective as they tend to violate tail latency requirements [188]. This is because, the metrics to schedule batch workloads are often focused on instructions per cycle, whereas the latency-critical workloads are dependent on the interactiveness. In this context, we introduced an energy efficient (and resource efficient) scheduler for multicore systems to meet tail latency requirements of interactive workloads. The proposed approach used a hybrid reinforcement learning approach making it application and architecture agnostic. Precisely, our work has shown that we can allocate "just enough" resources for a latency-critical job while meeting their performance targets and the remaining resources for the batch jobs to maximise throughput. Section 6.3 details solutions in the area of cluster-level data centre scheduling of interactive workloads.

6.1 Performance and Power Modelling

Performance and Power Modelling. Bellosa [127] use PMCs at runtime to build an OS power-aware policy. Lewis *et al.* [128] use a traditional regression-based methodology and multivariate adaptive regression splines (MARS) model to present a chaotic time series power activity. Isci *et al.* [180] study compares power phase classifications based on PMCs and proves that PMCs detect more power phases. Singh *et al.* [123] propose a power model using PMCs for an AMD processor. Isci *et al.* [129] show the program behaviour in terms of power phases. Floyd *et al.* [189] describe a power and temperature framework for IBM POWER7 processor using PMCs. Rountree *et al.* [73] estimate performance (quantified as IPC) across DVFS states by predicting the total number of leading load cycles. Miftakhutdinov *et al.* [135] predict performance on simulated architectures based on prefetch and variable memory access latencies. Spiliopoulos *et al.* [190] estimate the total time for each LLC miss, and sums these times to estimate the total time spent in memory, to predict performance at DVFS states. Torres *et al.* [191]

propose a memory compression and request discrimination technique to consolidate workload on minimal number of servers to reduce wastage of energy.

Dhiman and Rosing [181] build on two of their previous works for DVFS-based power management algorithm [192] and purely DPM [193]. Such methods are now accessible through Ring-1 of the Linux kernel as C-States.

Power capping. Reda *et al.* [194] introduce a technique to improve performance by dynamically changing power caps and thread allocations using DVFS. Rountree *et al.* [183] explore the idea of power limits in HPC environments using RAPL instead of DVFS. Patki *et al.* [182] propose an idea for over-provisioning compute nodes in power-constrained HPC data centres.

Das *et al.* [195] show a technique to enforce power limits via forced idleness. Brooks *et al.* [142] shows a power analysis at a per cycle level for an architecture. Deng *et al.* [196] propose a simulated system-level framework with performance and power predictions. Sasaki et al. [197] propose C-3PO, a power manager to maximise energy efficiency under power constraints. The hardware actuators used to allocate resources are the DVFS states and number of cores.

Cochran *et al.* [70] predict performance with an offline analysis trained using multinomial logistic regression classifier. When a change in configuration is required at runtime, this classifier returns the best candidate operating configuration. Petrucci *et al.* [33], predict performance in heterogeneous DVFS states on a homogeneous architecture. The work proposes to build one linear model for every combination of DVFS states using only LLC misses and IPS. However, prior works [39, 133, 190] have shown that considering only the LLC misses and MIPS is not a good metric to predict performance at a very high accuracy. Srinivasan *et al.* [145] predict the performance of threads running on heterogeneous cores, that is from one core type to another, using closed expressions. These expressions, however, do not suffice for a generic approach. Su *et al.* [39] proposes a system-level performance [198] and power model by taking advantage of the PMCs available in commodity AMD processors to estimate the total number of leading loads, and in turn, predicts performance and power across DVFS states.

By contrast to all prior works, Mccullough *et al.* [138] state that linear regression based power models tend to work only in restrictive scenarios and over-fit based on application types. Our results in prior chapters show that linear regression models built using a small training dataset do estimate power and performance for a broad range of workloads with relatively high accuracy and small computational cost. Similar observations were carried out in [123, 128, 129, 133].

In contrast to the aforementioned works, our modelling approach improves in at least three ways:

(1) The models are built using basic PMCs available across all architectures (Intel, AMD and ARM), making it a more generic approach with low complexity.

(2) Since our modelling approach is bottom-up, that is, based on single-core models to predict in multicore environments, it can facilitate for per core performance and power management. This is especially useful in multi-node, multicore data centre consisting of numerous applications with different performance and power constraints.

(3) Since our modelling approach can make predictions at different hardware settings simultaneously. The prediction approach is an excellent black-box approach for a single step fine-grained per core power or performance optimisation problem solver without external power meters or using application signatures [32].

6.2 Energy Efficient Scheduling

While SMT-based multicore systems are emerging as the norm for achieving the computing power necessary [199, 200], it is equally important to map these application to maximise energy efficiency [34, 201]. With energy efficient computing emerging as an important paradigm, recent approaches adopted to using linear programming or heuristics to map threads-to-cores based on a quantifiable relative gain metric.

Gandhi *et al.* [141] show a technique to minimise the product of response time and power costs. Articles [30, 32, 33] describe techniques to schedule workloads to improve energy efficiency based on the contention for shared resources.

Becchi *et al.* [140] show a technique for dynamic thread assignment based on IPC-driven technique on heterogeneous cores by computing speed up factor based on IPC ratios. Similarly, Kumar *et al.* [202] presented a power aware technique that dynamically schedules workloads to cores by predicting resource requirements of the program. For instance, a big core is given to program with high ILP and a small core a program with low ILP. The approach proposed by Kumar *et al.* allow optimising for different objectives such as performance and energy.

Sawalha *et al.* [203] present a thread scheduling technique to improve energy efficiency in heterogeneous multicore systems. For each instruction window, the approach estimates its working set signature based on the instructions executed in a given phase. By comparing consecutive windows, phases are identified. For the unmarked phases, the EDP on each core type is determined [33]. For a marked phase,

the EDP is assumed to the same as the prior occurrences. The lowest EDP values from the stored values is chosen for the thread-to-core mapping.

Zhuravlev *et al.* [55] and Padmanabha *et al.* [204] demonstrate that application signatures are an effective way to determine applications program context and execution history for each phase. These signatures are deployed in mappers to schedule workloads to cores can lead to higher energy efficiency. For instance, [204] use for scheduling workloads on suitable core type, whereas [55] identify memory intensive sections to distribute memory intensity uniformly across domains. Saez *et al.* [205] propose a scheduler to maximise performance of both single threaded and parallel workloads using the memory intensity of threads to guide thread assignment decisions. Koufaty *et al.* [206] change thread-to-core affinity measure based thread's memory intensity (LLC misses per committed instruction).

Goiri *et al.* [207] proposed a method to estimate the amount of solar energy that might be available and schedule jobs such that the green energy consumption can be maximised while meeting batch jobs' deadlines. Similarly, Le *et al.* [208] propose algorithms that minimises fossil fuel-based energy consumptions. Kontorinis *et al.* [209] have studied the effectiveness of using batteries in data centers. This study shows that using batteries as a primary energy source can lower the capital cost power delivery and the operating costs.

By contrast to the prior approaches, the following works [210–212] analyse microarchitectural characteristics of the heterogeneous platform to optimise the design area and power or performance efficiency to help design future multicore systems. Although these works do not provide a scheduling approach, these works are insightful to enhance capabilities of future multicore and heterogeneous multicore systems.

6.3 QoS Guarantees for Interactive Workloads

Energy efficient cluster computing is focused on meeting deadlines for latency-critical jobs [27, 112–114, 213] while utilising as little as energy as possible. This is a major problem in today's data centres [26, 60, 118, 214] and below we summarise the most relevant and related research.

Mars *et al.* [56, 58] detect at runtime the memory pressure and find the best collocation to avoid negative interference with latency-critical workloads. They also have a mechanism to detect negative interference allocations via execution modulation. However such fall-back mechanism would not adhere to applications like Memcached, as modulations have to be done at a finer granularity.

Novaković *et al.* [215] identifies and manages performance interference between VM systems collocated on the same system. Nathuji *et al.* [42] develop a feedback based mechanism to tune resource assignment to avoid negative interference to collocated VMs systems. Zhang *et al.* [216] enables race-to-finish for low-priority workloads to not have a deadlock with high priority services.

Petrucci *et al.* [24] was designed for big.LITTLE architectures to map workloads on big and small cores at highest DVFS using a feedback controller in response to changes in measured latency. Lo *et al.* [40] uses a feedback controller that exploits collocation of latency-critical and batch workloads while increasing the resource efficiency of CPU, memory and network as long as QoS target is met. However, this work is limited to modern Intel architectures due to its extensive use of cache allocation technology (CAT) and DRAM bandwidth monitor, which are available from Broadwell processors released after 2015. Lo *et al.* [103] achieves high CPU energy proportionality for low latency workloads using fine-grained DVFS techniques. Vamanan *et al.* [162] and Kasture *et al.* [60] exploit request queuing latency variation and apply any available slack from queuing delay to throughput-oriented workloads to improve energy efficiency. Delimitrou *et al.* [25] use runtime classification to predict interference and collocate workloads to minimise interference.

Carvalha *et al.* [217] introduce an approach to allocate a subset of the unused resources for long-term availability SLOs. This methodology deploys time series forecasting under the premise that availability of resource usage patterns in short jobs. Nevertheless, this approach does not handle resource allocation effectively as short jobs do not have certain resource patterns. Furthermore, the approach ignores the unused and underused resources caused by time-varying demands in data centre environments.

Tarcil [218] and Firmament [219] use information on the type of resources applications need in a sampling interval to deploy in distributed scheduler using an analytically-derived sampling framework that provides high quality resources within a few milliseconds.

Wong *et al.* [44] introduces a server architecture that couples commercial available compute nodes to adapt the changes in system load and improve energy proportionality. Wu *et al.* [220] (Autoscale) is for load-balancing a single workload, whereas Hipster could be used for multi-tenant data centres (different workloads on different nodes). Also, Autoscale cannot exploit heterogeneity properly. In contrast, at low utilisation, Hipster can use the small cores for the latency-critical workloads and leave the big cores for batch workloads.

Zhou *et al.* [221] proposed a heterogeneous platform-aware power provisioning system for data centres. The management framework distributes power from either renewable and non-renewable sources between small and big cores to achieve a higher energy efficiency while meeting SLO targets.

Violaine *et al.* [213] develop a dynamic resource allocation algorithm which considers each the architectural characteristics of the infrastructure such as performance, energy consumption, and their turn on/off latency. Using such information, the framework makes decisions of resource reallocation to ensure that there exist as little as possible QoS violations for latency-critical workloads while having potential energy gains.

Tesauro *et al.* [170] use an *offline model* based on heuristics for autonomous resource allocation, which may be limited to specific architectures or applications. Building a lookup table at runtime is important because applications have diverse power and performance characteristics which need to be learnt individually (as shown in Section 5.1). Prior work has previously introduced optimisations for lookup tables, which include building a priority queue for each load bucket, and eliminating configurations that seldom occur, and controlling the table size using function approximations as in [162, 222].

CHAPTER 7

Conclusion

This section summarises what the discussions and accomplishments in this thesis are. We began this thesis by addressing the energy issues with two different paradigms in data centre environments.

One of these paradigms, performance and power estimation, is difficult due to shared memory and CPU resources in data centre environments. As our experiments have demonstrated, the (thread) performance delivered and power consumed is heavily influenced by the contention for CPU resources and shared memory (these amongst other things). However, data centre administrators need information to maintain service level agreements (like batch throughput) while ensuring power is not over budgeted, and this raises the need to understand how applications' demand varies across different hardware settings (configurations). The question becomes: *how do we enable performance and power trade-offs so that the administrators have control?*

We designed *Runtime Estimation of Performance and Power (**REPP**)* to allow fine-grained estimation and control of performance and power. REPP makes use of the hardware performance monitoring counters (PMCs) to estimate performance and power across numerous configurations: dynamic voltage and frequency state (DVFS), Cl-States (intel_powerclamp), and core consolidation. REPP models performance and power for a single-core using PMCs to scale across many/multicore processors, but then uses a model to estimate the impact of contention for shared memory and CPU. The extent to which such sharing can impact performance and power can heavily depend on the workload type and its corunners. The system is designed to also detect when to shut down cores and decide which workloads to collocate so that the system's energy efficiency is maximised. The extent to which such models can be deployed depends on how much information can be obtained regarding the system-application interaction, and how many changes to the hardware can be made. This raises the need

for a scientific inquiry into the PMCs, hardware control settings, workloads and their system interaction. In Part II, we demonstrated that REPP is application agnostic and can be seamlessly integrated into any multicore architecture to estimate and control performance and power.

The other paradigm we discussed is one which allocates a latency-critical workload to cores such that the real-time performance guarantees are met while improving resource utilisation.

In this context, we proposed **Hipster**: a hybrid approach to solve this problem. Our system takes as input the performance guarantee, current latency, load, for a latency-critical job, and the static ordering of configurations of the system in order of power efficiency. Typically, a latency-critical job is allocated resources in response to changes in measured latency, and any remaining resources are allocated to batch jobs to maximise resource efficiency. We make the observation that different latency-critical jobs have different orderings of power efficient configurations that maximise the energy efficiency while meeting real-time performance guarantees. The question becomes: *how do we detect on-the-fly the best configurations for a latency-critical job on any architecture?* We developed a hybrid scheme that combines heuristics and reinforcement learning to manage heterogeneous cores with DVFS control for improved energy efficiency. In Part III, we have shown that Hipster performs well across workloads and interactively adapts the system by learning from the QoS/power/performance history to best map workloads to the heterogeneous cores and adjust their DVFS settings.

The objectives of maximising performance under power control and energy efficient computing could be viewed as complementing each other. But we found that they are usually tightly coupled, and pursuing one objective requires a trade-off with the other. We also noticed that to improve resource efficiency, we have to manage workloads to cores depending on application preferences. We were interested in building a data centre management solution one step at a time. First, to maximise performance subjected to a power bound. Next, to enable energy efficient computing and thereafter improve resource efficiency. We conclude that the aforementioned multi-objective problems do exist, but systems like **REPP** and **Hipster** can be quintessential for administrators' much needed control. The icing on the cake is that such systems can reduce costs by either saving power or meeting performance guarantees!

Publication List

This thesis is based in part on the following published papers:

① **Rajiv Nishtala**, Marc Gonzalez Tallada, Xavier Martorell, "A Methodology to Build Models and Predict Performance-Power in CMPs", in the 44th International Conference on Parallel Processing Workshops (ICPPW), Beijing, China, Sep. 1-4, 2015.

② **Rajiv Nishtala**, Xavier Martorell, Vinicius Petrucci, Daniel Mossé, "REPP-H: Runtime Estimation of Power and Performance on Heterogeneous Data Centers", in the 28th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD), Los Angeles, USA, Oct. 26-28, 2016.

③ **Rajiv Nishtala**, Xavier Martorell, "RePP-C: Runtime Estimation of Performance-Power with Workload Consolidation in CMPs", in the 7th International Green and Sustainable Computing Conference (IGSC), Hangzhou, China, Nov. 7-9, 2016.

④ **Rajiv Nishtala**, Paul Carpenter, Vinicius Petrucci, Xavier Martorell, "Hipster: Hybrid Task Manager for Latency-Critical Cloud Workloads", in the 23rd IEEE Symposium on High Performance Computer Architecture (HPCA) 2017, Austin, USA, Feb. 4-8, 2017.

The following publication is under review:

① **Rajiv Nishtala**, Paul Carpenter, Vinicius Petrucci, Xavier Martorell, "The Hipster Approach for Improving Cloud System Efficiency" in ACM Transactions of Computer Systems (ACM TOCS) Feb. 2017.

The following paper was my masters' work, and not included in this thesis:

① **Rajiv Nishtala**, Daniel Mossé, Vinicius Petrucci, "Energy-aware thread co-location in heterogeneous multicore processors", in the 11th ACM International Conference on Embedded Software (EMSOFT), Montreal, Canada, Sep. 29 - Oct. 4, 2013.

CHAPTER 8

Future Directions

With big data explosion, there is a rise in data centre requirements to manage workload processing time. To manage big-data workloads, data centres often require management solutions that revolve around performance guarantees, energy efficiency and QoS (these amongst other things). The solutions described in my thesis will become increasingly important as long as big-data explosion will continue, and thereby the need for better/smarter and faster computing services to process them. The remainder of this chapter describes some of the ideas that I would have liked to develop as a part of my thesis, but could not due to lack of time.

Part II of the thesis has mainly focused on performance and power prediction for sequential workloads (single thread and multiprogrammed workloads) using statistical models. These predictors have been used to maintain power and (or) performance targets in multicore processors. However, multithreaded workloads have been out of the scope of the evaluation, and the following ideas could not be tested:

Multithreaded workloads for REPP. It could be interesting to consider multithreaded workloads and improve the prediction model to adapt to such workloads. However, to understand where the bottlenecks for performance are, one could potentially look at some of the well-known challenges: (1) Highly scalable programs which offer additional performance often have threads which are independent and have less communication between each other. (2) Programs that do not scale well often are bound by threads that are dependent on each other with communication and synchronisations. This implies CPU time may be wasted waiting for other threads to respond. One of the avenues to determine if a thread is scalable or not could be the processor utilisation.

Part III of the thesis has mainly focused on maintaining QoS for latency-critical workloads on a (relatively) small infrastructure using a hybrid reinforcement learning mechanism. With our approach, we could determine on-the-fly the best configuration to satisfy the QoS for latency-critical jobs while minimising power consumption or maximising utilisation of the server. There are potential improvements for Hipster, and here are some of the ideas that could be tested:

Scaling Hipster on larger infrastructures. It could be interesting to run Hipster on a multi-node data centres in a master-slave scheduling architecture, where an instance of Hipster could be initiated on each node. In contrast to the current implementation of Hipster, the master scheduler may be responsible for allocating a specified set of workloads to a given node, whereas the slave scheduler could be used to allocate resources for a job on a node. The master-slave scheduling architecture could function in tandem to ensure the workload and (or) resource allocation is distributed uniformly across nodes in a cluster environment.

Request-level QoS guarantee. Traditionally, interactive workload scheduling algorithms allocate the workload (with multiple threads) to a given set of hardware resources to ensure application-level QoS metrics are satisfied. This implies that current algorithms are agnostic to the load on each thread (query length, the number of requests, etc.) and allocate (same) resources without knowledge of thread requirements. Our research has shown that query length (amongst other factors) influences the time taken to process each query for each thread.

In this context, we could identify functions where the thread spends a large chunk of the query processing time, and we refer to such functions as hot functions. These hot functions assist the scheduling algorithm in determining how long the thread/query has been in the pipeline. With this timing information, our scheduling algorithm would provide an initial allocation of resources to the thread, depending on the query length, number of queries, etc., in order to minimise energy consumption while generally satisfying the latency bound. The scheduler will monitor thread progress and, if necessary, allocate additional resources to ensure that the latency bound is satisfied.

Distributed workloads for REPP and Hipster. The current implementation of REPP and Hipster were conducted on stand-alone servers. It could be interesting to extend the research to distributed servers, where each server needs to communicate to exchange data. Distributed workloads such as Memcached and Elasticsearch are

typical examples of fan-out applications, wherein a query is processed on multiple servers and the query tail latency is not just dependent on the resource contention for a given server but on multiple servers. One way to approach this problem is by allocating each request to a specified set of resources and the feedback from the PMCs could be used to understand the intensity of the request and thereby satisfy thread level QoS requirements.

In summary, this thesis has demonstrated a methodology to build models and predict performance and power for single-threaded and multiprogrammed workloads in multicore processors, and an algorithm to maintain QoS for latency-critical workloads. Our results have shown substantial improvements beyond the prior state-of-the-art, and further improvements to our current algorithms can be achieved by following the ideas for future work outlined in this section.

APPENDIX A

Multiprogrammed benchmarks

We categorise the batch workloads into multiprogrammed batch workloads based on the methodology described in section 2.6.1. The multiprogrammed are generated from single threaded batch workloads. The single threaded batch workloads used are 22 SPEC CPU 2006 [104, 105], nine PARSEC 3.0 [106], six NAS [107], 11 SPLASH2x [108]. These workloads exhibit both memory and computational bound phases. The number of benchmarks in a multiprogrammed workload is equal to the number of cores in each architecture, thereby, having 35 workloads of four benchmarks on AMD and Intel, and ten workloads of two benchmarks on ARM. On the ARM processor, we use only the big cluster as the small cluster does not allow us to gather most counters. The methodology to build the workloads is described by Sanchez and Kozyrakis [109]. Tables 1.1 and 1.2 show the multiprogrammed workloads on ARM, AMD and Intel.

Table 1.1 Multiprogrammed workloads on ARM

Multiprogrammed workloads	Category
radiosity,streamcluster	FF
streamcluster,ep.c	FN
fluidanimate,is.c	NN
sp.c,facesim	SF
sp.c,blackscholes	SN
sp.c,mg.c	SS
ocean_cp,water_spatial	ST
radix,water_nsquared	TF
radix,lu_cb	TN
radix,radix	TT

Table 1.2 Multiprogrammed workloads on AMD and Intel

Multiprogrammed workloads (AMD)	Multiprogrammed workloads (Intel)	Category
lu_ncb bodytrack x264 fmm	libquantum, wrf, fmm, water_spatial	FFFF
vips cactusadm wrf water_nsquared	barnes water_spatial fmm hmmer	FFFN
wrf streamcluster hmmer gromacs	cactusadm lbm radiosity omnetpp	FFNN
lu_ncb blackscholes calculix ep.c	fmm omnetpp volrend facesim	FNNN
raytrace freqmine freqmine tonto	lu_cb povray radiosity tonto	NNNN
milc fmm bodytrack canneal	mcf lu_ncb cg.c lbm	SFFF
lu.c zeusmp streamcluster omnetpp	streamcluster fmm zeusmp bzip2	SFFN
libquantum zeusmp calculix, water_spatial	lu.c lu_ncb fluidanimate raytrace	SFNN
dedup hmmer gobmk povray	mg.c tonto fft fft	SNNN
gemsfdtd libquantum bodytrack, streamcluster	lu.c sp.c cactusadm fmm	SSFF
libquantum mg.c canneal tonto	lu.c mcf sjeng dedup	SSFN
lu.c lbm freqmine sjeng	mg.c sp.c radiosity ep.c	SSNN
sp.c lu.c milc canneal	milc lu.c milc water_spatial	SSSF
mcf gemsfdtd mg.c raytrace	milc milc xalancbmk blackscholes	SSSN
lu.c sp.c mcf gemsfdtd	xalancbmk xalancbmk streamcluster sp.c	SSSS
libquantum milc dedup soplex	mg.c lu.c xalancbmk soplex	SSST
mg.c dedup bwaves fmm	milc lu.c radix x264	SSTF
lbm sp.c soplex bzip2	lu.c lu.c astar raytrace	SSTN
mcf libquantum radix radix	streamcluster lu.C bwaves soplex	SSTT
libquantum astar x264 bodytrack	lu.c astar ocean_cp libquantum	STFF
milc astar bodytrack radiosity	xalancbmk astar zeusmp water_nsquared	STFN
gemsfdtd soplex hmmer radiosity	mcf soplex freqmine bodytrack	STNN
lbm soplex soplex bodytrack	milc astar soplex cg.c	STTF
dedup radix astar freqmine	mg.c astar astar bodytrack	STTN
libquantum astar astar radix	mcf canneal radix canneal	STTT
astar canneal canneal lu_ncb	astar fmm cactusadm water_spatial	TFFF
bwaves canneal canneal calculix	radix cg.c fmm vips	TFFN
astar x264 povray raytrace	soplex x264 ep.c tonto	TFNN
astar omnetpp omnetpp, water_spatial	gemsfdtd radiosity fft namd	TNNN
astar bwaves zeusmp vips	radix soplex sjeng barnes	TTFF
radix soplex bodytrack water_nsquared	radix radix cg.c radiosity	TTFN
astar astar blackscholes bzip2	soplex radix vips gromacs	TTNN
radix soplex bwaves zeusmp	canneal bwaves radix x264	TTTF
radix astar astar namd	bwaves soplex gemsfdtd is.c	TTTN
soplex astar astar bwaves	radix astar bwaves soplex	TTTT

Part V:

Bibliography

There are no passengers on spaceship earth. We are all crew.

Marshall McLuhan

References

[1] Joseph M. Williams. Style Towards Clarity and Grace, February 2017. unalmed.edu.co/~poboyca/documentos/Doc.%20Seminario%20I/style.pdf.

[2] Pierre Delforge and Josh Whitney. Data Center Efficiency Assessment. *Natural Resources Defense Council (NRDC)*, May 2014. nrdc.org/sites/default/files/data-center-efficiency-assessment-IP.pdf.

[3] Forbes. Berkeley Lab: It Takes 70 Billion Kilowatt Hours A Year To Run The Internet, June 2016. forbes.com/sites/christopherhelman/2016/06/28/how-much-electricity-does-it-take-to-run-the-internet.

[4] Natural Resource Defense Council (NRDC). The Power of Efficiency to Cut Data Center Energy Waste, June 2016. nrdc.org/experts/pierre-delforge/power-efficiency-cut-data-center-energy-waste.

[5] MIT Technology Review. Intel Puts the Brakes on Moore's Law, May 2016. technologyreview.com/s/601102/intel-puts-the-brakes-on-moores-law/.

[6] Mont-Blanc. Mont-Blanc, December 2016. montblanc-project.eu/.

[7] Ilkka Tuomi. The Lives and Death of Moore's Law. *First Monday*, 7(11), December 2002.

[8] Facebook Inc. Facebook, November 2016. facebook.com.

[9] Google Inc. Google search, November 2016. google.es.

[10] Twitter Inc. Twitter, November 2016. twitter.com.

[11] Amazon EC2. Amazon EC2 pricing, November 2016. aws.amazon.com/ec2/pricing/on-demand/.

[12] Rackspace. Rackspace, November 2016. rackspace.com.

[13] James Hamilton. Cost of Power in Large-Scale Data Centers, December 2016. perspectives.mvdirona.com/2008/11/cost-of-power-in-large-scale-data-centers/.

[14] Michael Allen. Data Center Power Costs and Requirements, December 2016. datacenters.com/news/infrastructure/135-data-center-power-costs-and-requirements.

[15] Arka A. Bhattacharya, David Culler, Aman Kansal, Sriram Govindan, and Sriram Sankar. The Need for Speed and Stability in Data Center Power Capping. In *Proceedings of the 2012 International Green Computing Conference (IGCC)*, IGCC '12, pages 1–10, Washington, DC, USA, 2012. IEEE Computer Society.

[16] Qiang Wu, Qingyuan Deng, Lakshmi Ganesh, Chang-Hong Hsu, Yun Jin, Sanjeev Kumar, Bin Li, Justin Meza, and Yee Jiun Song. Dynamo: Facebook's Data Center-Wide Power Management System. In *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, pages 469–480, June 2016.

[17] Schurman Eric and Brutlag Jake. The User and Business Impact of Server Delays, Additional Bytes, and HTTP Chunking in Web Search. *Velocity*, 2009.

[18] Facebook Inc. Luleå goes live, November 2016. facebook.com.

[19] Charles Lefurgy, Karthick Rajamani, Freeman Rawson, Wes Felter, Michael Kistler, and Tom W. Keller. Energy Management for Commercial Servers. *Computer*, 36(12):39–48, December 2003.

[20] Emre Kültürsay, Mahmut Kandemir, Anand Sivasubramaniam, and Onur Mutlu. Evaluating STT-RAM as an energy-efficient main memory alternative. In *2013 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pages 256–267, April 2013.

[21] Ibrahim Hur and Calvin Lin. A comprehensive approach to DRAM power management. In *2008 IEEE 14th International Symposium on High Performance Computer Architecture*, pages 305–316, Feb 2008.

[22] Luiz André Barroso, Jimmy Clidaras, and Urs Hölzle. The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines, Second edition. *Synthesis Lectures on Computer Architecture*, 8(3):1–154, 7 2013.

[23] John Carter and Karthick Rajamani. Designing Energy-Efficient Servers and Data Centers. *Computer*, 43(7):76–78, July 2010.

[24] Vinicius Petrucci, Michael A. Laurenzano, John Doherty, Yunqi Zhang, Daniel Mosse, Jason Mars, and Lingjia Tang. Octopus-Man: QoS-driven task management for heterogeneous multicores in warehouse-scale computers. In *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*, pages 246–258. IEEE, 2 2015.

[25] Christina Delimitrou and Christos Kozyrakis. Quasar: resource-efficient and QoS-aware cluster management. In *Proceedings of the 19th international conference on Architectural support for programming languages and operating systems - ASPLOS '14*, pages 127–144, New York, New York, USA, 2014. ACM Press.

[26] George Prekas, Mia Primorac, Adam Belay, Christos Kozyrakis, and Edouard Bugnion. Energy Proportionality and Workload Consolidation for Latency-critical Applications. In *Proceedings of the Sixth ACM Symposium on Cloud Computing*, SoCC '15, pages 342–355, New York, NY, USA, 2015. ACM.

[27] Luiz André Barroso, Jeffrey Dean, and Urs Hölzle. Web Search for a Planet: The Google Cluster Architecture. *IEEE Micro*, 23(2):22–28, March 2003.

[28] David Meisner, Christopher M. Sadler, Luiz André Barroso, Wolf-Dietrich Weber, and Thomas F. Wenisch. Power Management of Online Data-intensive Services. *SIGARCH Comput. Archit. News*, 39(3):319–330, June 2011.

[29] Ozlem Bilgir, Margaret Martonosi, Qiang Wu, and Facebook Inc. Exploring the Potential of CMP Core Count Management on Data Center Energy Savings. In *Proceedings of the Workshop on Energy Efficient Design (WEED)*, June 2011.

[30] Rajiv Nishtala, Daniel Mossé, and Vinicius Petrucci. Energy-aware Thread Co-location in Heterogeneous Multicore Processors. In *Proceedings of the Eleventh ACM International Conference on Embedded Software*, EMSOFT '13, pages 21:1–21:9, Piscataway, NJ, USA, 2013. IEEE Press.

[31] Sergey Zhuravlev, Juan Carlos Saez, Sergey Blagodurov, Alexandra Fedorova, and Manuel Prieto. Survey of Energy-Cognizant Scheduling Techniques. *IEEE Trans. Parallel Distrib. Syst.*, 24(7):1447–1464, July 2013.

[32] Sergey Blagodurov, Sergey Zhuravlev, and Alexandra Fedorova. Contention-Aware Scheduling on Multicore Systems. *ACM Trans. Comput. Syst.*, 28(4):8:1–8:45, December 2010.

[33] Vinicius Petrucci, Orlando Loques, and Daniel Mossé. Lucky Scheduling for Energy-efficient Heterogeneous Multi-core Systems. In *Proceedings of the 2012 USENIX Conference on Power-Aware Computing and Systems*, HotPower'12, pages 7–7, Berkeley, CA, USA, 2012. USENIX Association.

[34] Vinicius Petrucci, Orlando Loques, Daniel Mossé, Rami Melhem, Neven Abou Gazala, and Sameh Gobriel. Energy-Efficient Thread Assignment Optimization for Heterogeneous Multicore Systems. *ACM Trans. Embed. Comput. Syst.*, 14(1):15:1–15:26, January 2015.

[35] Darko Zivanovic, Milan Radulovic, Germán Llort, David Zaragoza, Janko Strassburg, Paul M. Carpenter, Petar Radojković, and Eduard Ayguadé. Large-Memory Nodes for Energy Efficient High-Performance Computing. In *Proceedings of the*

Second International Symposium on Memory Systems, MEMSYS '16, pages 3–9, New York, NY, USA, 2016. ACM.

[36] Kazi Asifuzzaman, Milan Pavlovic, Milan Radulovic, David Zaragoza, Ohseong Kwon, Kyung-Chang Ryoo, and Petar Radojković. Performance Impact of a Slower Main Memory: A Case Study of STT-MRAM in HPC. In *Proceedings of the Second International Symposium on Memory Systems*, MEMSYS '16, pages 40–49, New York, NY, USA, 2016. ACM.

[37] Chuanxiong Guo, Lihua Yuan, Dong Xiang, Yingnong Dang, Ray Huang, Dave Maltz, Zhaoyi Liu, Vin Wang, Bin Pang, Hua Chen, Zhi-Wei Lin, and Varugis Kurien. Pingmesh: A Large-Scale System for Data Center Network Latency Measurement and Analysis. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, SIGCOMM '15, pages 139–152, New York, NY, USA, 2015. ACM.

[38] Jason Mars and Lingjia Tang. Whare-map: heterogeneity in "homogeneous" warehouse-scale computers. *ACM SIGARCH Computer Architecture News*, 41(3):619, 7 2013.

[39] Bo Su, Junli Gu, Li Shen, Wei Huang, Joseph L. Greathouse, and Zhiying Wang. PPEP: Online Performance, Power, and Energy Prediction Framework and DVFS Space Exploration. In *Proceedings of the 47th Annual IEEE/ACM Int'l Symposium on Microarchitecture*, MICRO-47, pages 445–457. IEEE Computer Society, 2014.

[40] David Lo, Liqun Cheng, Rama Govindaraju, Parthasarathy Ranganathan, and Christos Kozyrakis. Heracles: improving resource efficiency at scale. *ACM SIGARCH Computer Architecture News*, 43(3):450–462, 6 2015.

[41] Jeffrey Dean and Luiz André Barroso. The Tail at Scale. *Commun. ACM*, 56(2):74–80, February 2013.

[42] Ripal Nathuji, Aman Kansal, and Alireza Ghaffarkhah. Q-clouds: managing performance interference effects for QoS-aware clouds. In *Proceedings of the 5th European conference on Computer systems - EuroSys '10*, page 237, New York, New York, USA, 4 2010. ACM Press.

[43] Matthew Halpern, Yuhao Zhu, and Vijay Janapa Reddi. Mobile CPU's rise to power: Quantifying the impact of generational mobile CPU design trends on performance, energy, and user satisfaction. In *2016 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 64–76. IEEE, 3 2016.

[44] Daniel Wong and Murali Annavaram. KnightShift: Scaling the Energy Proportionality Wall through Server-Level Heterogeneity. In *2012 45th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 119–130. IEEE, 12 2012.

[45] Vijay Janapa Reddi, Benjamin C. Lee, Trishul Chilimbi, and Kushagra Vaid. Web Search Using Mobile Cores: Quantifying and Mitigating the Price of Efficiency. In *Proceedings of the 37th Annual International Symposium on Computer Architecture*, ISCA '10, pages 314–325, New York, NY, USA, 2010. ACM.

[46] Nagabhushan Chitlur, Ganapati Srinivasa, Scott Hahn, P K Gupta, Dheeraj Reddy, David Koufaty, Paul Brett, Abirami Prabhakaran, Li Zhao, Nelson Ijih, Suchit Subhaschandra, Sabina Grover, Xiaowei Jiang, and Ravi Iyer. QuickIA: Exploring heterogeneous architectures on real prototypes. In *IEEE International Symposium on High-Performance Comp Architecture*, pages 1–8. IEEE, 2 2012.

[47] Marisabel Guevara, Benjamin Lubin, and Benjamin C. Lee. Navigating heterogeneous processors with market mechanisms. In *2013 IEEE 19th International Symposium on High Performance Computer Architecture (HPCA)*, pages 95–106. IEEE, 2 2013.

[48] Andrew Lukefahr, Shruti Padmanabha, Reetuparna Das, Ronald Dreslinski, Thomas F. Wenisch, and Scott Mahlke. Heterogeneous microarchitectures trump voltage scaling for low-power cores. In *2014 23rd International Conference on Parallel Architecture and Compilation Techniques (PACT)*, pages 237–249, Aug 2014.

[49] Vijay Janapa Reddi, Benjamin C. Lee, Trishul Chilimbi, and Kushagra Vaid. Web Search Using Mobile Cores: Quantifying and Mitigating the Price of Efficiency. *SIGARCH Comput. Archit. News*, 38(3):314–325, June 2010.

[50] Mont-Blanc 2. Mont-Blanc 2, European scalable and power efficient HPC platform based on low-power embedded technology, December 2016. cordis.europa.eu/project/rcn/110381_en.html.

[51] Alex Ramirez. The Mont-Blanc Prototype, October 2016. montblanc-project.eu/sites/default/files/publications/20140522montblanc-prototypes-workshop.pdf.

[52] Nikola Rajovic, Paul M. Carpenter, Isaac Gelado, Nikola Puzovic, Alex Ramirez, and Mateo Valero. Supercomputing with Commodity CPUs: Are Mobile SoCs Ready for HPC? In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, SC '13, pages 40:1–40:12, New York, NY, USA, 2013. ACM.

[53] John Russell. ARM Unveils Scalable Vector Extension for HPC at Hot Chips, October 2016. hpcwire.com/2016/08/22/arm-unveils-scalable-vector-extension-hpc-hot-chips/.

[54] Adrian Cristal, Daniel Ortega, Josep Llosa, and Mateo Valero. Out-of-Order Commit Processors. In *Proceedings of the 10th International Symposium on High Performance Computer Architecture*, HPCA '04, pages 48–, Washington, DC, USA, 2004. IEEE Computer Society.

[55] Sergey Zhuravlev, Juan Carlos Saez, Sergey Blagodurov, Alexandra Fedorova, and Manuel Prieto. Survey of Scheduling Techniques for Addressing Shared Resources in Multicore Processors. *ACM Comput. Surv.*, 45(1):4:1–4:28, December 2012.

[56] Jason Mars, Lingjia Tang, Robert Hundt, Kevin Skadron, and Mary Lou Soffa. Bubble-Up: Increasing Utilization in Modern Warehouse Scale Computers via Sensible Co-locations. In *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO-44, pages 248–259, New York, NY, USA, 2011. ACM.

[57] Jason Mars, Lingjia Tang, and Mary Lou Soffa. Directly Characterizing Cross Core Interference Through Contention Synthesis. In *Proceedings of the 6th International Conference on High Performance and Embedded Architectures and Compilers*, HiPEAC '11, pages 167–176, New York, NY, USA, 2011. ACM.

[58] Hailong Yang, Alex Breslow, Jason Mars, and Lingjia Tang. Bubble-flux: precise online QoS management for increased utilization in warehouse scale computers. *ACM SIGARCH Computer Architecture News*, 41(3):607, 7 2013.

[59] Rance Rodrigues, Arunachalam Annamalai, Israel Koren, and Sandip Kundu. A Study on the Use of Performance Counters to Estimate Power in Microprocessors. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 60(12):882–886, Dec 2013.

[60] Harshad Kasture, Davide B. Bartolini, Nathan Beckmann, and Daniel Sanchez. Rubik: fast analytical power management for latency-critical systems. In *Proceedings of the 48th International Symposium on Microarchitecture - MICRO-48*, pages 598–610, New York, New York, USA, 12 2015. ACM Press.

[61] Mohammad Alian, H.M.O Ahmed Abulila, Lokesh Jindal, Daehoon Kim, and Nam Sung Kim. NCAP: Network-Driven, Packet Context-Aware Power Management for Client-Server Architecture. In *2017 IEEE 23rd International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2 2017.

[62] Bull Atos Technology. Mont-Blanc European project gains new impetus for its climb to Exascale, December 2016. bull.com/mont-blanc-european-project-gains-new-impetus-its-climb-exascale.

[63] John Russell. ARM Waving: Attention, Deployments, and Development, February 2017. hpcwire.com/2017/01/18/arm-waving-gathering-attention/.

[64] Samsung. Samsung Exynos 5 Dual, February 2017. samsung.com/semiconductor/minisite/Exynos.

[65] ARM. ARM Cortex-15 processor, February 2017. arm.com.

[66] ARM. Mali graphics processing from ARM, February 2017. arm.com.

[67] Jason Mars, Lingjia Tang, and Robert Hundt. Heterogeneity in 'Homogeneous' Warehouse-Scale Computers: A Performance Opportunity. *IEEE Computer Architecture Letters*, 10(2):29–32, 2 2011.

[68] Jason Cong and Bo Yuan. Energy-efficient scheduling on heterogeneous multi-core architectures. In *Proceedings of the 2012 ACM/IEEE international symposium on Low power electronics and design - ISLPED '12*, page 345, New York, New York, USA, 7 2012. ACM Press.

[69] Violaine Villebonnet, Georges Da Costa, Laurent Lefèvre, Jean-Marc Pierson, and Patricia Stolf. "Big, Medium, Little": Reaching Energy Proportionality with Heterogeneous Computing Scheduler. *Parallel Processing Letters*, 25(3), 2015.

[70] Ryan Cochran, Can Hankendi, Ayse K. Coskun, and Sherief Reda. Pack & Cap: Adaptive DVFS and Thread Packing Under Power Caps. In *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO-44, pages 175–185, New York, NY, USA, 2011. ACM.

[71] Michael Huang, Jose Renau, Seung-Moon Yoo, and Josep Torrellas. A framework for dynamic energy efficiency and temperature management. In *Proceedings of the 33rd annual ACM/IEEE international symposium on Microarchitecture - MICRO 33*, pages 202–213, New York, New York, USA, 2000. ACM Press.

[72] 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 *2006 39th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'06)*, pages 347–358. IEEE, 12 2006.

[73] Barry Rountree, David Lowenthal, Martin Schulz, and Bronis R. de Supinski. Practical performance prediction under Dynamic Voltage Frequency Scaling. In *2011 International Green Computing Conference and Workshops*, IGCC 2011, pages 1–8, July 2011.

[74] Karan Singh, Major Bhadauria, and Sally A. McKee. Real time power estimation and thread scheduling via performance counters. *ACM SIGARCH Computer Architecture News*, 37(2):46, 7 2009.

[75] C. Michael Olsen and Chandra Narayanaswami. PowerNap: An Efficient Power Management Scheme for Mobile Devices. *IEEE Transactions on Mobile Computing*, 5(7):816–828, July 2006.

[76] A. Leonard Brown. The State of ACPI in the Linux Kernel. *SIGARCH Comput. Archit. News*, 1(1):121–132, 2004.

[77] Dominik Brodowski. CPU frequency and voltage scaling code in the Linux kernel, February 2017. kernel.org/doc/Documentation/cpu-freq/governors.txt.

[78] Intel. *Intel 64 and IA-32 Architecture Software Developer's Manual*, November 2011.

[79] Robert Schöne, Daniel Molka, and Michael Werner. Wake-up Latencies for Processor Idle States on Current x86 Processors. *Comput. Sci.*, 30(2):219–227, May 2015.

[80] Stephanie Eranian. Issues with libpfm4.7.0 and perf on ARM Juno r0, April 2016. sourceforge.net/p/perfmon2/mailman/message/34954001/.

[81] Intel. Power Clamp Driver, October 2016. kernel.org/doc/Documentation/thermal/intel_powerclamp.txt.

[82] AMD. *AMD64 Architecture Prog. Manual Volume 2: System Prog.*, September 2016.

[83] ARM. *Infocenter for ARM Cortex-A57*, September 2016.

[84] Applied Micro. Applied Micro XGene2, March 2016. goo.gl/XA04r1.

[85] Linux. Perf: Linux profiling with performance counters, December 2016. perf.wiki.kernel.org/index.php/Main{_}Page.

[86] Gang Ren, Eric Tune, Tipp Moseley, Yixin Shi, Silvius Rus, and Robert Hundt. Google-Wide Profiling: A Continuous Profiling Infrastructure for Data Centers. *IEEE Micro*, 30(4):65–79, 7 2010.

[87] Svilen Kanev, Juan Pablo Darago, Kim Hazelwood, Parthasarathy Ranganathan, Tipp Moseley, Gu-Yeon Wei, and David Brooks. Profiling a Warehouse-scale Computer. In *Proceedings of the 42nd Annual International Symposium on Computer Architecture*, ISCA '15, pages 158–169, New York, NY, USA, 2015. ACM.

[88] Stephanie Eranian. Perfmon2: Improving Performance Monitoring on Linux, April 2016. perfmon2.sourceforge.net.

[89] David Bernstein. Containers and Cloud: From LXC to Docker to Kubernetes. *IEEE Cloud Computing*, 1(3):81–84, 9 2014.

[90] ARM Limited. ARM ® Cortex ® -A53 MPCore Processor Technical Reference Manual. infocenter.arm.com/help/topic/com.arm.doc.ddi0500g/DDI0500G{_}cortex{_}a53{_}trm.pdf.

[91] ARM Limited. ARM ® Cortex ® -A57 MPCore Processor Revision: r1p0 Technical Reference Manual. infocenter.arm.com/help/topic/com.arm.doc.ddi0488c/DDI0488C{_}cortex{_}a57{_}mpcore{_}r1p0{_}trm.pdf.

[92] H. Peter Anvin. MSR Tools, March 2016. kernel.org/pub/linux/utils/cpu/msr-tools/.

[93] Wattsup Pro. Wattsup Pro, April 2016. wattsupmeters.com/secure/products.php.

[94] ARM. ARM Juno Power, April 2016. goo.gl/JryrgT.

[95] ARM. ARM Juno Power Registers, April 2016. github.com/ARM-software/devlib/blob/master/src/readenergy/readenergy.c".

[96] ARM. SYS_POW_SYS Register, April 2016. goo.gl/fmTTQi.

[97] Michael K. Patterson. The effect of data center temperature on energy efficiency. In *Thermal and Thermomechanical Phenomena in Electronic Systems, 2008. ITHERM 2008. 11th Intersociety Conference on*, pages 1167–1174, May 2008.

[98] Jörg Henkel, Santiago Pagani, Heba Khdr, Florian Kriebel, Semeen Rehman, and Muhammad Shafique. Towards Performance and Reliability-efficient Computing in the Dark Silicon Era. In *Proceedings of the 2016 Conference on Design, Automation & Test in Europe*, DATE '16, pages 1–6, San Jose, CA, USA, 2016. EDA Consortium.

[99] Google. Designing efficient data centers, September 2016. google.com/green/efficiency/datacenters/.

[100] Nathan Fisher, Jian-Jia Chen, Shengquan Wang, and Lothar Thiele. Thermal-Aware Global Real-Time Scheduling on Multicore Systems. In *Proceedings of the 2009 15th IEEE Symposium on Real-Time and Embedded Technology and Applications*, RTAS '09, pages 131–140, Washington, DC, USA, 2009. IEEE Computer Society.

[101] The Guardian. The node pole: inside Facebook's Swedish hub near the Arctic Circle, September 2016. theguardian.com/technology/2015/sep/25/facebook-datacentre-lulea-sweden-node-pole.

[102] Facebook. Facebook is opening a new wind-powered data center in Texas. goo.gl/dKVnSB.

[103] David Lo, Liqun Cheng, Rama Govindaraju, Luiz André Barroso, and Christos Kozyrakis. Towards energy proportionality for large-scale latency-critical workloads. *ACM SIGARCH Computer Architecture News*, 42(3):301–312, 10 2014.

[104] Tribuvan Kumar Prakash and Lu Peng. Performance Characterization of SPEC CPU2006 Benchmarks on Intel Core 2 Duo Processor. In *Proceedings of the ISAST Transactions on Computers and Software Engineering*, ISAST 2008, pages 36 –41. IEEE, 2008.

[105] John L. Henning. SPEC CPU2006 benchmark descriptions. *ACM SIGARCH Computer Architecture News*, 34(4):1–17, 9 2006.

[106] Christian Bienia, Sanjeev Kumar, Jaswinder Pal Singh, and Kai Li. The PARSEC Benchmark Suite: Characterization and Architectural Implications. In *Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques*, PACT '08, pages 72–81, New York, NY, USA, 2008. ACM.

[107] D. H. Bailey, E. Barszcz, J. T. Barton, D. S. Browning, R. L. Carter, L. Dagum, R. A. Fatoohi, P. O. Frederickson, T. A. Lasinski, R. S. Schreiber, H. D. Simon, V. Venkatakrishnan, and S. K. Weeratunga. The NAS Parallel Benchmarks;Summary and Preliminary Results. In *Proceedings of the 1991 ACM/IEEE Conference on Supercomputing*, Supercomputing '91, pages 158–165, New York, NY, USA, 1991. ACM.

[108] Steven C. Woo, Moriyoshi Ohara, Evan Torrie, Jaswinder P. Singh, and Anoop Gupta. The SPLASH-2 programs: characterization and methodological considerations. In *22nd International Symposium on Computer Architecture*, ISCA 1995, pages 24–36, June 1995.

[109] Daniel Sanchez and Christos Kozyrakis. Vantage: Scalable and Efficient Fine-grain Cache Partitioning. In *Proceedings of the 38th Annual International Symposium on Computer Architecture*, ISCA '11, pages 57–68, New York, NY, USA, 2011. ACM.

[110] Memcached. Memcached, March 2016. memcached.org.

[111] ElasticSearch. Elasticsearch, March 2016. github.com/elastic/elasticsearch.

[112] Kaushik Veeraraghavan, Justin Meza, David Chou, Wonho Kim, Sonia Margulis, Scott Michelson, Rajesh Nishtala, Daniel Obenshain, Dmitri Perelman, and Yee Jiun Song. Kraken: Leveraging Live Traffic Tests to Identify and Resolve Resource Utilization Bottlenecks in Large Scale Web Services. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pages 635–651, GA, 2016. USENIX Association.

[113] Rajesh Nishtala, Hans Fugal, Steven Grimm, Marc Kwiatkowski, Herman Lee, Harry C. Li, Ryan McElroy, Mike Paleczny, Daniel Peek, Paul Saab, David Stafford, Tony Tung, and Venkateshwaran Venkataramani. Scaling Memcache at Facebook. In *Proceedings of the 10th USENIX conference on Networked Systems Design and Implementation*, pages 385–398. USENIX Association, 2013.

[114] Berk Atikoglu, Yuehai Xu, Eitan Frachtenberg, Song Jiang, and Mike Paleczny. Workload analysis of a large-scale key-value store. In *Proceedings of the 12th ACM SIGMETRICS/PERFORMANCE joint international conference on Measurement and Modeling of Computer Systems - SIGMETRICS '12*, page 53, New York, New York, USA, 2012. ACM Press.

[115] Faban. Faban, March 2016. faban.org.

[116] Michael Ferdman, Babak Falsafi, Almutaz Adileh, Onur Kocberber, Stavros Volos, Mohammad Alisafaee, Djordje Jevdjic, Cansu Kaynak, Adrian Daniel Popescu, and Anastasia Ailamaki. Clearing the clouds: a study of emerging scale-out workloads on modern hardware. *ACM SIGARCH Computer Architecture News*, 40(1):37, 4 2012.

[117] Google. Efficiency: How we do it, October 2016. google.com/about/datacenters/efficiency/internal/.

[118] 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.

[119] Intel Tivoli Management Framework. Intel Intelligent Power Node Manager, June 2016. ibm.com/software/tivoli/products/mgt-framework/.

[120] Ripal Nathuji, Canturk Isci, and Eugene Gorbatov. Exploiting Platform Heterogeneity for Power Efficient Data Centers. In *Proceedings of the Fourth International Conference on Autonomic Computing*, ICAC '07, pages 5–, Washington, DC, USA, 2007. IEEE Computer Society.

[121] Intel. Intel Intelligent Power Node Manager, June 2016. download.intel.com/support/motherboards/server/sysmgmt/sb/node_manager_white_paper.pdf.

[122] Openstack. OpenStack, June 2016. openstack.org.

[123] Karan Singh, Major Bhadauria, and Sally A. McKee. Real Time Power Estimation and Thread Scheduling via Performance Counters. *SIGARCH Comput. Archit. News*, 37(2):46–55, July 2009.

[124] Rajiv Nishtala, Marc González Tallada, and Xavier Martorell. A Methodology to Build Models and Predict Performance-Power in CMPs. In *44th International Conference on Parallel Processing Workshops, ICPPW 2015, Beijing, China, September 1-4, 2015*, pages 193–202, 2015.

[125] Rajiv Nishtala, Xavier Martorell, Vinicius Petrucci, and Daniel Mossé. REPP-H: Runtime Estimation of Power and Performance on Heterogeneous Data Centers. In *2016 28th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*, pages 150–157, Oct 2016.

[126] Rajiv Nishtala and Xavier Martorell. REPP-C: Runtime Estimation of Performance-Power with Workload Consolidation in CMPs. In *Proceedings of the 7th International Green and Sustainable Computing Conference*, IGSC '16. IEEE Press, 2016.

[127] Frank Bellosa. The Benefits of Event: Driven Energy Accounting in Power-sensitive Systems. In *Proceedings of the 9th Workshop on ACM SIGOPS European*

Workshop: Beyond the PC: New Challenges for the Operating System, EW 9, pages 37–42, New York, NY, USA, 2000. ACM.

[128] Adam Lewis, Jim Simon, and Nian-Feng Tzeng. Chaotic Attractor Prediction for Server Run-time Energy Consumption. In *Proceedings of the 2010 International Conference on Power Aware Computing and Systems*, HotPower'10, pages 1–16, Berkeley, CA, USA, 2010. USENIX Association.

[129] Canturk Isci and Margaret Martonosi. Runtime Power Monitoring in High-End Processors: Methodology and Empirical Data. In *Proceedings of the 36th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO 36, pages 93–, Washington, DC, USA, 2003. IEEE Computer Society.

[130] Ramon Bertran, Marc Gonzàlez, Xavier Martorell, Nacho Navarro, and Eduard Ayguadé. Counter-Based Power Modeling Methods. *Comput. J.*, 56(2):198–213, February 2013.

[131] Ramon Bertran, Alper Buyuktosunoglu, Meeta S. Gupta, Marc Gonzalez, and Pradip Bose. Systematic Energy Characterization of CMP/SMT Processor Systems via Automated Micro-Benchmarks. In *Proceedings of the 2012 45th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO-45, pages 199–211, Washington, DC, USA, 2012. IEEE Computer Society.

[132] Ramon Bertran, Yolanda Becerra, David Carrera, Vicenç Beltran, Marc Gonzílez, Xavier Martorell, Nacho Navarro, Jordi Torres, and Eduard Ayguadé. Energy Accounting for Shared Virtualized Environments Under DVFS Using PMC-based Power Models. *Future Gener. Comput. Syst.*, 28(2):457–468, February 2012.

[133] Ramon Bertran, Marc Gonzalez, Xavier Martorell, Nacho Navarro, and Eduard Ayguade. A Systematic Methodology to Generate Decomposable and Responsive Power Models for CMPs. *IEEE Transactions on Computers*, 62(7):1289–1302, 2013.

[134] Trevor E. Carlson, Wim Heirman, and Lieven Eeckhout. Sniper: Exploring the level of abstraction for scalable and accurate parallel multi-core simulation. In *High Performance Computing, Networking, Storage and Analysis (SC), 2011 International Conference for*, pages 1–12, Nov 2011.

[135] Rustam Miftakhutdinov, Eiman Ebrahimi, and Yale N. Patt. Predicting Performance Impact of DVFS for Realistic Memory Systems. In *Proceedings of the 2012 45th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO-45, pages 155–165, Washington, DC, USA, 2012. IEEE Computer Society.

[136] M. Le Thang. A study on Linux Kernel Scheduler Version 2.6.32, March 2016. cfile28.uf.tistory.com/attach/1266C1385075387C151D16.

[137] Charles Lefurgy, Xiaorui Wang, and Malcolm Ware. Power Capping: A Prelude to Power Shifting. *Cluster Computing*, 11(2):183–195, June 2008.

[138] John C. McCullough, Yuvraj Agarwal, Jaideep Chandrashekar, Sathyanarayan Kuppuswamy, Alex C. Snoeren, and Rajesh K. Gupta. Evaluating the Effectiveness of Model-based Power Characterization. In *Proceedings of the 2011 USENIX Conference on USENIX Annual Technical Conference*, USENIXATC'11, pages 12–12, Berkeley, CA, USA, 2011. USENIX Association.

[139] Satoshi Imamura, Hiroshi Sasaki, Koji Inoue, and Dimitrios S. Nikolopoulos. Power-capped DVFS and thread allocation with ANN models on modern NUMA systems. In *Computer Design (ICCD), 2014 32nd IEEE International Conference on*, pages 324–331, Oct 2014.

[140] Michela Becchi and Patrick Crowley. Dynamic Thread Assignment on Heterogeneous Multiprocessor Architectures. In *Proceedings of the 3rd Conference on Computing Frontiers*, CF '06, pages 29–40, New York, NY, USA, 2006. ACM.

[141] Anshul Gandhi, Varun Gupta, Mor Harchol-Balter, and Michael A. Kozuch. Optimality Analysis of Energy-performance Trade-off for Server Farm Management. *Perform. Eval.*, 67(11):1155–1171, November 2010.

[142] David Brooks, Vivek Tiwari, and Margaret Martonosi. Wattch: A Framework for Architectural-level Power Analysis and Optimizations. In *Proceedings of the 27th Annual International Symposium on Computer Architecture*, ISCA '00, pages 83–94, New York, NY, USA, 2000. ACM.

[143] Anshul Gandhi, Mor Harchol-Balter, Rajarshi Das, and Charles Lefurgy. Optimal Power Allocation in Server Farms. In *Proceedings of the Eleventh International Joint Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS '09, pages 157–168, New York, NY, USA, 2009. ACM.

[144] Mohammad A. Haque, Hakan Aydin, and Dakai Zhu. Energy management of standby-sparing systems for fixed-priority real-time workloads. In *Green Computing Conference (IGCC), 2013 International*, pages 1–10, June 2013.

[145] Sadagopan Srinivasan, Li Zhao, Ramesh Illikkal, and Ravishankar Iyer. Efficient Interaction Between OS and Architecture in Heterogeneous Platforms. *SIGOPS Oper. Syst. Rev.*, 45(1):62–72, February 2011.

[146] Kishore Kumar Pusukuri, David Vengerov, and Alexandra Fedorova. A Methodology for Developing Simple and Robust Power Models using Performance Monitoring Events. In *Proceedings of the Workshop Interaction between Operating Systems and Computer Architecture*, WIOSCA 2009, Washington, DC, USA, 2009. IEEE Computer Society.

[147] William Lloyd Bircher and Lizy K. John. Complete System Power Estimation Using Processor Performance Events. *IEEE Trans. Comput.*, 61(4):563–577, April 2012.

[148] Tapas Kanungo, David M. Mount, Nathan S. Netanyahu, Christine D. Piatko, Ruth Silverman, and Angela Y. Wu. An Efficient k-Means Clustering Algorithm: Analysis and Implementation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 24(7):881–892, July 2002.

[149] Dimitrios Chasapis, Marc Casas, Miquel Moretó, Raul Vidal, Eduard Ayguadé, Jesús Labarta, and Mateo Valero. PARSECSs: Evaluating the Impact of Task Parallelism in the PARSEC Benchmark Suite. *ACM Trans. Archit. Code Optim.*, 12(4):41:1–41:22, December 2015.

[150] Hui Kang and Jennifer L. Wong. To Hardware Prefetch or Not to Prefetch?: A Virtualized Environment Study and Core Binding Approach. *SIGPLAN Not.*, 48(4):357–368, March 2013.

[151] Yingxin Wang, Yan Cui, Pin Tao, Haining Fan, Yu Chen, and Yuanchun Shi. Reducing Shared Cache Contention by Scheduling Order Adjustment on Commodity Multi-cores. In *Parallel and Distributed Processing Workshops and Phd Forum (IPDPSW), 2011 IEEE International Symposium on*, pages 984–992, May 2011.

[152] Vinicius Petrucci, Orlando Loques, Daniel Mosse, Rami Melhem, Neven Abou Gazala, and Sameh Gobriel. Thread Assignment Optimization with Real-Time Performance and Memory Bandwidth Guarantees for Energy-Efficient Heterogeneous Multi-core Systems. In *Proceedings of the 2012 IEEE 18th Real Time and Embedded Technology and Applications Symposium*, RTAS '12, pages 263–272, Washington, DC, USA, 2012. IEEE Computer Society.

[153] Linux Kernel. sysfs devices, January 2015. kernel.org/doc/Documentation/ABI/testing/sysfs-devices-system-cpu.

[154] Ethan Zhao. mce_restart() race with CPU hotplug operation, September 2015. lkml.org/lkml/2015/5/1/29.

[155] Rajiv Nishtala, Paul Carpenter, Vinicius Petrucci, and Xavier Martorell. Hipster: Hybrid Task Manager for Latency-Critical Cloud Workloads. In *Proceedings of the 23rd Symposium on High Performance Computer Architecture*, HPCA'16. IEEE Press, 2016.

[156] Intel. Time Stamp Counter, February 2017. en.wikipedia.org/wiki/Time_Stamp_Counter.

[157] Wonyoung Wonyoung Kim, Meeta S. Gupta, Gu-Yeon Wei, and David Brooks. System level analysis of fast, per-core DVFS using on-chip switching regulators.

In *2008 IEEE 14th International Symposium on High Performance Computer Architecture*, pages 123–134. IEEE, 2 2008.

[158] Waclaw Godycki, Christopher Torng, Ivan Bukreyev, Alyssa Apsel, and Christopher Batten. Enabling Realistic Fine-Grain Voltage Scaling with Reconfigurable Power Distribution Networks. In *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO-47, pages 381–393, Washington, DC, USA, 2014. IEEE Computer Society.

[159] Andrew Putnam, Adrian M. Caulfield, Eric S. Chung, Derek Chiou, Kypros Constantinides, John Demme, Hadi Esmaeilzadeh, Jeremy Fowers, Gopi Prashanth Gopal, Jan Gray, Michael Haselman, Scott Hauck, Stephen Heil, Amir Hormati, Joo-Young Kim, Sitaram Lanka, James Larus, Eric Peterson, Simon Pope, Aaron Smith, Jason Thong, Phillip Yi Xiao, and Doug Burger. A reconfigurable fabric for accelerating large-scale datacenter services. In *2014 ACM/IEEE 41st International Symposium on Computer Architecture (ISCA)*, pages 13–24. IEEE, 6 2014.

[160] David Lo and Christos Kozyrakis. Dynamic management of TurboMode in modern multi-core chips. In *2014 IEEE 20th International Symposium on High Performance Computer Architecture (HPCA)*, pages 603–613. IEEE, 2 2014.

[161] Christina Delimitrou and Christos Kozyrakis. Paragon: QoS-aware scheduling for heterogeneous datacenters. *ACM SIGARCH Computer Architecture News*, 41(1):77–77, 3 2013.

[162] Balajee Vamanan, Hamza Bin Sohail, Jahangir Hasan, and T. N. Vijaykumar. TimeTrader: exploiting latency tail to save datacenter energy for online search. In *Proceedings of the 48th International Symposium on Microarchitecture, MICRO 2015, Waikiki, HI, USA, December 5-9, 2015*, pages 585–597, 2015.

[163] Haishan Zhu and Mattan Erez. Dirigent: Enforcing QoS for Latency-Critical Tasks on Shared Multicore Systems. In *Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems - ASPLOS '16*, pages 33–47, New York, New York, USA, 2016. ACM Press.

[164] Arjun Roy, Hongyi Zeng, Jasmeet Bagga, George Porter, and Alex C. Snoeren. Inside the Social Network's (Datacenter) Network. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, SIGCOMM '15, pages 123–137, New York, NY, USA, 2015. ACM.

[165] Nathan Bronson, Zach Amsden, George Cabrera, Prasad Chakka, Peter Dimov, Hui Ding, Jack Ferris, Anthony Giardullo, Sachin Kulkarni, Harry Li, Mark Marchukov, Dmitri Petrov, Lovro Puzar, Yee Jiun Song, and Venkat Venkataramani. TAO: Facebook's Distributed Data Store for the Social Graph. In *Presented*

as part of the 2013 USENIX Annual Technical Conference (USENIX ATC 13), pages 49–60, San Jose, CA, 2013. USENIX.

[166] Christina Delimitrou and Christos Kozyrakis. QoS-Aware Scheduling in Heterogeneous Datacenters with Paragon. *ACM Trans. Comput. Syst.*, 31(4):12:1–12:34, December 2013.

[167] Jialin Li, Naveen Kr. Sharma, Dan R. K. Ports, and Steven D. Gribble. Tales of the Tail. In *Proceedings of the ACM Symposium on Cloud Computing - SOCC '14*, pages 1–14, New York, New York, USA, 2014. ACM Press.

[168] Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., New York, NY, USA, 1st edition, 1994.

[169] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2 2015.

[170] Gerald Tesauro, Nicholas Jong, Rajarshi Das, and Mohamed Bennani. A Hybrid Reinforcement Learning Approach to Autonomic Resource Allocation. In *2006 IEEE International Conference on Autonomic Computing*, pages 65–73. IEEE, 2006.

[171] IBM. Model-Based and Model-Free Approaches to Autonomic Resource Allocation, February 2007. domino.watson.ibm.com/library/cyberdig.nsf/papers/F5E3B7F574B24BAD852570C1005E35A9/$File/rc23802.pdf.

[172] Gerald Tesauro. Online Resource Allocation Using Decompositional Reinforcement Learning. In *Proceedings of the 20th National Conference on Artificial Intelligence - Volume 2*, AAAI'05, pages 886–891. AAAI Press, 2005.

[173] Tibor Horvath, Tarek Abdelzaher, Kevin Skadron, and Xue Liu. Dynamic Voltage Scaling in Multitier Web Servers with End-to-End Delay Control. *IEEE Transactions on Computers*, 56(4):444–458, 4 2007.

[174] Richard Suton and Andrew Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, 1998.

[175] Jacob Leverich, Matteo Monchiero, Vanish Talwar, Parthasarathy Ranganathan, and Christos Kozyrakis. Power Management of Datacenter Workloads Using Per-Core Power Gating. *IEEE Computer Architecture Letters*, 8(2):48–51, 2 2009.

[176] Niti Madan, Alper Buyuktosunoglu, Pradip Bose, and Murali Annavaram. A case for guarded power gating for multi-core processors. In *2011 IEEE 17th*

International Symposium on High Performance Computer Architecture, pages 291–300. IEEE, 2 2011.

[177] Richard Pattis. Complexity of Python Operations, June 2016. ics.uci.edu/{~{}}pattis/ICS-33/lectures/complexitypython.txt".

[178] Sunder Ram Krishnan and Chandra Sekhar Seelamantula. On the Selection of Optimum Savitzky-Golay Filters. *Trans. Sig. Proc.*, 61(2):380–391, January 2013.

[179] Santhosh Kumar Rethinagiri, Oscar Palomar, Rabie Ben Atitallah, Smail Niar, Osman Unsal, and Adrian Cristal Kestelman. System-level Power Estimation Tool for Embedded Processor Based Platforms. In *Proceedings of the 6th Workshop on Rapid Simulation and Performance Evaluation: Methods and Tools*, RAPIDO '14, pages 5:1–5:8, New York, NY, USA, 2014. ACM.

[180] Canturk Isci and Margaret Martonosi. Phase characterization for power: evaluating control-flow-based and event-counter-based techniques. In *The Twelfth International Symposium on High-Performance Computer Architecture, 2006.*, pages 121–132, Feb 2006.

[181] Gaurav Dhiman and Tajana Šimunic Rosing. System-level Power Management Using Online Learning. *Trans. Comp.-Aided Des. Integ. Cir. Sys.*, 28(5):676–689, May 2009.

[182] Tapasya Patki, David K. Lowenthal, Barry Rountree, Martin Schulz, and Bronis R. de Supinski. Exploring Hardware Overprovisioning in Power-constrained, High Performance Computing. In *Proceedings of the 27th International ACM Conference on International Conference on Supercomputing*, ICS '13, pages 173–182, New York, NY, USA, 2013. ACM.

[183] Barry Rountree, Dong Ahn, Bronis R. de Supinski, David K. Lowenthal, and Martin Schulz. Beyond DVFS: A First Look at Performance under a Hardware-Enforced Power Bound. In *Parallel and Distributed Processing Symposium Workshops PhD Forum (IPDPSW), 2012 IEEE 26th International*, pages 947–953, May 2012.

[184] Alexandra Fedorova, Margo Seltzer, and Michael D. Smith. Improving Performance Isolation on Chip Multiprocessors via an Operating System Scheduler. In *Proceedings of the 16th International Conference on Parallel Architecture and Compilation Techniques*, PACT '07, pages 25–38, Washington, DC, USA, 2007. IEEE Computer Society.

[185] Rob Knauerhase, Paul Brett, Barbara Hohlt, Tong Li, and Scott Hahn. Using OS Observations to Improve Performance in Multicore Systems. *IEEE Micro*, 28(3):54–66, May 2008.

[186] David Tam, Reza Azimi, and Michael Stumm. Thread Clustering: Sharing-aware Scheduling on SMP-CMP-SMT Multiprocessors. In *Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007*, EuroSys '07, pages 47–58, New York, NY, USA, 2007. ACM.

[187] David K. Tam, Reza Azimi, Livio B. Soares, and Michael Stumm. RapidMRC: Approximating L2 Miss Rate Curves on Commodity Systems for Online Optimizations. *SIGPLAN Not.*, 44(3):121–132, March 2009.

[188] David Lo, Liqun Cheng, Rama Govindaraju, Parthasarathy Ranganathan, and Christos Kozyrakis. Improving Resource Efficiency at Scale with Heracles. *ACM Trans. Comput. Syst.*, 34(2):6:1–6:33, May 2016.

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

[190] Vasileios Spiliopoulos, Stefanos Kaxiras, and Georgios Keramidas. Green governors: A framework for Continuously Adaptive DVFS. In *Green Computing Conference and Workshops (IGCC), 2011 International*, pages 1–8, July 2011.

[191] Jordi Torres, David Carrera, Kevin Hogan, Ricard Gavalda, Vicenc Beltran, and Nicolas Poggi. Reducing wasted resources to help achieve green data centers. In *2008 IEEE International Symposium on Parallel and Distributed Processing*, pages 1–8, April 2008.

[192] Gaurav Dhiman and Tajana Simunic Rosing. Dynamic Voltage Frequency Scaling for Multi-tasking Systems Using Online Learning. In *Proceedings of the 2007 International Symposium on Low Power Electronics and Design*, ISLPED '07, pages 207–212, New York, NY, USA, 2007. ACM.

[193] Gaurav Dhiman and Tajana Simunic Rosing. Dynamic Power Management Using Machine Learning. In *Proceedings of the 2006 IEEE/ACM International Conference on Computer-aided Design*, ICCAD '06, pages 747–754, New York, NY, USA, 2006. ACM.

[194] Sherief Reda, Ryan Cochran, and Ayse Coskun. Adaptive Power Capping for Servers with Multithreaded Workloads. *Proceedings of the 2012 45th Annual IEEE/ACM International Symposium on Microarchitecture*, 32(5):64–75, Sept 2012.

[195] Rajarshi Das, Mor Harchol-balter, Anshul Gandhi, Jeffrey O. Kephart, and Charles Lefurgy. Power Capping Via Forced Idleness, June 2009. repository.cmu.edu/cgi/viewcontent.cgi?article=1868&context=compsci.

[196] Qingyuan Deng, David Meisner, Abhishek Bhattacharjee, Thomas F. Wenisch, and Ricardo Bianchini. CoScale: Coordinating CPU and Memory System DVFS in Server Systems. In *Proceedings of the 2012 45th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO-45, pages 143–154, Washington, DC, USA, 2012. IEEE Computer Society.

[197] Hiroshi Sasaki, Satoshi Imamura, and Koji Inoue. Coordinated Power-performance Optimization in Manycores. In *Proceedings of the 22Nd International Conference on Parallel Architectures and Compilation Techniques*, PACT '13, pages 51–62, Piscataway, NJ, USA, 2013. IEEE Press.

[198] Bo Su, Joseph L. Greathouse, Junli Gu, Michael Boyer, Li Shen, and Zhiying Wang. Implementing a Leading Loads Performance Predictor on Commodity Processors. In *Proceedings of the 2014 USENIX Conference on USENIX Annual Technical Conference*, USENIX ATC'14, pages 205–210, Berkeley, CA, USA, 2014. USENIX Association.

[199] István Z. Reguly, Abdoul-Kader Keita, and Michael B. Giles. Benchmarking the IBM Power8 Processor. pages 61–69, 2015.

[200] Anip Das, Geoff V. Merrett, and Bashir M. Al-Hashimi. The slowdown or race-to-idle question: Workload-aware energy optimization of SMT multicore platforms under process variation. In *2016 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 535–538, March 2016.

[201] Leo Porter, Michael A. Laurenzano, Ananta Tiwari, Adam Jundt, William A. Ward, Jr., Roy Campbell, and Laura Carrington. Making the Most of SMT in HPC: System- and Application-Level Perspectives. *ACM Trans. Archit. Code Optim.*, 11(4):59:1–59:26, January 2015.

[202] 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 *Proceedings of the 36th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO 36, pages 81–, Washington, DC, USA, 2003. IEEE Computer Society.

[203] Jason Cong and Bo Yuan. Energy-efficient Scheduling on Heterogeneous Multi-core Architectures. In *Proceedings of the 2012 ACM/IEEE International Symposium on Low Power Electronics and Design*, ISLPED '12, pages 345–350, New York, NY, USA, 2012. ACM.

[204] Shruti Padmanabha, Andrew Lukefahr, Reetuparna Das, and Scott Mahlke. Trace Based Phase Prediction for Tightly-coupled Heterogeneous Cores. In *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO-46, pages 445–456, New York, NY, USA, 2013. ACM.

[205] Juan Carlos Saez, Manuel Prieto, Alexandra Fedorova, and Sergey Blagodurov. A Comprehensive Scheduler for Asymmetric Multicore Systems. In *Proceedings of the 5th European Conference on Computer Systems*, EuroSys '10, pages 139–152, New York, NY, USA, 2010. ACM.

[206] David Koufaty, Dheeraj Reddy, and Scott Hahn. Bias Scheduling in Heterogeneous Multi-core Architectures. In *Proceedings of the 5th European Conference on Computer Systems*, EuroSys '10, pages 125–138, New York, NY, USA, 2010. ACM.

[207] Íñigo Goiri, Kien Le, Md. E. Haque, Ryan Beauchea, Thu D. Nguyen, Jordi Guitart, Jordi Torres, and Ricardo Bianchini. GreenSlot: Scheduling Energy Consumption in Green Datacenters. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '11, pages 20:1–20:11, New York, NY, USA, 2011. ACM.

[208] Kien Le, Ricardo Bianchini, Thu D. Nguyen, Ozlem Bilgir, and Margaret Martonosi. Capping the Brown Energy Consumption of Internet Services at Low Cost. In *Proceedings of the International Conference on Green Computing*, GREENCOMP '10, pages 3–14, Washington, DC, USA, 2010. IEEE Computer Society.

[209] Vasileios Kontorinis, Liuyi Eric Zhang, Baris Aksanli, Jack Sampson, Houman Homayoun, Eddie Pettis, Dean M. Tullsen, and Tajana Simunic Rosing. Managing Distributed Ups Energy for Effective Power Capping in Data Centers. *SIGARCH Comput. Archit. News*, 40(3):488–499, June 2012.

[210] Rakesh Kumar, Dean M. Tullsen, and Norman P. Jouppi. Core Architecture Optimization for Heterogeneous Chip Multiprocessors. In *Proceedings of the 15th International Conference on Parallel Architectures and Compilation Techniques*, PACT '06, pages 23–32, New York, NY, USA, 2006. ACM.

[211] Vishakha Gupta, Rob Knauerhase, and Karsten Schwan. Attaining System Performance Points: Revisiting the End-to-end Argument in System Design for Heterogeneous Many-core Systems. *SIGOPS Oper. Syst. Rev.*, 45(1):3–10, February 2011.

[212] Alexandra Fedorova, Juan Carlos Saez, Daniel Shelepov, and Manuel Prieto. Maximizing Power Efficiency with Asymmetric Multicore Systems. *Commun. ACM*, 52(12):48–57, December 2009.

[213] Violaine Villebonnet, Georges Da Costa, Laurent Lefèvre, Jean-Marc Pierson, and Patricia Stolf. Energy Aware Dynamic Provisioning for Heterogeneous Data Centers. In *2016 28th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*, pages 206–213, Oct 2016.

[214] Daniel Wong. Peak Efficiency Aware Scheduling for Highly Energy Proportional Servers. *SIGARCH Comput. Archit. News*, 44(3):481–492, June 2016.

[215] Dejan Novaković, Nedeljko Vasić, Stanko Novaković, Dejan Kostić, and Ricardo Bianchini. DeepDive: Transparently Identifying and Managing Performance Interference in Virtualized Environments. In *Proceedings of the 2013 USENIX Conference on Annual Technical Conference*, USENIX ATC'13, pages 219–230, Berkeley, CA, USA, 2013. USENIX Association.

[216] Xiao Zhang, Eric Tune, Robert Hagmann, Rohit Jnagal, Vrigo Gokhale, and John Wilkes. CPI 2: CPU performance isolation for shared compute clusters. In *Proceedings of the 8th ACM European Conference on Computer Systems - EuroSys '13*, page 379, New York, New York, USA, 4 2013. ACM Press.

[217] Marcus Carvalho, Walfredo Cirne, Franciso Brasileiro, and John Wilkes. Long-term SLOs for reclaimed cloud computing resources. In *ACM Symposium on Cloud Computing (SoCC)*, pages 20:1–20:13, Seattle, WA, USA, 2014.

[218] Christina Delimitrou, Daniel Sanchez, and Christos Kozyrakis. Tarcil: Reconciling Scheduling Speed and Quality in Large Shared Clusters. In *Proceedings of the Sixth ACM Symposium on Cloud Computing*, SoCC '15, pages 97–110, New York, NY, USA, 2015. ACM.

[219] Ionel Gog, Malte Schwarzkopf, Adam Gleave, Robert N. M. Watson, and Steven Hand. Firmament: Fast, Centralized Cluster Scheduling at Scale. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pages 99–115, GA, 2016. USENIX Association.

[220] Qiang Wu. Making Facebook's software infrastructure more energy efficient with Autoscale. research.facebook.com/blog/making-facebook-s-software-infrastructure-more-energy-efficient-with-autoscale/.

[221] Xu Zhou, Haoran Cai, Qiang Cao, Hong Jiang, Lei Tian, and Changsheng Xie. GreenGear: Leveraging and Managing Server Heterogeneity for Improving Energy Efficiency in Green Data Centers. In *Proceedings of the 2016 International Conference on Supercomputing*, ICS '16, pages 12:1–12:14, New York, NY, USA, 2016. ACM.

[222] Engin Ipek, Onur Mutlu, José F. Martínez, and Rich Caruana. Self-Optimizing Memory Controllers: A Reinforcement Learning Approach. In *Proceedings of the 35th Annual International Symposium on Computer Architecture*, ISCA '08, pages 39–50, Washington, DC, USA, 2008. IEEE Computer Society.

BIOGRAPHICAL SKETCH

Rajiv Nishtala was born in Chilkaluripet, India. He received a Bachelor of Science degree from ICFAI University, India in 2011 and a Master of Science degree from FH Heidelberg, Germany in 2013, both in Computer Science. As a masters student in Germany, he gave up a social life to be a programmer at Molcad GmbH, Germany and Deutsches Krebsforschungszentrum (DKFZ), Germany. To pursue research on energy efficient scheduling as a part of his master thesis, he moved to the University of Pittsburgh, USA in 2012 as a visiting scholar to work with Daniel Mossé (Professor).

He began his doctoral studies at Universitat Politècnica de Catalunya (UPC) and Barcelona Supercomputing Center (BSC), Spain where he worked under the graceful supervision of Xavier Martorell (Professor), Paul Carpenter (Senior Researcher) and Daniel Mossé. His current research interests include power and performance modelling, grid computing, energy efficient scheduling in heterogeneous environments and applied machine learning.

Rajiv Nishtala is expected to receive a Doctor of Philosophy degree in Computer Architecture in July 2017.