# Energy Optimising Methodologies On Heterogeneous Data Centres

**Rajiv Nishtala**

Department of Computer Architecture
Universitat Politécnica de Catalunya

This dissertation is submitted for the degree of
*Doctor of Philosophy*

September 2017

# Energy Optimising Methodologies On Heterogeneous Data Centres

by

Rajiv Nishtala

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

Thesis Advisory Committee

. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
Prof. Xavier Martorell Bofill, PhD.
Universitat Politécnica de Catalunya, Spain

. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
Prof. Daniel Mossé, PhD.
University of Pittsburgh, USA

Barcelona, September, 2017

# Energy Optimising Methodologies On Heterogeneous Data Centres

Copyright ©2017
by
Rajiv Nishtala

# Abstract

This is where you write your abstract ...

# Preface

Contributions present in this thesis have been previously presented in the following proceedings:

nishtala: need to fix bibtex for preface

1. **R. Nishtala**, Marc Gonzalez Tallada, Xavier Martorell, “A Methodology Methodology to Build Models and Predict Performance-Power in CMPs", in Proceedings of the 44th International Conference on Parallel Processing Workshops, ICPPW 2015, Beijing, China, September 1-4, 2015, pp. 193-202

2. **R. Nishtala**, 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 2016, Hangzhou, China, November 7-9, 2016, pp.**NEED TO FILL THIS**

3. **R. Nishtala**, Xavier Martorell, Vinicius Petrucci, Daniel Mossé, “REPP-H: Runtime Estimation of Power and Performance on Heterogeneous Data Centers", in Proceedings of the 28th International Symposium on Computer Architecture and High Performance Computing, SBAC-PAD 2016, Los Angeles, USA, October 26-28, 2016, pp.**NEED TO FILL THIS**

The following publications are under review:

1. **R. Nishtala**, Paul Carpenter, Xavier Martorell, Vincius Petrucci, “Hipster: Hybrid Task Manager for Latency-Critical Cloud Workloads", in Proceedings of 23rd IEEE Symposium on High Performance Computer Architecture, HPCA, 2017, Austin, USA, February 4-8, 2017, pp.**NEED TO FILL THIS**

2. **R. Nishtala**, Paul Carpenter, Xavier Martorell, “REPP: A Methodology for Runtime Estimation of Performance–Power in CMPs", in Proceedings of IEEE International Symposium on Performance Analysis of Systems and Software, ISPASS, 2017, San Francisco, USA, April 23-25, 2017, pp.**NEED TO FILL THIS**

The following publications have been published, but not included in this thesis:

1. **R. Nishtala**, Daniel Mossé, Vinicius Petrucci, “Energy-aware thread co-location in heterogeneous multicore processors", in Proceedings of the Eleventh ACM International Conference on Embedded Software, EMSOFT, 2013, Montreal, Canada, September 29 - October 4, 2013, pp.21:1–21:9

# Table of contents

**Preface** **v**

**List of figures** **ix**

**List of tables** **x**

**Nomenclature** **xv**

**1 Introduction** **1**

**2 Background** **2**

**3 Architectures and Benchmarks** **3**

3.1 Architecture . . . . . 3

3.2 Power Meters . . . . . 4

3.3 Workloads . . . . . 5

3.3.1 Batch workloads . . . . . 5

3.3.2 Interactive workloads . . . . . 6

3.4 Performance Monitoring . . . . . 7

**4 REPP: Runtime Estimation of Performance–Power** **10**

4.1 REPP: Runtime Estimation for Performance and Power . . . . . 14

4.1.1 Single core offline modelling . . . . . 14

4.1.2 Evaluation . . . . . 23

4.1.3 Conclusion . . . . . 33

4.2 REPP-H: Runtime Estimation for Performance and Power in Heterogeneous Data Centres . . . . . 34

4.2.1 Multi-core modelling . . . . . 34

4.2.2 REPP-H configuration selector . . . . . 35

4.2.3 Evaluation . . . . . 36

4.3 REPP-C . . . . . . . . . . 37
4.4 Implementation . . . . . . . . . . 37
4.5 Scalability . . . . . . . . . . 37

**5 Validating Methodology** **38**

**6 Hipster** **39**

**7 Related Work** **40**

**8 Conclusion** **41**

**References** **42**

# List of figures

4.1 Component activity ratios over 100 million instructions at all P–States for *lbm* from the SPEC CPU 2006 benchmark suite. . . . . . . . . . . . . . . . . 19
4.2 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. . . . . . . . . . . . . . . . . . . . . . . . . . . . . 22
4.3 Percentage error in performance, and power prediction for SPEC benchmarks Astar (mid memory intensive), Calculix (compute intensive), and Mcf (memory intensive). . . . . . . . . . . . . . . . . . . . . . . . . . . . . 24
4.4 Percentage error per grid in performance, and power prediction for SPEC benchmarks Astar (mid memory intensive), Calculix (compute intensive), and Mcf (memory intensive). . . . . . . . . . . . . . . . . . . . . . . . . 25
4.5 Average PAAE when predicting performance and power on a single core for across a combination of P-States, Cl-States and architectures. The error bars represent the STDEV . . . . . . . . . . . . . . . . . . . . . . . . . . . 30
4.6 Average PAAE when predicting performance and power per suite across architectures. The error bars represent the STDEV. . . . . . . . . . . . . . 31
4.7 Average PAAE when predicting performance and power on a single core for each benchmark from four benchmark suites across a combination of P-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, SPLASH2x . . . . . . . . . . . . . . . . . . . . . . . . . . . . 32
4.8 High-level view of REPP runtime system. . . . . . . . . . . . . . . . . . . 36

# List of tables

3.1 Workload configurations, maximum load while meeting the target tail latency with two big cores for latency-critical applications. . . . . . . . . . . . . . 7
3.2 Machines used in this study. . . . . . . . . . . . . . . . . . . . . . . . . . 9
3.3 Categorization of workloads. . . . . . . . . . . . . . . . . . . . . . . . . . 9

4.1 Summary and description of the symbolic notation . . . . . . . . . . . . . . 15
4.2 Component definitions for Intel Core i7 . . . . . . . . . . . . . . . . . . . 16
4.3 Events and *perf* raw event numbers used on AMD (top), ARM (middle) and Intel (bottom) machines . . . . . . . . . . . . . . . . . . . . . . . . . . . . 17
4.4 Range of activity ratios over 100 million instructions at minimum and maximum P–States for SPECcpu2006. . . . . . . . . . . . . . . . . . . . . . . . 19
4.5 PAAE, and STDEV for power when switching between P-States. . . . . . . 26
4.6 PAAE, and STDEV for performance when switching between P-States. . . 27
4.7 PAAE, and STDEV for power when switching between Cl-States at 1.8GHz. 28
4.8 PAAE and STDEV for MIPS when switching between Cl-States at 1.8GHz. 29
4.9 PAAE when predicting performance for every combination of switch in P-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. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 33

# Todo list

need to fix bibtex for preface . . . . . . . . . . . . . . . . . . . . . . . . . . . . v
need to speak about how latency-critical, and batch workloads are used . . . . . . . . 1
need for power caping? . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 1
need glossary for power efficient . . . . . . . . . . . . . . . . . . . . . . . . . . 3
mention somewhere it is real hardware . . . . . . . . . . . . . . . . . . . . . . . . 3
will Hipster and repp be explained by now? . . . . . . . . . . . . . . . . . . . . . 3
show in appendix.. which workloads are from which suites . . . . . . . . . . . . . . 5
need not be called contributions.. if REPP and HIPSTER are defined before . . . . . 6
diurnal load figure number . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 6
Hipster is mentioned here so maybe think about how you will satisfy this scenario . . 7
PLOT GOOGLE FIGURE, and EXPLAIN . . . . . . . . . . . . . . . . . . . . . . . . . 10
check if this is what you want to write?... still flow is missing . . . . . . . . . . . . 12
should i separate the results? fix arm results.. maybe use overpic?! . . . . . . . . . . 32
lots of citations are missing . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 34
rewrite this part on . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 35
explain about the architecture here . . . . . . . . . . . . . . . . . . . . . . . . . 35
REPP ARCHITECTURE diagram . . . . . . . . . . . . . . . . . . . . . . . . . . . . 37

*Dedicated in loving memory of my grandfather, Rajeswara Rao Nishtala (PhD).*

# Declaration

I hereby declare that except where specific reference is made to the work of others, the contents of this dissertation are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other university. This dissertation is my own work and contains nothing which is the outcome of work done in collaboration with others, except as specified in the text and Acknowledgements. This dissertation contains fewer than 65,000 words including appendices, bibliography, footnotes, tables and equations and has fewer than 150 figures.

Rajiv Nishtala
September 2017

# Acknowledgements

I would like to dedicate this thesis to my struggle

# Nomenclature

## Roman Symbols

| | |
|---|---|
| $F$ | Cache Friendly batch workload |
| $N$ | Insensitive batch workload |
| $S$ | Thrashing batch workload |
| $T$ | Cache Fitting batch workload |

## Greek Symbols

| | |
|---|---|
| $\alpha_c$ | Current configuration, that is, $(P_c, Cl_c)$ |
| $\alpha_f$ | Future configuration, that is, $(P_f, Cl_f)$ |
| $Cl_c$ | Current Cl-State |
| $Cl_f$ | Future Cl-State |
| $Cl_i$ | Intermediate Cl-State |
| $\eta_c$ | Performance (MIPS) at current configuration |
| $\eta_f$ | Performance (MIPS) at future configuration |
| $P_c$ | Current frequency for P-State |
| $P_f$ | Future frequency for P-State |
| $P_i$ | Intermediate frequency for P-State |
| $\rho_c$ | Power at current configuration |
| $\rho_f$ | Power at future configuration |

## Acronyms / Abbreviations

| | |
|---|---|
| $AR$ | Activity Ratio |
| $Big$ | ARM Cortex-A57 |
| $BPU$ | Branch Predictor Unit |
| $CFS$ | Completely Fair Scheduler |
| $DPM$ | Dynamic Power Management |
| $DVFS/P-States$ | Dynamic Voltage, and Frequency Scaling |
| $FE$ | Front End |
| $FP$ | Floating Point Unit |
| $INT$ | Integer Unit |
| $IPS$ | Instructions Per Second |
| $L1$ | L1 Cache Memory |
| $L2$ | L2 Cache Memory |
| $LLC$ | Last Cache Memory |
| $LLC$ | Last Level Cache |
| $MSR$ | 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 |
| $REPP$ | Runtime Estimation of Performance and Power |
| $REPP-C$ | Runtime Estimation of Performance and Power with workload consolidation |

| | |
|---|---|
| $REPP-H$ | Runtime Estimation of Performance and Power across Heterogeneous architecture |
| $RPS$ | Requests per Second |
| $Small$ | ARM Cortex-A53 |
| $STDEV$ | Standard Deviation |

# CHAPTER 1

# Introduction

test[1]

Importance of low end processors, and the extension of vector support for ARM processors (!) [1]

nishtala: need to speak about how latency-critical, and batch workloads are used

nishtala: need for power caping?

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

# CHAPTER 2

# Background

# CHAPTER 3

# Architectures and Benchmarks

nishtala: need glossary for power efficient

nishtala: mention somewhere it is real hardware

THIS chapter gives a gist of the architectures, the power meters used, the type of applications, and their characterisation. In recent years, the multicore architectures deployed in server systems are increasingly visible in both mobile processors (for example: the ARM Juno, and AppliedMicro XGene), and supercomputers such as `MareNostrum` that they have become a norm for modern 64 bit multicore systems. Such non-restrictive deployment of multicore architectures in large-scale data centres led to a great divide between power and performance, which are tightly bound. For instance, the `MontBlanc` project [2–4] from the European union deployed multiple mobile cores to build a supercomputer which is "power efficient" (performance per watt). This great divide makes it increasingly difficult to find a balance between the two. On the one hand, delivering higher performance implies consuming more power; On the other hand, delivering lower performance might imply missing a deadline for a latency-critical workload (such as a bank transaction or web-request which tend to terminate within milli/micro seconds [5, 6]), while saving some power. In the recent years, several new processors [7–12] have been proposed in search of the optimal point in the "pareto curve", thereby making an algorithm generalised if it works across multiple architectures and processors.

## 3.1 Architecture

nishtala: will Hipster and repp be explained by now?

In this section, we show the architectures and processors our contributions have been evaluated on, unless otherwise stated.

**Intel** Intel Corei7 Sandy Bridge processor [13] 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 Dynamic Voltage and Frequency Scaling (DVFS/P-States) from 0.8 GHz to 2.4 GHz. Turbo boost was disabled. The total number of Cl-States in this study are 50 per core. We do not enable any other internal thermal/power management algorithms which are run by the firmware.

**AMD** AMD Phenom II [14] 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. We do not enable any other internal thermal/power management algorithms which are run by the firmware.

**ARM** ARM Juno R1 developer board [15] 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. 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.

**X-Gene2** AppliedMicro X-Gene2 [16] 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.

## 3.2 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 neither with a power sensor nor attached to a power meter.

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 [17]. The RAPL interface is accessed using the Machine Specific Registers (MSR) registers available on Intel processors [18]. (2) *WattsUP pro* for AMD: external device that records power of the entire system (power from outlet) [19]. (3) *Four native energy meters* for ARM [20, 21]. These registers report separately the power consumed by the big cluster, small cluster, and the rest of the system (Juno's *sys* register [22]). The power consumption of the Mali GPU is also available, but it is negligible because the GPU is disabled in all our experiments.

Table 3.2 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 had no control over the kernel and gcc versions; regardless, the results for each architecture are consistent within that set of experiments. In what follows, I refer to Intel Sandybridge as Intel, AMD Phenom II as AMD, and ARM Juno R1 as ARM.

## 3.3 Workloads

Our contributions have been evaluated with two classes of workloads: batch workloads or throughput-oriented workloads [23, 24], and interactive or latency-critical workloads [25–30]. Both classes are extensively used in data centres and mobile architectures **CITE**. Therefore, in evaluating our contributions we have explored a wide variety of both workloads. On the one hand, batch workloads facilitate for a large spectrum of activity in different microarchitectural components; On the other hand, latency-critical workloads require us to deliver the a user request within a fraction of a second, which otherwise might compromise on the user database.

nishtala: show in appendix.. which workloads are from which suites

### 3.3.1 Batch workloads

We categorize the batch workloads into single threaded batch workloads, and multiprogrammed batch workloads. For single-threaded batch workloads, we ran 22 SPEC CPU 2006 [31, 32], nine PARSEC 3.0 [33], six NAS [34], 11 SPLASH2x [35]. 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, based on the methodology described by Sanchez and Kozyrakis [36].

The benchmarks are divided into four categories, following the categorization by Sanchez and Kozyrakis [36]. We run all the applications in isolation and gather performance monitoring counters (PMCs) at a sampling frequency of 250 ms until the end, and compute mean value of the ratio of number of kilo-LLC misses (LLC: Last Level Cache) for kilo-instructions-per-second (kilo-IPS). For all workloads, we select the native input set size. Those which have ratio less 0.1 are considered as Thrashing (**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 3.3 shows the categorization 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.

### 3.3.2 Interactive workloads

nishtala: need not be called contributions.. if REPP and HIPSTER are defined before

We evaluate the effectiveness of the contributions using two interactive workloads, Memcached [37], and Web-Search [38], which have distinct characteristics and impact on shared resources [26]. **Memcached** [37] is an open source implementation of an in-memory key-value store for data caching used in many services from Twitter, Facebook and Google [5, 6]. The backend of **Web-Search** [39, 40] is an instance of Elasticsearch [38], an open source implementation of a search engine used by many companies including Netflix and Facebook. The load generator (Faban [41]) for Memcached and Web-Search is adapted from CloudSuite 3.0 [42]. It is configured to model diurnal load changes (Figure **FIG NUMBER**)

| 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[41] | 500 (90%ile) |

Table 3.1 Workload configurations, maximum load while meeting the target tail latency with two big cores for latency-critical applications.

nishtala: diurnal load figure number

, simulating a period of 36 hours [30]; each hour in the original workload corresponds to one minute in our experiments.

We also evaluate the effectiveness by collocating a single latency-critical 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 utilized by the latency-critical workload. We report the system throughput by aggregating the IPS of all batch programs.

nishtala: Hipster is mentioned here so maybe think about how you will satisfy this scenario

When implementing Hipster, the ARM processor is used either when colocating interactive workload with batch workload, or running an interactive workload in isolation, while the AppliedMicro XGene2 machine is allocated exclusively for the workload generator.

**Tail latency requirements —** For Memcached, we define the tail latency to be the 95th percentile request latency, with a target of 10 ms [25, 42]; for Web-Search, we define it to be the 90th percentile query latency, with a target of 500 ms [42]. Table 3.1 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.

## 3.4 Performance Monitoring

We use performance monitoring tool, `perf` [43], on all architectures to gather `perf_events`. Alternatives to perf include the profiling tools [44] supported by Docker, Kubernetes and LXC [45].

The ARM Juno board raises two challenges while gathering PMCs: ① It does not allow 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, thereby limiting the scenarios it can be used. ② There is a known bug[**CITE**] that causes perf to generate garbage values for all cores whenever any core enters an idle state. Since performance statistics are only needed when batch workloads are run, we overcome this by disabling CPUidle [46, 47]. This prevents Linux from entering the cores in an idle state when changes in the mapping cause idle periods longer than 3500 µs.

Since `CPTR_EL3` is not implemented on Cortex-A53, it is used only to gather basic performance statistics when batch workloads are colocated with latency-critical workloads.

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.

Table 3.2 Machines used in this study.

| 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 (ARMv8 – 2 Cortex A57 and 4 Cortex A53) | 4.3.0 | Native energy meter | 4.9.2 | 0.6-1.15 (Cortex A57) 0.65 (Cortex A53) | 2048 (Cortex A57) 1024 (Cortex A53) | no L3 |
| Applied X-Gene2 64 bit (8 cores) | 4.10 | no meter | 5.2.0 | 2.4 | 256 | 8 |

Table 3.3 Categorization of workloads.

| Label | Description | Intel-Benchmarks | AMD-Benchmarks | ARM-Benchmarks |
|---|---|---|---|---|
| N | Insensitive | bzip2, blackscholes, bodytrack, calculix, dedup, ep.C, freqmine, gobmk, gromacs, h264ref, hmmer, is.C, lu_cb, namd, omnetpp, povray, raytrace, tonto, vips | blackscholes, bzip2, calculix, ep.C, facesim, freqmine, gobmk, gromacs, h264ref, hmmer, is.C, lu_cb, namd, omnetpp, povray, radiosity, raytrace, raytrace, sjeng, tonto, volrend water_nsquared, water_spatial | blackscholes, ep.C, fft fluidanimate, freqmine is.C, lu_cb,lu_ncb |
| F | Cache Friendly | cactusADM, cg.C, lbm, libquantum lu_ncb, ocean_cp, sjeng water_spatial, x264, zeusmp | bodytrack, cactusADM, canneal, cg.C fmm, lu_ncb, ocean_cp, streamcluster vips, wrf, x264, zeusmp | facesim, radiosity streamcluster,volrend water_nsquared |
| T | Cache Fitting | astar, bwaves, canneal, gemsFDTD radix, soplex | astar, bwaves, dedup, radix soplex | canneal, radix, raytrace, vips water_spatial |
| S | Thrashing | lu.C, mcf, milc, mg.C, sp.C streamcluster, xalancbmk | gemsFDTD, lbm, libquantum, lu.C mcf, mg.C, milc, sp.C, xalancbmk | bodytrack, cg.C, dedup, fmm lu.C, mg.C, ocrean_cp, sp.C |

CHAPTER 4

# REPP: Runtime Estimation of Performance–Power

So far we have understood the types of architecture, and the cla f workloads used in this dissertation: those that make intensive use of different microarchitectural components on the modern data centre infrastructures, aimed at workload throughput, a those which are aimed to be interactive, and require a deadline (or target) is met within a certain time frame. Th also exists mult other types of applications that require immediate access to resources. In this chapter describe our work on power and performance management while considering contention for CPU, as well as shared resources of the machine for throughput-oriented workloads.

Our prior work[1] [48], ch considers workload characterisation using a small number of PMCs to schedule/map the applications to cores such that we optimise for energy efficiency (throughput per watt of energy consumed). In such scenarios applications are scheduled by either sharing a core with another application from a different class to improve energy efficiency (i.e., time sharing a core), or by running in isolation (i.e., no time sharing). Time sharing a core allows to consolidate workloads on a same server, and save energy. This arises from the fact most data centre servers go underutilised for long durations of the day [49, 30, 25, 29, 26].

nishtala: PLOT GOOGLE FIGURE, and EXPLAIN

However, with the increasing demand for low cost, and low power high performance computing, multiple vendors such as Amazon EC2, and IBM Soft Layer, etc., provide users the capability to "rent" computing nodes to process their computational needs without the need for purchasing the equipment. provide such low cost, and low power computing services, modern data centre architectures [50], incidentally or intentionally, have to deal with server architecture heterogeneity [51, 52]. Nevertheless, some critical challenges for

[1]Published as a part of my master thesis (and published in the proceedings of EMSOFT 2013), and not included in this thesis

such a heterogeneous data centre administration are ① map applications across multiple architectures; ② provide a service that is reliable; ③ available on demand for a given computing service; ④ availability at an attractive cost for end-users.

Given the aforementioned challenges in heterogeneous data centres, such a service that can be delivered using native hardware capabilities 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 enable energy-efficient cluster management in data centres (which thread is assigned to each core and what frequency to set each core when). For instance, current data centre tools such as the Intel Intelligent Power Node Manager [53], the OpenStack framework [54] or the IBM Tivoli framework [55], enable power-thermal control and optimisation like monitoring and capping capabilities. 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 DVFS, C-State or other low-level power state would translate into real thread performance and power demand, and satisfy these constraints.

Traditionally such services have been met using iterative algorithms which are P-State based: they monitor the application behaviour history periodically [56] and set the DVFS configuration for the next quantum. But these approaches require multiple iterations before power and performance criteria are satisfied and can lead to massive violations in power and performance budgets for data centres [51, 57].

In response to traditional algorithms, fast and reactive prediction based algorithms provide the benefit of quick response and reaction for a small fraction of computation costs. The broad spectrum of computation and memory behaviour can be understood by monitoring the PMCs, available on most commercial processors. For instance, applications that are memory bounded, do not take benefit of higher P-State as they stall for memory, thus using a lower P-State might be sufficient to satisfy the performance constraint while saving power. At the same time, the converse is true for compute bounded workloads. Using PMCs, we demonstrate that our prediction algorithm can determine if the application is scalable with DVFS at runtime and can select the most appropriate configuration given a constraint. Such techniques are also important to enable energy-efficient cluster management in data centres (which thread is assigned to each core and what frequency to set each core when).

With the advent of multiple dynamic power management [58] (DPM) features in Linux kernel which led to multiple low-level hardware capabilities on modern processors, thereby making it very difficult to understand the combinatorial complexity they bring in terms of service comprehension even for reactive prediction based algorithms. Moreover, administrators of large-scale data centres have to deal with metrics such as performance, power and efficiency levels (e.g., IPS, IPS/W, or Power Usage Effectiveness (PUE)), which do not directly correspond to DVFS, Cl-States or core shutdown. For instance, on the Intel machine with four cores there are nine different DVFS states that can be combined with 50 clamping configurations (Cl-States) per core. This gives the administrator a total of 40 billion configurations, each one resulting in a particular performance-power level.

This complexity in hardware explains why current solutions operate at a coarse granularity, at node level, while the hardware features work at a core level, with low overheads. Understanding the behaviour at per-core or thread level allows for a precise control over performance power contribution and their corresponding effects.

nishtala: check if this is what you want to write?... still flow is missing

There is a need for application agnostic prediction approach in heterogeneous data centres to meet a criteria, irrespective of how it is formulated. This is precisely what our methodology provides.

In this Chapter, firstly we introduce *Runtime Estimation of Performance and Power*, **REPP** [59], a performance and power model for single-core Intel processors parametrised by P-States and Cl-States. REPP is a methodology to generate fast, and accurate performance-power predictions. 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. As a result, irrespective of the formulation of the problem, the models built can used to optimise a given parameter.

Secondly, we introduce an extension for REPP to scale power and performance prediction techniques from single-threaded workload to a variety of multi-programmed workload across server architectures. We specifically designed a *Runtime Estimation of Performance and Power across Heterogeneous architectures*, **REPP-H** [60], to estimate and *control* power-performance on server architectures running multiple workloads using statistics gathered on real processors.

Finally, we also extend REPP to scale power and performance prediction technique with workload consolidation. Specifically, we designed a *Runtime Estimation of Performance and Power with workload consolidation*, **REPP-C** [61], to estimate and control power and performance on server architecture (with consolidation) running multiple workloads.

The evaluation of REPP-H, and REPP-C, on each architecture, was carried out for single-threaded and multiprogrammed applications with multiple constraints, and we show that power capping mechanism and ensuring minimum performance is delivered are met.

Recapping the discussion, we introduce REPP, a methodology to build single core model for predicting power and performance (Section **ONE**) across P-State, and Cl-States. Next, these models are extended to work in a multi-core environment (Section **TWO**), and meeting power and performance constraints across multiple single-threaded, and multi-programmed workloads. Finally, these models are extended to work with workload consolidation which is increasingly common across modern data centres.

## 4.1 REPP: Runtime Estimation for Performance and Power

We propose Runtime Estimation for Performance and Power (*REPP*) in order to model and estimate performance and power on three major architectures: Intel, ARM or AMD. *REPP* is modelled using multi linear regression models, constructed in two steps: *offline* modelling and *online* modelling.

**Offline modeling:** In the offline modelling technique, a subset of applications from representative benchmarks (Section 4.1.1.1) suites are profiled to build to power (Section 4.1.1.2), and performance (Section 4.1.1.3) models offline for a single-core. The models built can predict power, and performance at the available P-State, and Cl-State with high accuracy.

**Online modeling:** In the online modelling technique, we extend our models to the multi-core environment, by *aggregating* the per core contributions. The models exposed to the multi-core case, are modelled to tackle the contention due shared resources. Finally, we describe a technique for power, and performance control across multi-core environment.

The offline design phase is attractive because ⓐ it eliminates the overhead of learning and tuning the performance and power model at numerous P-States and Cl-States; ⓑ it does not rely on using power sensors/meters to estimate power and performance at runtime; and ⓒ it is a one-time effort.

In the remainder of the section, we focus on the the offline modelling procedure, and the offline, and online validation of the single-core models. The notations used hence-forth for the rest of the *chapter* are detailed in Table 4.1.

### 4.1.1 Single core offline modelling

We predict performance and power in a different P-State and Cl-State, based on the activity recorded 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**. The activity ratio is defined as the component's average number of µops per cycle (µops/cycle).

Table 4.2 summarises for Intel [13] the microarchitectural components, and modeled components. Similar components were also modeled on AMD, and ARM. Table 4.3 summarises the microarchitectural components, activity formulas for AMD, ARM and Intel, and

Table 4.1 Summary and description of the symbolic notation

| Notation | Description |
|---|---|
| $P_c$ | Current P-State. |
| $P_i$ | Intermediate P-State. |
| $P_f$ | Future P-State. |
| $P_c$ | Current frequency for P-State. |
| $P_i$ | Intermediate frequency for P-State. |
| $P_f$ | Future frequency for P-State. |
| $Cl_c$ | Current Cl-State. |
| $Cl_i$ | Intermediate Cl-State. |
| $Cl_f$ | Future Cl-State. |
| $\alpha_c$ | Current configuration, that is, $(P_c, Cl_c)$. |
| $\alpha_f$ | Future configuration, that is, $(P_f, Cl_f)$. |
| $\eta_c$ | Performance (MIPS) at current configuration. |
| $\eta_f$ | Performance (MIPS) at future configuration. |
| $\rho_c$ | Power at current configuration. |
| $\rho_f$ | Power at future configuration. |

the raw perf event registers [14, 15, 13]. To compute the activity ratio, the activity in each component, for each architecture, is divided by `cpu_clk_unhalted (r076)`, `cpu_cycles (cycles)` and `cpu_clk_unhalted:0x01 (r13c)`, respectively.

In building the multilinear 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 between 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 total number of integer instructions.

Table 4.2 Component definitions for Intel Core i7

| Component | Modeled 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) |

#### 4.1.1.1 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). These benchmarks are executed to gather the PMCs values during a period of 100 seconds with a sampling period of 250 ms, at all P-States and a subset of the Cl-States. The models built are exclusive to a given architecture, and are rebuilt for each architecture REPP is implemented on.

For this study, the training data set consists of three floating point, and three integer benchmarks, which are chosen at random, and run at all P-States, and five Cl-States. 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. The Cl-States are chosen such that there is at least 10% variation (specifically the Cl-States 10, 20, 30, 40, 50). To validate the resulting single-core models at runtime/online, we use the remainder of the benchmarks from SPEC CPU 2006, PARSEC3.0, NAS, and PARSEC3.0. The *offline validation* of the models was carried extensively on the Intel processor for a proof-of-concept using the SPEC CPU 2006 benchmark suite(Section 4.1.2). Nevertheless, the models built have shown very high accuracy in predicting performance, and power over three different architectures (as will be shown in Section 4.2).

We have also conducted a study with varying number of training benchmarks with all possible combinations, i.e. with two floating point, and two integer, one floating point, three integer, etc., and the prediction error was not acceptable [62–64].

To build models offline, we collect traces which contain the PMCs samples of individual components per application. The traces, obtained at different P-States, need to be realigned in order to compare the activity ratios at similar points of execution. (Section 4.1.1.5). This realigning is done with respect to the total instruction and exclude the trace points for which

Table 4.3 Events and *perf* raw event numbers used on AMD (top), ARM (middle) and Intel (bottom) machines

| Component | AMD | AMD (*perf* raw event) |
|---|---|---|
| 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 |

| Component | ARM | ARM (*perf* raw event) |
|---|---|---|
| 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 |

| Component | Intel | Intel (*perf* raw event) |
|---|---|---|
| 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 |

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 a the activity ratio is comparable.

We empirically select a sampling period of 250 ms to collect PMCs and power using the *RAPL* register for building offline models as it increases the total number of trace points for a given application without exceeding ten million instructions.

#### 4.1.1.2 Modelling Power

Power can be modelled based on the PMCs [62–64]. Multilinear regression models allow for power prediction with high accuracy (less than 5% error rate). To the best of our knowledge, all the models previously proposed only predict the power consumption in the current hardware configuration.

**Predicting power in the current configuration —** We build on the technique proposed by Bertran et al. [65] to predict power at the current configuration ($\rho_c$), and is computed based on the activity recorded in FP, FE, BPU, L1 cache, L2 cache, LLC cache and MEM.

$$\rho_c = \sum_{i=0}^{comps} (\Delta_i \times AR_i) + constant \tag{4.1}$$

Equation 4.1 represents the multi linear regression model for predicting $\rho_c$, where $\Delta_i$ 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 P-State with Cl-State zero.

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

For example, Table 4.4 shows the average activity ratio at the minimum and maximum P-State for the microarchitectural components INT, LLC, and FP for 16 SPEC benchmarks [31] for the first 100 million instructions of execution. Specifically, *Lbm* (memory intensive floating point benchmark) shows a 67.4% change in activity ratio from 0.8 GHz to 2.4 GHz in FP. However, observe that the absolute change in activity ratio is negligible.

We visually represent an example from Table 4.4 in Figure 4.1. The Figure shows the average activity ratio of the microarchitectural components for the first 100 million instructions of execution across all P-States. Specifically, Figure 4.1a shows a 65% change in activity ratio from 0.8 GHz to 2.4 GHz in FP. However, observe that the absolute change in activity ratio 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%).

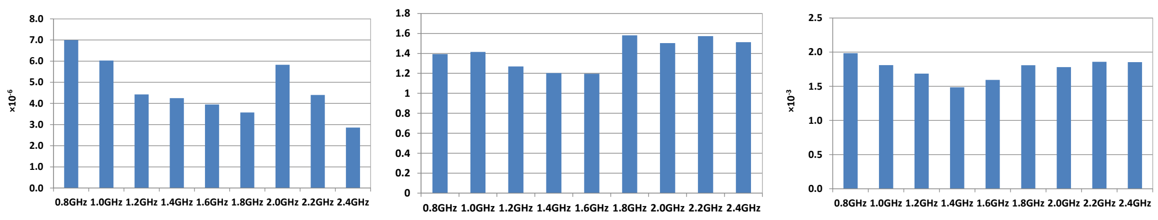
Table 4.4 Range of activity ratios over 100 million instructions at minimum and maximum P–States for SPECcpu2006.

| 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 |

This behaviour is consistently observed across any benchmark for any microarchitectural component, as it is an microarchitectural property; An analogy for this can be drawn to a unique finger print for every person.

On the other hand, having negligible absolute difference in activity ratio across P-States does not imply that the power across P-States for a given Cl-State is constant, but instead the difference in the real activity ratio across P-State is reflected using the coefficients derived in Equation 4.1.



(a) Floating Point Unit (b) Integer Unit (c) LLC Cache

Figure 4.1 Component activity ratios over 100 million instructions at all P–States for *lbm* from the SPEC CPU 2006 benchmark suite.

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

**Predicting power across Cl-States —** To predict power ($\rho_f$) across Cl-States, moving from $Cl_c$ to $Cl_f$, we build a multilinear regression model based on the ratio of the Cl-States and predicted power at the current P-State ($\rho_c$).

$$\rho_f = (\Delta \times \rho_c + (\beta \times \frac{Cl_f}{Cl_c})) + constant \tag{4.2}$$

Equation 4.2 represents the multi linear regression model for predicting $\rho_f$, where $\Delta$, $\beta$ and *constant* represent the multi linear regression coefficients and we build one model at each P-State.

#### 4.1.1.3 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 very difficult to predict. Therefore, the most accurate way to estimate performance is by implementing a full-scale simulator [66, 67]. 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 (`INST_RETIRED`). We take advantage of this counter and report the number of instructions retired (in millions) per second (MIPS), that is, $\eta_c$.

**Predicting performance across P-States —** We predict the performance in MIPS ($\eta_f$) of a thread using multi linear regression models. The components modelled are MIPS, instructions per cycle (IPC), private L1, private L2 and LLC cache and the ratio of change of P-State moving from $P_c$ to $P_f$.

$$\eta_f = \sum_{i=0}^{comps} (\Delta_i \times AR_i) + (\beta \times \frac{P_f}{P_c}) + constant \tag{4.3}$$

Equation 4.3 represents the multi linear regression model for predicting $\eta_f$, where $\Delta_i$, $\beta$ 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 P-State with no idle cycles.

**Predicting performance across Cl-States —** Similarly, for predicting performance ($\eta_f$) across Cl-States, moving from $Cl_c$ to $Cl_f$, we build multi linear regression models while keeping the P-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.

$$\eta_f = \sum_{i=0}^{comps} (\Delta_i \times AR_i) + (\beta \times \frac{Cl_f}{Cl_c}) + constant \tag{4.4}$$

Equation 4.4 represents the multilinear regression model for predicting $\eta_f$, where $\Delta_i$, $\beta$ and *constant* represents the coefficients to be modelled and $AR_i$ represents the activity ratio of the individual components. We build one model for every P-State available.

#### 4.1.1.4 Single-core algorithm

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

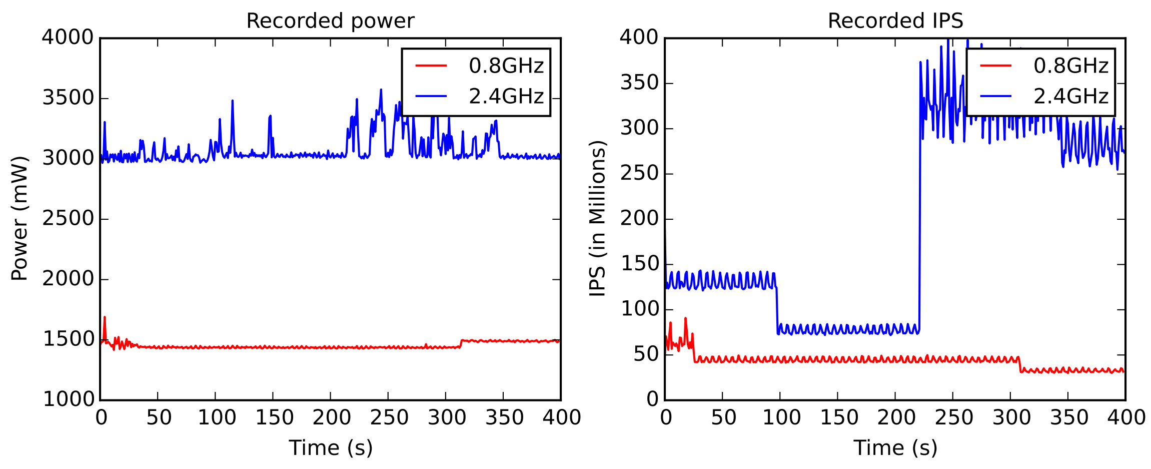
① Read the PMCs at current configuration and compute the activity ratios.
② Predict power and report performance at current configuration.
③ Predict power and performance for current Cl-State and future P-State.
④ Predict power and performance for current P-state and future Cl-State.

#### 4.1.1.5 Trace files

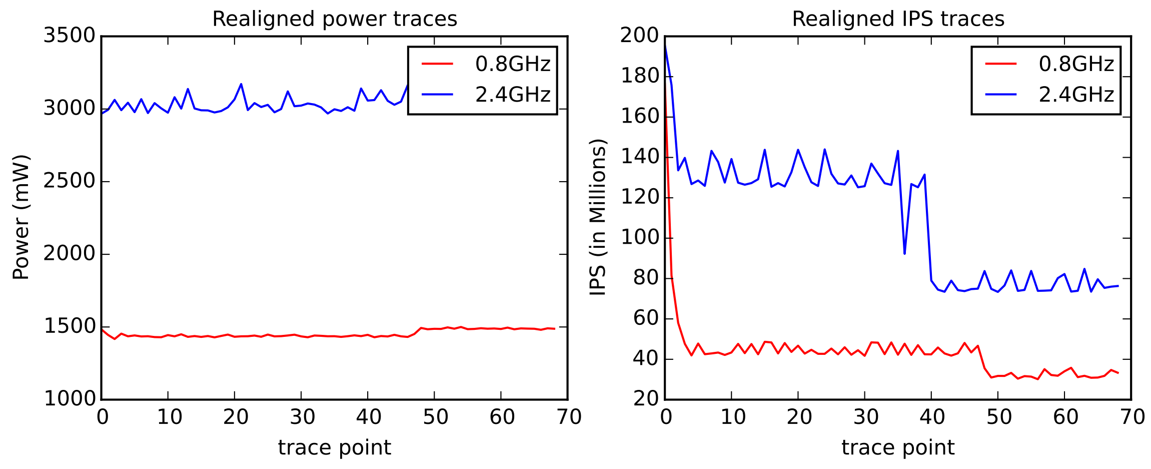
The offline models are built using the statistical information gathered, and stored in trace-files. The trace-files contain activity ratio for individual components for each benchmark. The traces, obtained at different P-States, Cl-States need to be realigned such that the activity ratios are comparable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

The performance statistics for each benchmark in the training dataset are gathered by binding the benchmark to the cores using the Linux syst call `sched_setaffinity`. Assigning the affinity [68] for a thread bypasses the completely fair scheduler (CFS) and helps avoid excessive thread migration and provides a more controlled environment.

Realigining traces gathered at P-State, Cl-State to build models offline provides a unique challenge because the correlation between MIPS, the frequency of execution (P-State), and the time are non-linear.



(a) Power and MIPS with variation in time.



(b) Power and MIPS traces realigned with IPS.

Figure 4.2 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.

For instance, Figure 4.2a shows the time varying demands for performance and power consumed (in mW) for the SPEC benchmark *astar* executing at 0.8 GHz and at 2.4 GHz with Cl-State zero. As can be seen, the totally number of retired instructions at 0.8 GHz and 2.4 GHz are not equal. Observe that the total performance for the first 300 seconds at 0.8 GHz and first 100 seconds at 2.4 GHz is the approximately same. This makes it clear that the correlation between frequency, and total performance is not one-to-one. Therefore, a prediction model based on time can not provide a measure of performance or power at similar point in the execution of the thread and thus, raises the need for realigning traces with respect to total performance.

As indicated before, we use a buffer of ten millions instructions to compare traces for each benchmark, to ensure that the boundary of error is fixed (!). Figure 4.2b 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 execution in the thread. This methodology is followed for every P-State, and Cl-State combination, and such traces here-forth are referred to as *realigned traces*.

### 4.1.2 Evaluation

In this section, we evaluate the models to predict both performance, and power using the SPEC CPU 2006 benchmark suite for the traces gathered. Next, we evaluate the single-core models when predicting performance, and power at runtime/online. Both the evaluations are carried out for numerous P-State and Cl-State combinations.

#### 4.1.2.1 Model Assessment

The evaluation of the prediction models in the remainder of this chapter are evaluated, and shown in terms of percentage absolute average error (PAAE), and the standard deviation (STDEV) for each data point over a period of 300 seconds:

$$PAAE = \frac{1}{N}\left(\sum_{i=1}^{N} \frac{|M_i - m_i|}{m_i}\right) \tag{4.5}$$

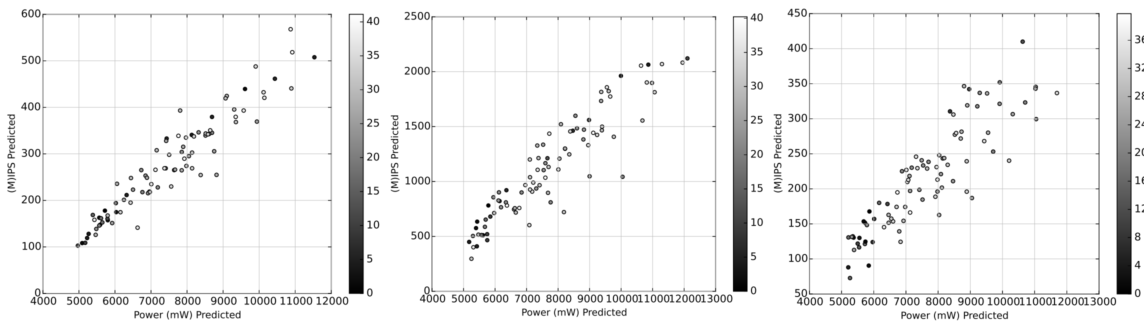
Where N is the number of data points for each benchmark, M is the predicted value, and m is the measured value.

The PAAE is a well-known metric [65, 69–71] to evaluate the predicted values used to meet the performance (or power) requirements and the actual values reported by PMCs for performance (or by power monitors for power consumption). The standard deviation over PAAE determines how far the prediction is from the optimal point.

#### 4.1.2.2 Single-Core offline evaluation

We performed evaluation for a subset of P-States, and Cl-States. The P-States combinations we evaluate are 0.8 GHz, 1.6 GHz, 2.4 GHz; similarly, we consider evaluation at 0%, 10%, 30%, and 50% variation in sleep cycles.

In the single-core offline validation technique, the PAAE is computed using the error between the PMCs gathered to predict power, and performance, and the value recorded using the PMCs at the corresponding number of instructions retired. This information is stored in the realigned trace file. The PAAE, and STDEV are computed over a period of 300 seconds.



(a) astar (b) calculix (c) mcf

Figure 4.3 Percentage error in performance, and power prediction for SPEC benchmarks Astar (mid memory intensive), Calculix (compute intensive), and Mcf (memory intensive).

Table 4.5 shows the power prediction error when switching between P-States across current P-State ($P_{\mathrm{c}}$) to future P-State ($P_{\mathrm{f}}$). Our results have shown that the PAAE over the subset of switches in P-States is less than 3% (mean), while the maximum error is 5.75%.

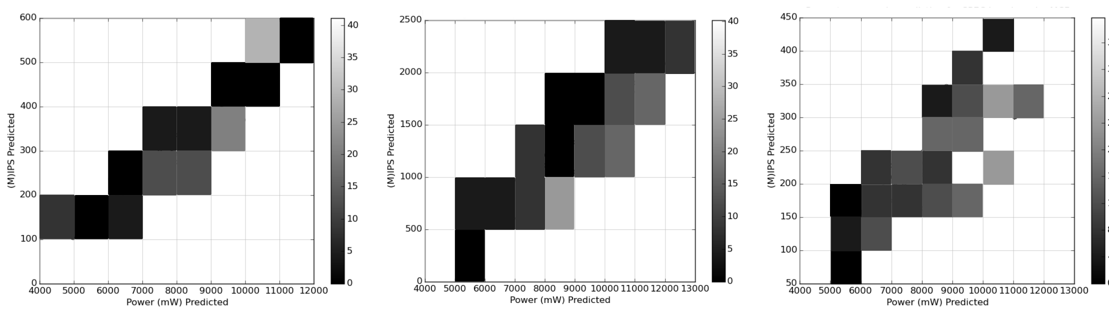
Table 4.6 details the performance prediction error when switching between P-States across current P-State ($P_{\mathrm{c}}$) to future P-State ($P_{\mathrm{f}}$). The worst-case scenario PAAE is 16.3% for the memory-intensive benchmark *mcf*. Also note that there exists a collinearity between PAAE, and the ratio of change in P-State. For instance, observe there is linear rise (or decrease) in PAAE for benchmark *Astar* when switching from current P-State to a higher P-State (or lower).

Table 4.7 shows the power prediction error when switching between Cl-States across current Cl-State ($Cl_{\mathrm{c}}$) to future Cl-State ($Cl_{\mathrm{f}}$) at 1.8 GHz. Our results have shown that the PAAE over the subset of switches in Cl-State is less than 5% (mean), with a maximum error of 10%.

Table 4.8 shows the performance prediction error when switching between Cl-States across current Cl-State ($Cl_{\mathrm{c}}$) to future Cl-State ($Cl_{\mathrm{f}}$) at 1.8 GHz. The maximum error observed when switching between the Cl-States is 15.7% (with maximum error of 22.43% for *GemsFDTD*).

The results obtained so far demonstrate that the error across numerous switches, for both P-States, and Cl-States, in an offline modelling technique is “acceptable” [61, 59, 48, 60, 70, 69], and determine the real behaviour of the application.

We demonstrate the effectiveness of the technique using three benchmarks from the SPEC suite with ranging from compute-intensive to memory-intensive. Figure 4.3 represents the predicted performance ($y$-axis) and power ($x$-axis) for *astar* (mid memory intensive), *mcf* (memory bound) and *Calculix* (compute bound) at nine P-States and six Cl-States. The initial configuration is set to 0.8 GHz and Cl-State zero. Each point in the graph represents



(a) astar (b) calculix (c) mcf

Figure 4.4 Percentage error per grid in performance, and power prediction for SPEC benchmarks Astar (mid memory intensive), Calculix (compute intensive), and Mcf (memory intensive).

a unique hardware configuration. The colour-map indicates the maximum PAAE between performance, and power ($\max_{\text{PAAE}} = \max(\text{PAAE}_{\text{power}}, \text{PAAE}_{\text{performance}})$). Observe that there exist multiple points grading from *black* (low PAAE) to white (higher PAAE) distributed across the grid. Figure 4.4 represents the data plotted in Figure 4.3 in terms of maximum error per sub-grid, where a configuration exists. Figures 4.3 and 4.4 show that there exists at least one prediction with less than 15% error. This behaviour is observed across all SPEC benchmarks.

Table 4.5 PAAE, and STDEV for power when switching between P-States.

| Benchmarks | Lower - Higher | | | | | | Higher - Lower | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 0.8-1.0 | | 0.8-1.6 | | 0.8-2.4 | | 1.0-0.8 | | 1.6-0.8 | | 2.4-0.8 | |
| | PAAE | STDEV | PAAE | STDEV | PAAE | STDEV | PAAE | STDEV | PAAE | STDEV | PAAE | STDEV |
| astar | 1.772 | 1.282 | 1.492 | 0.971 | 1.172 | 0.779 | 1.735 | 0.670 | 1.718 | 0.713 | 2.245 | 2.580 |
| bzip2 | 1.927 | 0.945 | 1.922 | 2.482 | 1.401 | 0.981 | 1.136 | 0.679 | 0.824 | 0.525 | 2.053 | 1.727 |
| calculix | 1.267 | 1.195 | 1.274 | 1.133 | 3.480 | 2.098 | 1.114 | 0.572 | 2.371 | 2.245 | 2.356 | 2.042 |
| gemsFDTD | 3.882 | 2.773 | 3.541 | 2.127 | 4.238 | 2.303 | 3.739 | 2.952 | 3.781 | 2.241 | 3.357 | 1.704 |
| gobmk | 1.054 | 0.940 | 1.339 | 1.605 | 2.409 | 3.201 | 0.856 | 0.652 | 0.697 | 0.466 | 0.665 | 0.572 |
| hmmer | 2.600 | 0.528 | 2.523 | 0.861 | 3.062 | 1.452 | 2.738 | 0.618 | 2.696 | 0.598 | 2.307 | 0.927 |
| lbm | 0.849 | 0.686 | 1.824 | 1.101 | 2.155 | 1.380 | 0.840 | 0.697 | 1.431 | 0.825 | 1.473 | 0.771 |
| leslie3d | 1.095 | 0.862 | 1.324 | 1.283 | 2.036 | 1.200 | 1.240 | 0.958 | 1.165 | 0.931 | 1.377 | 1.233 |
| libquantum | 2.676 | 3.906 | 1.643 | 1.255 | 2.435 | 1.953 | 3.330 | 2.766 | 1.957 | 1.496 | 1.852 | 1.852 |
| mcf | 3.172 | 2.355 | 3.785 | 2.677 | 5.666 | 2.249 | 2.419 | 2.656 | 2.292 | 2.227 | 2.422 | 2.724 |
| povray | 1.143 | 0.615 | 2.049 | 1.093 | 1.897 | 3.250 | 1.234 | 0.514 | 1.396 | 0.546 | 1.060 | 0.466 |
| soplex | 5.132 | 1.722 | 2.761 | 1.629 | 2.816 | 1.163 | 5.775 | 1.988 | 5.687 | 1.880 | 4.135 | 1.706 |
| xalancbmk | 2.522 | 1.587 | 3.627 | 1.580 | 5.807 | 1.457 | 1.986 | 1.358 | 1.617 | 1.154 | 1.401 | 1.242 |
| **Avg.** | 2.238 | 1.492 | 2.239 | 1.523 | 2.967 | 1.805 | 2.165 | 1.314 | 2.126 | 1.219 | 2.054 | 1.504 |

Table 4.6 PAAE, and STDEV for performance when switching between P-States.

| Benchmarks | Lower - Higher | | | | | | Higher - Lower | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 0.8-1.0 | | 0.8-1.6 | | 0.8-2.4 | | 1.0-0.8 | | 1.6-0.8 | | 2.4-0.8 | |
| | PAAE | STDEV | PAAE | STDEV | PAAE | STDEV | PAAE | STDEV | PAAE | STDEV | PAAE | STDEV |
| astar | 9.409 | 16.745 | 11.174 | 14.054 | 24.076 | 9.518 | 7.923 | 8.647 | 9.126 | 9.41 | 16.242 | 14.443 |
| bzip2 | 13.163 | 11.362 | 17.277 | 11.752 | 22.979 | 15.542 | 13.061 | 10.495 | 19.802 | 17.591 | 22.663 | 13.478 |
| calculix | 22.19 | 13.4358 | 19.24 | 15.115 | 26.968 | 20.412 | 17.219 | 26.997 | 18.165 | 24.183 | 19.888 | 23.932 |
| gemsFDTD | 21.687 | 15.48 | 23.884 | 14.5 | 15.65 | 14.648 | 21.707 | 15.58 | 22.433 | 12.316 | 12.709 | 9.618 |
| gobmk | 8.968 | 7.824 | 8.742 | 8.686 | 9.236 | 7.158 | 8.852 | 7.607 | 8.889 | 9.119 | 9.614 | 8.045 |
| h264ref | 6.095 | 4.689 | 9.177 | 14.613 | 1.887 | 1.353 | 6.116 | 4.707 | 7.705 | 8.513 | 1.94 | 1.42 |
| hmmer | 5.704 | 5.982 | 6.088 | 5.342 | 6.999 | 4.742 | 5.985 | 7.567 | 6.136 | 5.337 | 7.434 | 5.26 |
| lbm | 6.931 | 5.103 | 5.824 | 4.122 | 5.773 | 3.866 | 6.843 | 4.984 | 5.734 | 4.009 | 5.629 | 3.771 |
| libquantum | 8.602 | 6.359 | 10.3 | 7.095 | 12.577 | 10.284 | 8.635 | 6.521 | 9.545 | 6.445 | 10.541 | 7.697 |
| mcf | 11.081 | 9.698 | 17.721 | 15.106 | 29.327 | 20.954 | 10.441 | 9.11 | 14.672 | 12.777 | 21.431 | 11.328 |
| povray | 6.15 | 4.415 | 12.09 | 22.553 | 6.932 | 5.114 | 6.241 | 4.673 | 9.145 | 10.637 | 7.121 | 5.42 |
| soplex | 8.829 | 6.744 | 10.344 | 8.555 | 16.778 | 13.797 | 8.192 | 5.758 | 9.039 | 6.553 | 13.366 | 8.671 |
| xalancbmk | 9.653 | 8.565 | 19.31 | 13.979 | 33.991 | 16.745 | 9.702 | 9.427 | 15.461 | 9.203 | 25.172 | 11.007 |
| **Avg.** | 10.651 | 8.954 | 13.167 | 11.959 | 16.398 | 11.087 | 10.071 | 9.390 | 11.989 | 10.469 | 13.365 | 9.545 |

Table 4.7 PAAE, and STDEV for power when switching between Cl-States at 1.8GHz.

| Benchmarks | Lower - Higher | | | | | | Higher - Lower | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 0-10 | | 0-30 | | 0-50 | | 10-0 | | 30-0 | | 50-0 | |
| | PAAE | STDEV | PAAE | STDEV | PAAE | STDEV | PAAE | STDEV | PAAE | STDEV | PAAE | STDEV |
| astar | 3.190 | 2.308 | 2.686 | 1.748 | 2.110 | 1.402 | 3.123 | 1.206 | 3.092 | 1.283 | 4.041 | 4.644 |
| bzip2 | 3.469 | 1.701 | 3.460 | 4.468 | 2.522 | 1.766 | 2.045 | 1.222 | 1.483 | 0.945 | 3.695 | 3.109 |
| calculix | 2.281 | 2.151 | 2.293 | 2.039 | 6.264 | 3.776 | 2.005 | 1.030 | 4.268 | 4.041 | 4.241 | 3.676 |
| gemsFDTD | 6.988 | 4.991 | 6.374 | 3.829 | 7.628 | 4.145 | 6.730 | 5.314 | 6.806 | 4.034 | 6.043 | 3.067 |
| gobmk | 1.897 | 1.692 | 2.410 | 2.889 | 4.336 | 5.762 | 1.541 | 1.174 | 1.255 | 0.839 | 1.197 | 1.030 |
| hmmer | 4.680 | 0.950 | 4.541 | 1.550 | 5.512 | 2.614 | 4.928 | 1.112 | 4.853 | 1.076 | 4.153 | 1.669 |
| lbm | 1.528 | 1.235 | 3.283 | 1.982 | 3.879 | 2.484 | 1.512 | 1.255 | 2.576 | 1.485 | 2.651 | 1.388 |
| leslie3d | 1.971 | 1.552 | 2.383 | 2.309 | 3.665 | 2.160 | 2.232 | 1.724 | 2.097 | 1.676 | 2.479 | 2.219 |
| libquantum | 4.817 | 7.031 | 2.957 | 2.259 | 4.383 | 3.515 | 5.994 | 4.979 | 3.523 | 2.693 | 3.334 | 3.334 |
| mcf | 5.710 | 4.239 | 6.813 | 4.819 | 10.199 | 4.046 | 4.354 | 4.781 | 4.126 | 4.009 | 4.360 | 4.903 |
| povray | 2.057 | 1.107 | 3.688 | 1.967 | 3.415 | 5.850 | 2.221 | 0.925 | 2.513 | 0.983 | 1.908 | 0.839 |
| soplex | 9.238 | 3.100 | 4.970 | 2.932 | 5.069 | 2.093 | 10.395 | 3.578 | 10.237 | 3.384 | 7.443 | 3.071 |
| xalancbmk | 4.540 | 2.857 | 6.529 | 2.844 | 10.453 | 2.623 | 3.575 | 2.444 | 2.911 | 2.077 | 2.522 | 2.236 |
| **Avg.** | 4.028 | 2.686 | 4.030 | 2.741 | 5.341 | 3.249 | 3.897 | 2.365 | 3.826 | 2.194 | 3.697 | 2.707 |

Table 4.8 PAAE and STDEV for MIPS when switching between Cl-States at 1.8GHz.

| Benchmarks | Lower - Higher | | | | | | Higher - Lower | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 0-10 | | 0-30 | | 0-50 | | 10-0 | | 30-0 | | 50-0 | |
| | PAAE | STDEV | PAAE | STDEV | PAAE | STDEV | PAAE | STDEV | PAAE | STDEV | PAAE | STDEV |
| astar | 9.836 | 9.799 | 12.440 | 8.439 | 23.711 | 11.462 | 9.394 | 7.750 | 11.462 | 8.602 | 13.588 | 11.935 |
| bzip2 | 12.486 | 8.747 | 19.400 | 12.959 | 28.110 | 16.538 | 12.847 | 10.342 | 19.678 | 11.790 | 23.965 | 19.870 |
| calculix | 23.681 | 12.9840 | 17.184 | 16.378 | 37.767 | 22.141 | 17.679 | 14.590 | 19.893 | 8.623 | 23.289 | 6.008 |
| gemsFDTD | 17.164 | 14.747 | 15.109 | 9.980 | 24.321 | 16.465 | 16.932 | 15.113 | 15.312 | 11.467 | 23.785 | 17.364 |
| gobmk | 9.290 | 5.734 | 12.157 | 8.156 | 24.456 | 11.360 | 9.499 | 5.947 | 11.563 | 7.668 | 14.496 | 10.778 |
| h264ref | 10.302 | 5.408 | 10.646 | 10.863 | 25.193 | 12.028 | 10.669 | 6.161 | 12.767 | 8.660 | 15.129 | 9.913 |
| hmmer | 7.408 | 4.300 | 10.465 | 6.956 | 24.085 | 12.732 | 7.382 | 4.372 | 8.186 | 6.064 | 16.058 | 10.183 |
| lbm | 10.238 | 5.787 | 12.015 | 7.584 | 25.261 | 12.579 | 10.339 | 6.474 | 11.138 | 7.127 | 15.771 | 13.232 |
| libquantum | 8.857 | 6.154 | 12.238 | 10.852 | 24.808 | 12.907 | 8.865 | 5.854 | 13.729 | 12.958 | 14.651 | 10.185 |
| mcf | 9.689 | 7.396 | 15.263 | 10.717 | 27.495 | 13.709 | 10.650 | 9.180 | 14.185 | 18.806 | 19.416 | 21.161 |
| povray | 8.500 | 5.055 | 9.751 | 8.519 | 23.424 | 10.504 | 8.718 | 5.406 | 11.298 | 8.313 | 12.902 | 6.855 |
| soplex | 7.855 | 5.257 | 13.407 | 8.983 | 25.487 | 14.176 | 7.972 | 5.204 | 12.413 | 7.405 | 17.496 | 16.825 |
| xalancbmk | 9.528 | 7.612 | 10.698 | 9.382 | 31.952 | 15.761 | 9.000 | 6.764 | 11.377 | 11.185 | 14.587 | 8.897 |
| **Avg.** | 11.141 | 7.614 | 13.136 | 9.982 | 26.621 | 14.028 | 10.765 | 7.935 | 13.308 | 9.898 | 17.318 | 12.554 |

### 4.1.2.3 Single core online evaluation

The evaluation of the results for a single core model are described in two steps. First, we evaluate, and summarise the results for over 150 applications using `K-Means` [72]. Then, we expand the results obtained for each benchmark per benchmark suite. Irrespective of the form in which the results are presented – we predict performance, and power across a combinations of P-State, and Cl-State at runtime. The PAAE on a single core is computed using the error between 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, 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.

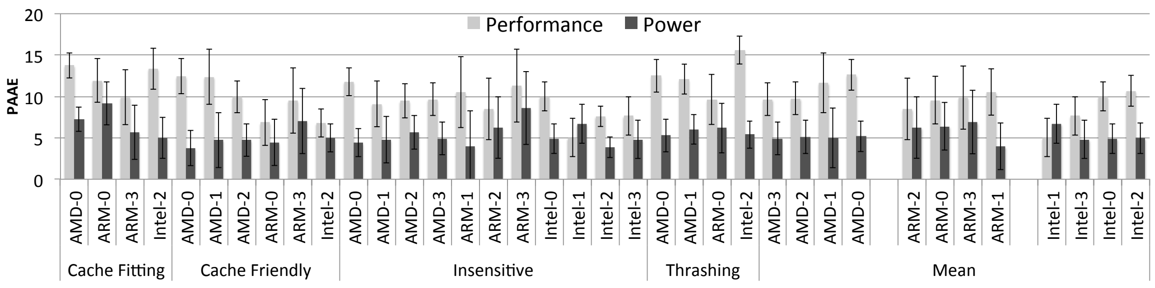


Figure 4.5 Average PAAE when predicting performance and power on a single core for across a combination of P-States, Cl-States and architectures. The error bars represent the STDEV

.

Figure 4.5 shows the average PAAE over all applications in a cluster on a single core, and Figure 4.7 presents the results for each application in detail on a single core. The results in the Figure 4.7 are organised as follows: Intel (top row (a)), AMD (middle row (b)), and ARM (bottom row (c)). We analyse the data points with the higher error and also pointed out the sources of error below.

ⓐ 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 [31] and generates more than 41000 LLC misses per million instructions retired and the models are not trained for that range. On the other hand, applications like *Lbm – 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 [73], 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 [74], 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.

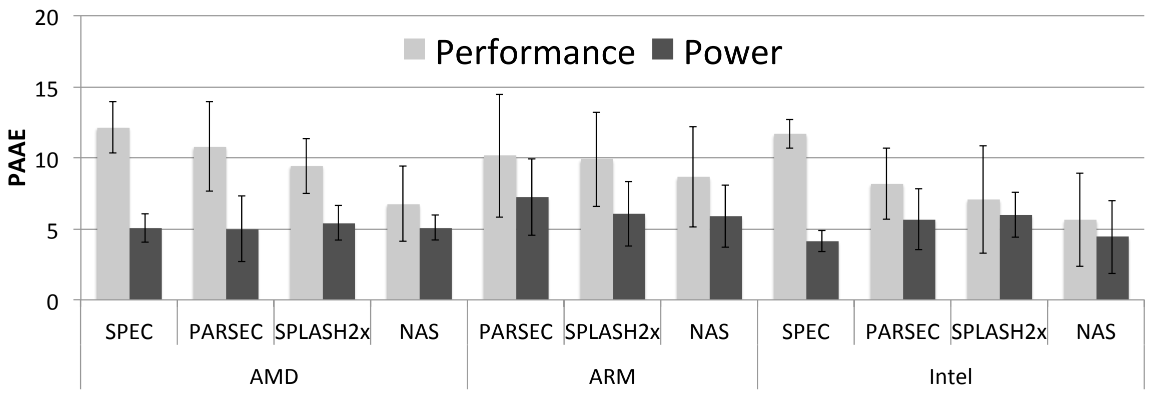
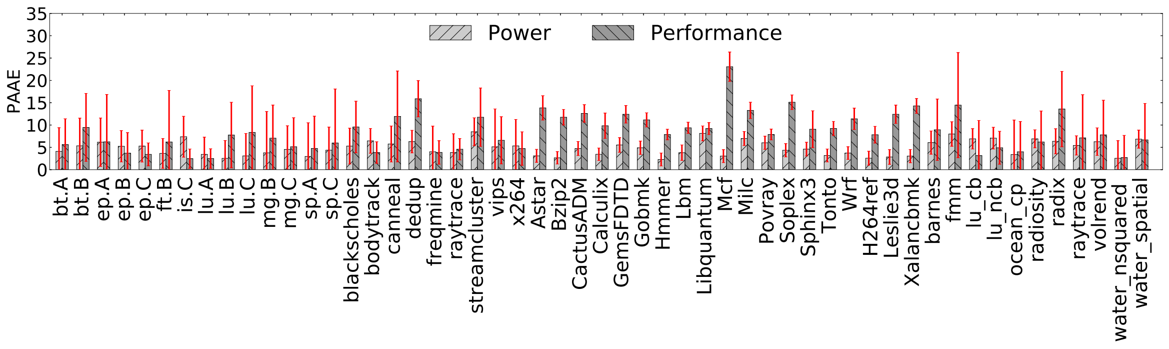


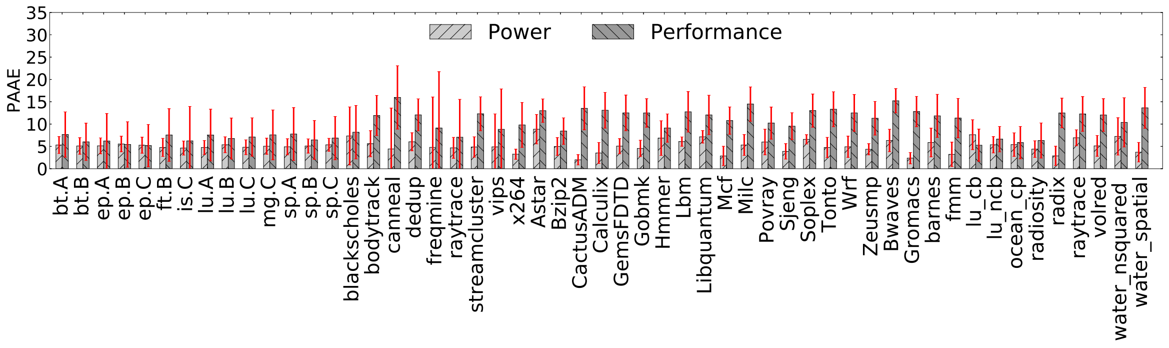
Figure 4.6 Average PAAE when predicting performance and power per suite across architectures. The error bars represent the STDEV.

Figure 4.6 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. The models to predict power can be built using standardised power meters and the models are built using PMCs that are available across architectures.

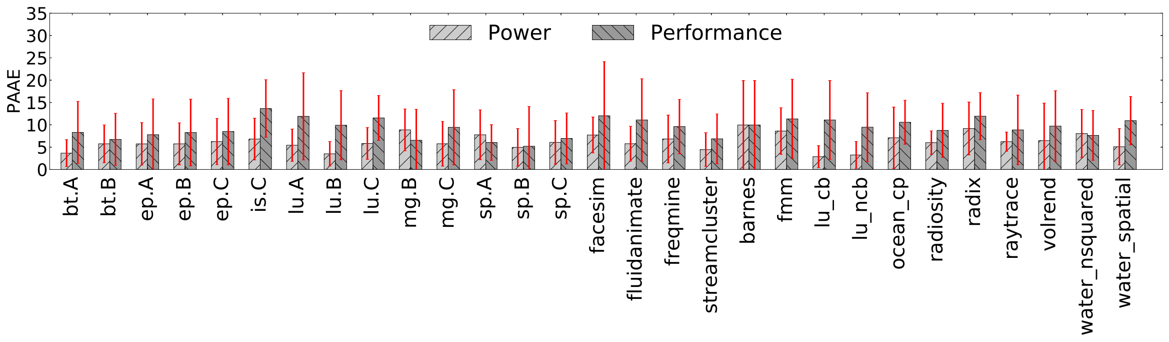
In addition, it is observed that when predicting performance at the future P-State on Intel processors, the error varies depending on the size of the leap between current P-State to future P-State or current Cl-State to future Cl-State. For every switch in P-State, we calculate



(a) Intel



(b) AMD



(c) ARM

Figure 4.7 Average PAAE when predicting performance and power on a single core for each benchmark from four benchmark suites across a combination of P-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, SPLASH2x

.

nishtala: should i separate the results? fix arm results.. maybe use overpic?!

the PAAE when switching between every combination of P-States. Similarly, we calculate the PAAE for Cl-States when the difference in switches is a multiple of five for every P-State. As can be seen in Table 4.9, with a higher switch in hardware configurations from the current

Table 4.9 PAAE when predicting performance for every combination of switch in P-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 P-State | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | - |
|---|---|---|---|---|---|---|---|---|---|
| P-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 |

configuration, a greater error is observed. This is because, we train the models using a small subset of benchmarks, and use it for a wide range of threads which are not a part of the training set. Similar results were observed across ARM, and AMD.

### 4.1.3 Conclusion

Our single-core models for the prediction technique are built such that the coefficients are not dependent on the number of applications. Instead they are based on the activity in each of the microarchitectural components. Therefore, it can scale across multi-node, multi-core data centres, and is aimed to ensure user-defined constraints for power, and performance are met. In contrast to prior work [75], REPP is not based on application signature or similarities between application. Instead, REPP leverages online monitoring of basic PMCs that provide application behaviour at runtime, which require a one-time, offline profiling effort per architecture, and not per application. Such data is then fed into statistical tools to predict performance and power, making REPP fast, accurate, and architecture-agnostic (**REFERENCE TABLE**).

However, such single core models are built, it is important to understand that modern data centres are equipped with multiple nodes consisting of racks of multicore servers, instead of a single core, thus making the existence of REPP alone futile. In the next section, we introduce REPP-H, which in contrast to REPP, is a modelling and prediction technique for both single-threaded, and multiprogrammed workloads in data centres.

## 4.2 REPP-H: Runtime Estimation for Performance and Power in Heterogeneous Data Centres

In the previous section, we introduced offline procedure to build performance, and power models, and also the offline, and online validation of the single-core models on three different architectures. In this section, we extend the model built to predict for single-threaded and multi-programmed in data centres by first tackling the issue of shared resource contention, and then use the predictions made to control power, and performance in multicore servers running multiprogrammed workloads.

### 4.2.1 Multi-core modelling

To predict total performance or power in multi-core architectures, we *aggregate* the results from each of the *single core* models (Section 4.1.1.4).

nishtala: lots of citations are missing

Single core models when exposed to multi-core prediction techniques are bound to suffer from an error due to the contention for shared resources [56, 48]. To compensate for this contention, we model the error in performance and power using four different types of workloads with variations in memory footprint and validate the models, by switching between combinations of P-States and Cl-States.

It impossible to be at two P-States or Cl-States at the same time. Therefore for training and validating the models, we generate multiple random tuples of P-State, Cl-State combinations per core within the minimum and maximum P-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 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 P-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 4.1.1.4) 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 PMC read (or RAPL register) and prediction for performance (or power) using *REPP* 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. The sleep interval of 250 ms was chosen empirically as most SPEC benchmarks have less than 1% change in performance or power at a particular P-State and allowed us to predict in the same phase.

$$Error = \sum_{i=0}^{comps} (\Delta_i \times AR_i) + constant \tag{4.6}$$

Equation 4.6 represents the multi linear regression model to predict the error due to shared resource contention. Where $\Delta_i$ represents the coefficient to be learnt and $AR_i$ represents the activity ratio of the individual components.

Intuitively, in a multi-core 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 modeled error for power and performance is added and subtracted for every prediction on a per thread level respectively.

nishtala: rewrite this part on

Then, we evaluate REPP-H across a wide range of performance and power constraints. Contrary to prior works [69, 67, 76, 77, 57], which predict power and performance at a system level, our work actually predicts at a system level on a per core basis.

nishtala: explain about the architecture here

### 4.2.2 REPP-H 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. To ensure that these constraints are met, REPP-H dynamically selects a configuration per core every 250 ms by performing a linear search for the P-State that is the nearest to the given constraint; next, REPP-H selects the Cl-State for the given P-State that satisfies the constraint. This selected configuration, P-State, Cl-State, is used to ensure that the constraint is met for each interval. This process is repeated across all cores 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). At runtime, for the spawned applications, REPP-H samples application behaviour periodically and uses the models built to predict performance and power at all configurations. REPP-H then selects the configuration for each interval to satisfy the local constraint.

This ability to satisfy constraints per core makes REPP-H better than previous works [69, 67, 76, 77, 57], allowing for multi-node, multi-core data centres running numerous application to satisfy a wide-range of performance and power requirements.

### 4.2.3 Evaluation

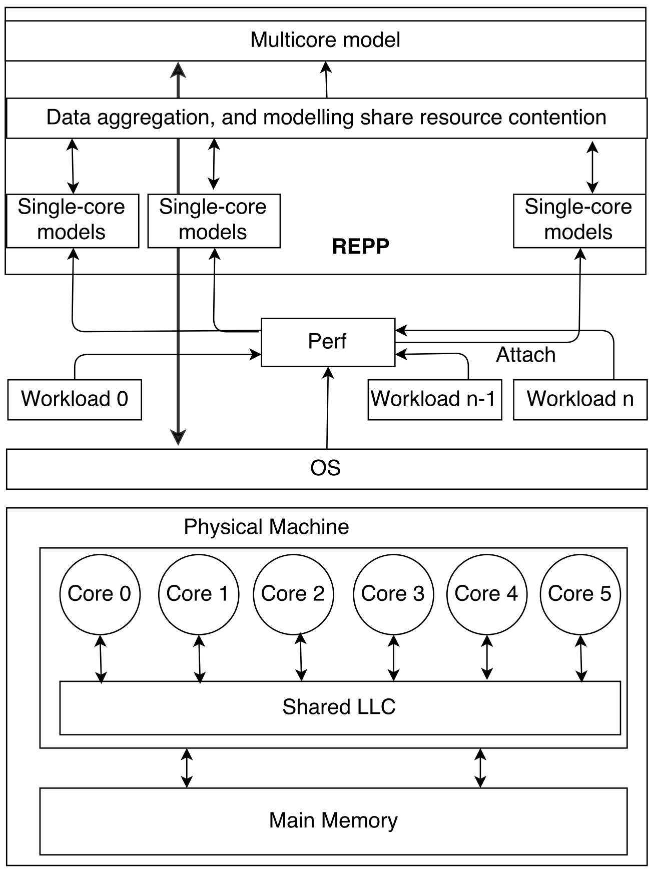


Figure 4.8 High-level view of REPP runtime system.

## 4.3 REPP-C

## 4.4 Implementation

MSR registers to change frequency cost to use

## 4.5 Scalability

nishtala: REPP ARCHITECTURE diagram

# CHAPTER 5

# Validating Methodology

# CHAPTER 6

# Hipster

# CHAPTER 7

# Related Work

# CHAPTER 8

# Conclusion

# References

[1] John Russell. ARM Unveils Scalable Vector Extension for HPC at Hot Chips. Accessed: 2016-08-09.

[2] Mont-Blanc. Mont-blanc. Accessed: 2016-08-09.

[3] Mont-Blanc 2. Mont-Blanc 2, European scalable and power efficient HPC platform based on low-power embedded technology. Accessed: 2016-08-09.

[4] Bull atos Technology. Mont-Blanc European project gains new impetus for its climb to Exascale. Accessed: 2016-08-09.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] M. Guevara, B. Lubin, and B. 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.

[11] 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.

[12] 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.

[13] Intel. *Intel 64 and IA-32 Architecture Software Developer's Manual.* Intel Corp.

[14] AMD. *AMD64 Architecture Prog. Manual Volume 2: System Prog.* AMD Corp.

[15] ARM. *Infocenter for ARM Cortex-A57.* ARM.

[16] Applied Micro XGene2. http://goo.gl/XA04r1, 2016. Online; accessed 31 August 2016.

[17] B. Rountree, D. K. Lowenthal, M. Schulz, and B. 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.

[18] H. Peter Anvin. MSR tools. Accessed: 2014-10-14.

[19] Wattsup Pro. https://www.wattsupmeters.com/secure/products.php, 2016. Online; accessed 31 August 2016.

[20] ARM Juno Power. https://github.com/ARM-software/devlib/blob/master/src/readenergy/readenergy.c, 2016. Online; accessed 31 August 2016.

[21] ARM. Arm juno power registers, 2016. Accessed: 2016-6-27.

[22] ARM. SYS_POW_SYS Register, https://goo.gl/fmTTQi.

[23] Jason Mars and Lingjia Tang. Whare-map. *ACM SIGARCH Computer Architecture News*, 41(3):619, 7 2013.

[24] Hailong Yang, Alex Breslow, Jason Mars, and Lingjia Tang. Bubble-flux. *ACM SIGARCH Computer Architecture News*, 41(3):607, 7 2013.

[25] 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.

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

[27] Harshad Kasture, Davide B. Bartolini, Nathan Beckmann, and Daniel Sanchez. Rubik. In *Proceedings of the 48th International Symposium on Microarchitecture - MICRO-48*, pages 598–610, New York, New York, USA, 12 2015. ACM Press.

[28] 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.

[29] 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.

[30] 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.

[31] 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.

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

[33] 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.

[34] 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.

[35] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta. The splash-2 programs: characterization and methodological considerations. In *22nd International Symposium on Computer Architecture*, ISCA 1995, pages 24–36, June 1995.

[36] 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.

[37] Memcached. http://memcached.org, 2016. Online; accessed 31 August 2016.

[38] Elasticsearch. https://github.com/elastic/elasticsearch, 2016. Online; accessed 31 August 2016.

[39] 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.

[40] L.A. Barroso, J. Dean, and U. Holzle. Web search for a planet: the google cluster architecture. *IEEE Micro*, 23(2):22–28, 3 2003.

[41] Faban. https://faban.org, 2016. Online; accessed 31 August 2016.

[42] 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. *ACM SIGARCH Computer Architecture News*, 40(1):37, 4 2012.

[43] Linux. Perf: Linux profiling with performance counters.

[44] 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.

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

[46] ARM Limited. ARM ® Cortex ® -A53 MPCore Processor Technical Reference Manual.

[47] ARM Limited. ARM ® Cortex ® -A57 MPCore Processor Revision: r1p0 Technical Reference Manual.

[48] 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.

[49] 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.

[50] Jason Mars and Lingjia Tang. Whare-map: Heterogeneity in "homogeneous" warehouse-scale computers. In *Proceedings of the 40th Annual International Symposium on Computer Architecture*, ISCA '13, pages 619–630, New York, NY, USA, 2013. ACM.

[51] J. Mars, Lingjia Tang, and R. Hundt. Heterogeneity in homogeneous; warehouse-scale computers: A performance opportunity. *Computer Architecture Letters*, 10(2):29–32, July 2011.

[52] 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.

[53] Intel. Intel intelligent power node manager.

[54] Openstack. Openstack.

[55] Tivoli Management Framework. Intel intelligent power node manager.

[56] Sergey Blagodurov. *Addressing Shared Resource Contention in Datacenter Servers*. PhD thesis, Simon Fraser University, August 2013.

[57] 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.

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

[59] 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.

[60] Rajiv Nishtala, Xavier Martorell, Vinicius Petrucci, and Daniel Mosse. Repp-h: Runtime estimation of performance-power on heterogeneous data centers. In *Proceedings of the 28th International Symposium on Computer Architecture and High Performance Computing*, SBAC-PAD '16. IEEE Press, 2016.

[61] 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.

[62] 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.

[63] 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.

[64] 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.

[65] R. Bertran, M. Gonzalez, X. Martorell, N. Navarro, and E. Ayguade. A Systematic Methodology to Generate Decomposable and Responsive Power Models for CMPs. *IEEE Transactions on Computers*, 62(7):1289–1302, 2013.

[66] T.E. Carlson, W. Heirman, and L. 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.

[67] 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.

[68] T. M. Le. A study on Linux Kernel Scheduler Version 2.6.32.

[69] 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.

[70] William Lloyd Bircher and Lizy K. John. Complete system power estimation: A trickle-down approach based on performance events. In *Proceedings of the IEEE/ACM International Symposium on Performance Analysis of Systems Software*, ISPASS 2007, pages 158–168, Washington, DC, USA, 2007. IEEE Computer Society.

[71] 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.

[72] 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.

[73] Dimitrios Chasapis and Marc Casas, et al. ParsecSs: Evaluating the Impact of Task Parallelism in the Parsec Benchmark Suite. In *ACM TACO*, 2015.

[74] 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.

[75] 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.

[76] Ryan Cochran, Can Hankendi, Ayse K. Coskun, and Sherief Reda. Pack & cap: Adaptive dvfs and thread packing under power caps. In *Proc. of the 44th Annual IEEE/ACM Int'l Symposium on Microarchitecture*, MICRO-44, pages 175–185. ACM, 2011.

[77] 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.