ENERGY-EFFICIENT MANAGEMENT OF RECONFIGURABLE COMPUTERS

by

Rathijit Sen

A dissertation submitted in partial fulfillment of
the requirements for the degree of

Doctor of Philosophy

(Computer Sciences)

at the

UNIVERSITY OF WISCONSIN–MADISON

2016

Date of final oral examination: 05/13/16

The dissertation is approved by the following members of the Final Oral Committee:
David A. Wood, Professor, Computer Sciences
Mark D. Hill, Professor, Computer Sciences
Mikko H. Lipasti, Professor, Computer Sciences
Gurindar S. Sohi, Professor, Computer Sciences
Michael M. Swift, Associate Professor, Computer Sciences

To my parents Meena Sen and Ranjit Kumar Sen

ACKNOWLEDGMENTS

I am honored to have Prof. David Wood as my advisor. He has patiently guided me and supported me over so many years. David is an erudite scholar and outstanding professor. I have learned so much from him about computer architecture and analytical modeling. He encouraged me to consider the theoretical underpinnings in addition to the practical feasibility of problem solutions. He helped me explore ideas, refine drafts of writeups, improve my presentation skills, and network with other researchers through attending conferences. Thank you, David, for your time, support, and help.

I would like to thank the other members of my committee, Profs. Mark Hill, Mikko Lipasti, Guri Sohi, and Mike Swift, for giving me valuable feedback on my work. I have learned a lot from their vast experience and deep insight. Mark and Mike gave me great inputs for improving my papers and posters. Mike pointed me to many interesting papers. I have learned a lot from Mark about queueing models and how to present my work better. I had insightful discussions with Mikko about computer architecture research. Guri gave me power meters that I used to measure server power for this work.

I am also grateful to the Multifacet research group for providing me with necessary infrastructure to do the work in this dissertation. The server clusters, Lapis full-system simulator, Ruby memory hierarchy simulator, and Wisconsin commercial workloads have been invaluable to me. Technical discussions at group meetings were highly instructive.

Dan Gibson helped me a lot with setting up of the simulation infrastructure. Somayeh Sardashti and Hamid Reza Ghasemi also helped with creating simulation checkpoints.

This work was supported in part by the National Science Foundation (CNS-0916725, CCF-1017650, CNS-1117280, CCF-1218323, CNS-1302260, CCF-1438992, CCF-1533885), Microsoft Corporation (MSN140822), Sandia National Labs/DOE (MSN123960/DOE890426), and a University of Wisconsin Vilas award.

I would like to thank Prof. Somesh Jha and Prof. Jignesh Patel for their collaboration and help on multiple projects in the areas of static program analysis and databases. Jignesh gave me an opportunity to contribute to the Quickstep project and generously supported me during my final summer at UW-Madison. I look forward to continuing our collaboration. I would also like to thank Prof. Tom Reps for supporting me during my first year at UW-Madison and for giving me an opportunity to work on projects involving static analysis of executables.

My current and former office mates—Jayneel Gandhi, Derek Hower, Lena Olson, and Somayeh Sardashti—have given me company and have always been ready to help me. I have enjoyed countless hours chatting with them. I will miss Lena's delicious home-baked muffins. Derek helped me understand many details of the Ruby simulator.

I have enjoyed working with Gagan Gupta on a long project involving static analysis and having many conversations with him about computer architecture, emerging technologies, reading groups, paper reviews and rebuttals. I also explored other projects with Arkaprava Basu, Asim Kadav, Nilay Vaish, Cong Wang, and Jianqiao Zhu. Discussions with Siddharth Barman helped me appreciate different perspectives on theory and systems research. Jason Power's elevator pitch workshop was useful and fun.

I have also benefited from interacting with many other excellent colleagues including Muhammad Shoaib Bin Altaf, Newsha Ardalani, Piramanayagam Arumuga Nainar, Raghuraman Balasubramanian, Emily Blem, Jayaram Bobba, Evan Driscoll, Polina Dudnik, Yasuko Eckert, Christopher Feilbach, Venkatraman Govindaraju, Swapnil Haria, Bill Harris, Joel Hestness, Nick Kidd, Marc De Kruijf, Akash Lal, Junghee Lim, Jaikrishnan Menon, Sanketh Nalli, Tony Nowatzki, Marc Orr, Sankaralingam Pannerselvam, Matt Sinclair, Srinath Sridharan, Swaminathan Sundararaman, Aditya Thakur, Aditya Venkataraman, Haris Volos, and Hongil Yoon.

A number of external researchers, including UW-Madison computer architecture alumni, gave useful feedback on my work. I would like to acknowledge Luiz Barroso, Edouard Bugnion, Lisa Hsu, Mike Marty, Kathryn McKinley, and Ravi Rajwar for helpful suggestions and comments. The Architecture Affiliates meetings helped to get useful feedback also from Alaa Almeldeen, Brad Beckmann, John Davis, Dan Gibson, Peter Hsu, Konrad Lai, Kevin Moore, Steve Reinhardt, Greg Thorson, Greg Wright, and others.

I learned about new applications for analytical models while working with Trishul Chilimbi, Wei Huang, Srilatha Manne, and Indrani Paul during internships at Microsoft and AMD. I also enjoyed interacting with Manish Arora, Joseph Greathouse, and fellow interns Lavanya Subramanian and Zhe Wang. Weekend trips around and near Seattle with Arkaprava Basu, Tushar Krishna, and Shekhar Srikantaiah were enjoyable.

Angela Thorp, our graduate program coordinator, always helped me with understanding and following graduate school and departmental procedures. The Computer Systems Lab staff, and in particular Tim Czerwonka, promptly resolved all issues with the computing infrastructure that I ran into while doing my work.

Prof. Y. N. Srikant (IISc) and Prof. Reinhard Wilhelm (Saarland University) have greatly encouraged me. I wish to also acknowledge the friendship of some wonderful people, among them being Prof. Jan Reineke (Saarland University), Oindrilla Gupta, Kuntal Dey, Mohamed Abdel Maksoud, S. V. N. Narayana Rao, and Arpan Sen.

My sister Rakhee, brother-in-law Sandip, and niece Swagnita have brought me joy with their kindness, encouragement, and arrangements for many memorable excursions.

It has been a long and difficult journey, one on which I could not have persevered without the steadfast support of my parents. They have been a pillar of strength for me. I dedicate this dissertation to them. Thank you for your boundless love, encouragement, and good wishes throughout the years.

CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

Power and energy consumption are first-order constraints on the design and operation of computer systems today. Improving energy efficiency reduces the amount of energy needed to perform a given computation as well as enables more computation to be performed for the same amount of energy. This saves operational costs to use these systems as well as capital costs to provision for them.

Conventionally, energy proportionality (energy consumption in proportion to the work done, or equivalently, power consumption in proportion to utilization/performance/load served) as proposed by Barroso and Hölzle, has been the gold standard of an ideal system's energy efficiency. While this model is valid for fixed-resource systems, modern systems are reconfigurable in many aspects, allowing them to adapt to changing workload characteristics. Smart reconfigurability increases energy efficiency. However, we show that reconfigurability invalidates the conventional notions of ideal energy proportionality if the system starts to behave super-proportionally. Super-proportional systems provide more performance (or work) in proportion to the power (or energy) used. We propose a new ideal model, Energy Optimal Proportional (EOP), that subsumes the conventional model and improves upon it by also accounting for super-proportional systems.

EOP can guide system designers to improve the maximum efficiency attainable over the operating range and forms a basis for comparisons of energy efficiency across systems. Power-performance Pareto optimality, on the other hand, can guide system operators to manage load and configure resources appropriately to make the current system execute efficiently. We propose a new intellectual framework that interrelates these two complementary energy efficiency goals.

The rest of this dissertation focuses on energy-efficient management. We develop new reactive governors that coordinate processor frequency (and voltage) and hardware

prefetching to improve energy efficiency on a real (Haswell) server. We also propose a space-efficient hardware mechanism to estimate temporal locality (reuse) in cache accesses. The estimated distributions can be used by our new analytical models for cache performance to drive resizing decisions of the last-level cache.

Finally, we propose a new classification system for system reconfiguration capabilities. The classification is based on the semantics of what the reconfiguration affects—computation, communication, storage, scheduling, speculation. We hope that this classification will be insightful to future researchers while exploring the space of reconfigurable systems, in categorizing existing work and in identifying coordination options that have been less well explored.

1 INTRODUCTION

Computers are used extensively in a variety of applications, e.g., in hosting and searching the Web, in predicting the weather, in managing online markets and social networks, in analyzing genomes, in processing signals to detect gravitational waves, etc. Increased computational demands over the years have resulted in significant energy and power costs to provision for and operate systems. Today, power and energy are among the most critical constraints for the design and use of computer systems [22, 160].

Datacenters host large numbers of computers that serve the computational needs of its users. A recent report [164] states that U.S. datacenters used around 91 billion KWh of electricity, equivalent to the energy consumption of 34 coal-fired 500 MW power plants, in 2013. This is projected to rise to 140 billion KWh of electricity, equivalent to the energy consumption of 50 coal-fired 500 MW power plants and costing around $13 billion, in 2020. Thus, reducing the energy consumption of datacenters will have significant economic and environmental benefits.

Datacenters use energy not only for running servers, but also for operating power distribution systems, cooling systems, lighting systems, etc. The PUE (Power Usage Effectiveness) metric [16] was developed to quantify these extra energy overheads. PUE is the ratio of the total facility energy to the IT equipment energy. Smaller values of PUE are better since they indicate that a greater portion of the total energy is being used to run the IT equipment to do useful work instead of being used up by the non-IT systems mentioned above.

The PUE metric has been very influential in driving down overheads due to non-IT infrastructure. In the early days of the metric, PUE values of around 3.0 were common. In recent years, the average PUE value is 1.7 [108]. A number of modern datacenters report far lower PUE values of around 1.1 [73, 83, 156, 166, 210]. Sophisticated cooling

technologies allow even lower PUE values [1]. PUE, however, does not track overheads in IT infrastructure. For low PUE values, most of the datacenter energy is used by the IT equipment, particularly, by the servers.

In order to quantify a part of the energy losses in the servers, Barroso and Hölzle proposed the SPUE (Server PUE) metric [104]. This tracks losses in the server power supply units (PSUs) that happen due to inefficiencies in converting A.C. power to low voltage D.C. power required by components within the server, e.g., processors, disks, fans, etc. Some modern PSUs can attain upwards of 95% efficiency over a portion of their operating range.

Barroso and Hölzle [22] also observed that servers lose energy during computation due to under-utilization. They observed that the energy efficiency (work done per unit of energy used) of servers peak at maximum utilization but drop drastically as the utilization decreases. The reason for this is energy losses that persist even when the server is idle. At zero utilization, that is, when a server is not performing any useful work, it still draws power. This energy consumption is due to leakage in processor components, DRAM refresh, powered-on hard disks, fans, other components in the motherboard, and high PSU inefficiency at low loads. At higher utilizations, the server needs more energy to perform the given computations. This reduces the relative overheads due to the other components and improves the energy efficiency.

This dependence of server energy efficiency on server utilization means that, when not fully utilized, less work gets done per unit of energy used, or equivalently, more energy is needed to do the same amount of work. This is problematic since servers in datacenters are typically only 10–50% utilized [22], thus using more energy to perform the computations than they would use if fully utilized.

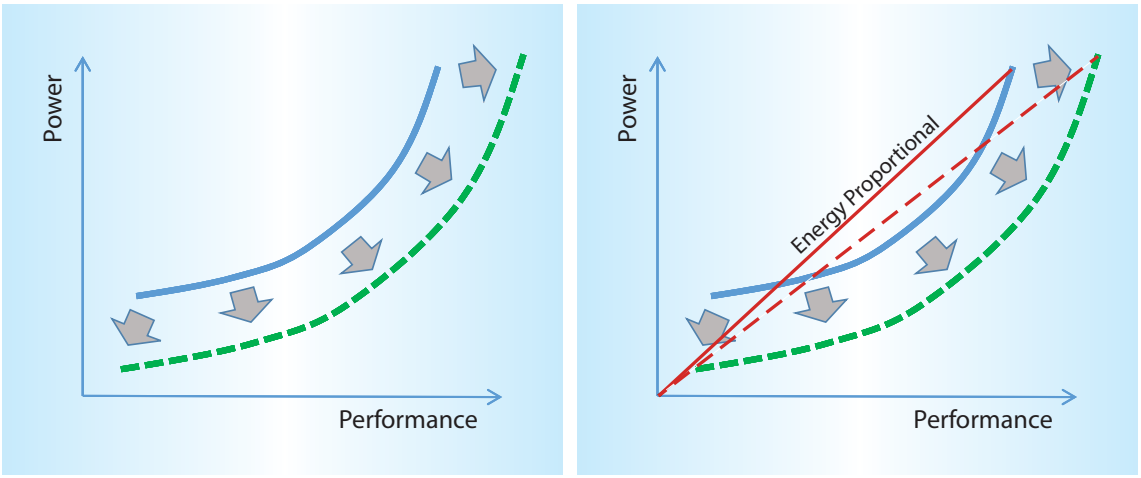
To eliminate utilization-dependent energy losses, Barroso and Hölzle [22] advocated

for "energy-proportional" system designs. Such systems would use energy in proportion to work, or equivalently, power in proportion to utilization. This would ensure that they retain their maximum energy efficiency even at low utilizations. They would significantly save server energy consumption by preventing the drastic drop in energy efficiency at low utilizations.

Power overheads that persist when the system is idle make it non–energy-proportional. Thus, perfect energy-proportional systems must have zero idle power. This model of an ideal system has inspired system designers to build systems that have low idle power (thus, have low power overheads) and a wide dynamic power range (so that the relative power overheads are low at high utilizations).

Conventionally, attaining energy proportionality has been the cherished ideal for minimizing energy waste. However, with recent technological and architectural advances, modern systems may exhibit super-proportional behavior, that is, they exceed the energy efficiency of energy-proportional systems. In these systems, being only energy-proportional is significantly wasteful. Proportionality is a lesser prize to aim for in this changed situation with substantial energy savings remaining to be realized with more ambitious goals.

Figure 1.1 illustrates this point. Figure 1.1a shows a representative power vs performance profile with voltage and frequency scaling for an Intel Haswell processor [98]. Power consumption increases *non-linearly* with performance. Equivalently, energy consumption increases *non-linearly* with the amount of work (computation) done. The dashed line shows how improvements in processor design have lowered the power-performance profile, enabling more energy-efficient operations. Figure 1.1b augments the original graph with "Energy-Proportional" lines. Most of the power-performance curve lies below the energy proportional line. This means that configurations represented by points on the



(a) Per Hammarlund et al. [98] (b) Augmented with Energy-Proportional lines

Figure 1.1: Trends in Processor power-performance profiles.

curve are more efficient, that is, super-proportional. These configurations use less power (and less energy) than a perfect energy-proportional system at the same performance (or, serviced load). Chapter 3 corroborates this observation for our workloads, both batch and interactive ones. Intel's data suggests that this trend is increasing with technological and architectural advances.

Since energy proportionality, which is the conventional ideal model, is no longer sufficient to describe the energy efficiency potential of modern computers, we need to define new ideals for system energy efficiency. These ideals will be useful to system designers for building more efficient future systems and to system operators for operating current systems more efficiently.

Moreover, neither PUE, nor SPUE, nor energy proportionality quantifies waste in computational energy with reference to ideal system operations. We propose a new metric, CPUE (Computational PUE), to address this need.

Super-proportionality happens in conjunction with reconfiguration capabilities found

in modern computers. Many resources may be configurable, e.g., processor frequency (and voltage), cache size, prefetching ability, etc. Reconfigurability and super-proportionality together necessitate the development of new concepts and mechanisms to minimize computational energy waste.

The dissertation focuses on the energy efficiency of reconfigurable computers that may also exhibit super-proportional behavior. We develop new models for ideal system design and operations, new metrics for quantifying computational energy waste and attributing losses to root causes, and mechanisms to efficiently operate such systems.

1.1 Iron Law of Energy

Knowledge about ideal system efficiency and energy waste with respect to the ideal is not sufficient to correct the situation to eliminate energy waste. We also need to quantify the root causes of energy waste.

To do this, we draw inspiration from the popular Iron Law of (processor) Performance [72, 194] that decomposes workload execution time into three components as shown below:

$$\frac{\text{Time}}{\text{Program}} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Cycles}}{\text{Instruction}} \times \frac{\text{Time}}{\text{Cycle}}$$

This separation helps compiler writers to focus on improving the first component, computer architects to focus on improving the second component, and circuit designers to focus on improving the third component.

A similar decomposition of workload energy consumption does not exist. We now

propose a new Iron Law of Energy in Chapter 2 as follows:

$$E(c, l) = LUE(l) \times RUE(c, l) \times E_{min}, \qquad l > 0$$

where:

- $E(c, l)$ is the energy consumption by the system to do the work using configuration c at non-zero load (or, processing rate or, performance) l,
- $LUE(l)$ is the relative energy used by the most energy-efficient configuration for load l compared to the energy needed (E_{min}) by the most energy-efficient configuration (the optimal load can be different from l),
- $RUE(c, l)$ is the relative energy used due to operating with configuration c at load l compared to the energy needed by the most energy-efficient configuration for that same load l,
- E_{min} is the minimum energy needed by the system to do this work when operating at the optimal configuration and load.

Similar to the Iron Law of Performance, we expect that the above decomposition will help different actors to focus on particular aspects of energy consumption—system designers to focus on reducing E_{min} and system operators to focus on reducing LUE and RUE through load management and configuration management.

1.1.1 Load Management

One major source of wasted energy is due to suboptimal load levels, particularly low loads, serviced by the machine. This aspect is captured by the LUE term in the Iron Law of Energy. A value greater than 1 for LUE means that energy is wasted due to operating

with non-optimal loads. Chapter 2 shows that non-optimal loads can use in excess of 350% energy compared to that needed at optimal loads. System operators can prevent or mitigate this loss by managing load levels serviced by machines.

In a datacenter, load management is a global policy decision since multiple machines are affected by it, either to absorb load from or relinquish load to other machines so that the total serviced load is unaffected. This may not always be possible, e.g., in the case of stateful services with expensive state migration costs.

We do not focus on mechanisms to perform inter-server load management in this dissertation.

1.1.2 Configuration Management

Modern computers are reconfigurable in multiple ways and not being careful about the operating configuration can be extremely wasteful in terms of energy consumption. This aspect is captured by the RUE term in the Iron Law of Energy. A value greater than 1 for RUE means that energy is wasted due to operating with non-optimal configurations. As Chapter 2 shows, up to 51% more energy can be used by suboptimal configurations even if load is managed to be in the optimal range. For a given load, configuration management is local to the machine in the sense that other machines need not be aware of or affected by the configuration of the target machine.

This dissertation focuses primarily on configuration management of single machines—management of processor frequency (Chapters 3 and 5), cache prefetching (Chapter 3), and cache organization (Chapters 4 and 5).

1.2 Service-Level Agreements (SLA)-aware Governors

We use existing terminology of calling resource management policies as governors. System operators may need to use governors to ensure that runtime Service-Level Agreements (SLAs) are satisfied. There is thus a need for SLA-aware governors. Some examples of SLAs are:

- Maximize energy efficiency, that is, minimize energy used.
- Maximize performance while operating within a given power budget.
- Maximize power savings, that is, minimize power consumption, while meeting a given performance target.

Minimizing system energy consumption reduces the operational cost of machines and datacenters by reducing electricity bills. Limiting/capping power consumption has multiple benefits as follows.

1. It generates less heat and thereby reduces cooling needs and operational costs.
2. It requires provisioning for a smaller amount of total power to the system and datacenter. This reduces capital costs.
3. It allows for better utilization of datacenter capacity. Otherwise, the datacenter has to be provisioned for worst-case/nameplate power consumption of all servers whereas most servers are poorly utilized leading to stranded capacity. Improving datacenter capacity utilization can significantly reduce the Total Cost of Ownership (TCO) [21].

Operating under power constraints is important for both servers and mobile systems [98]. However, this may slow down computation. So the user may want to specify SLAs that include performance targets that should be reached.

Linux distributions include governors that manage processor frequency. Some of the well-known governors are Performance (highest frequency), Powersave (lowest frequency) and OnDemand (dynamically change frequency according to utilization). The OnDemand governor is the default for many distributions. However, these governors are not sufficient for managing modern servers. There are three main reasons why the existing Linux governors are inadequate:

1. The existing governors only control processor frequency. However, other aspects of the system, e.g., prefetching, number of active cores, etc. are also controllable at run time. System designers are increasingly making reconfigurable interfaces public and new governors should exploit those to increase energy savings.

2. There is no way for the user to specify SLAs/high-level management goals, e.g., minimize energy while meeting a performance target. The assumption made by the existing governors is that the user always wants peak performance or lowest power, but there could be other management goals. New governors need to be more expressive in order to meet the needs of the users.

3. Modern processors automatically transition to low-power states when idle. This significantly reduces the utility of the OnDemand governor. We will show in Chapter 3 that the OnDemand governor performs almost identically to the Performance governor for all of our workloads.

Chapter 3 shows that, for the SPECpower benchmark, the existing governors are significantly energy-inefficient for most load levels other than peak and idle. We see

significant inefficiencies for batch workloads as well. We develop new governors that address the shortcomings of the current governors by considering prefetch control and cache resizing in addition to frequency control, by being SLA-aware, and by aiming to constrain system operations to the Pareto frontier (Dynamic EO).

1.3 Contributions

The main contributions of this dissertation are:

1. **Definition of new ideals and metrics for reasoning about energy efficiency.**

   Instead of Energy Proportional (EP), we propose Energy Optimal Proportional (EOP) as the new *design ideal* for energy efficiency. A system that is EOP will always use minimum energy, E_{min}, (or equivalently, always have maximum efficiency) to do a given amount of work irrespective of the load. EOP thus characterizes a lower bound on the energy consumption, or equivalently, an upper bound on the energy efficiency of the given system. EOP will be helpful to system designers as they improve the system's maximum energy efficiency (reduces E_{min}) and make a greater portion of the operating range closer to EOP (reduces LUE over a greater range of loads).

   For system operators, we propose a new *operational ideal* called Dynamic Energy Optimal (Dynamic EO). This is determined by the set of system configurations that have the lowest power among all configurations that can serve the same load. System operators should strive to operate their system close to Dynamic EO. This will ensure that RUE is close to 1.

   We also propose a new metric, CPUE, for quantifying computational energy waste. Our new Iron Law of Energy in Chapter 2 helps quantify two operational con-

tributors to wasted energy—non-optimal loads and non-optimal configurations for serving the loads—that will help system operators to focus better on load management and configuration management.

Chapter 2 discusses these new ideals and energy consumption metrics in more detail.

2. **Development of new SLA-aware governors that control both processor frequency and cache prefetching.**

   In contrast to the existing Linux governors, our governors

   a) take user-specified SLAs into account while managing resources, and

   b) aim to constrain system operations to Dynamic EO.

   Chapter 3 shows that our new governors save significant energy compared to the existing Linux governors. Two of our workloads, `md` and SPECpower, show significant performance improvements for the same power budget with dynamic control of prefetch settings.

3. **Development of new cache performance models based on reuse distance properties to determine optimal cache size and associativity at runtime.**

   Since workloads differ in their cache utilization properties, being able to efficiently and accurately predict cache performance for different cache sizes and organizations is important for saving cache leakage energy, with controlled performance impact, by dynamically resizing the cache. Determining the optimal cache configuration without needing to try out all possible configurations requires low-cost models that can be used online to predict the performance of potential target cache configurations. Our analytical models are based on the reuse distance distributions of accesses to

the cache. The reuse distance of an element in an address stream is the number of unique elements accessed between two successive accesses to the same element. Chapter 4 develops online methods that monitor cache access streams to determine reuse distance distributions and use that to predict cache miss rates for any cache size and associativity.

In contrast to earlier work on way counters [175, 214] that can predict cache miss rates only for different associativities, our models can predict cache performance for different cache sizes and associativity not only for LRU, but also for other replacement policies such as PLRU, RANDOM, and NMRU.

4. **Development of a new governor that controls for cache organization (size, number of sets) and processor frequency.**

   Depending on the workload characteristics, the last-level cache can be resized and the processor frequency increased to get more performance for the same power budget. Chapter 5 develops a governor that controls these knobs to achieve 0.5–15% performance improvement for a given power budget.

5. **Development of a new classification system for system reconfiguration capabilities.**

   Computer architecture has been greatly enriched by classifications/taxonomies of various aspects of system design and operation, such as Flynn's classification of machine organizations [78], Hill's 3C classification of cache misses [103], Wang-Baer-Levy's classification of virtual-real cache hierarchies [227], etc. We propose a new classification system for reconfigurability, based on the semantics of reconfiguration knobs, in Chapter 6. We hope that this classification is more insightful than a component-based or mechanism-based classification of reconfiguration knobs.

1.4 Implications

The implications of our work are three-fold.

Firstly, the notion of conventional energy proportionality (EP) being a model of ideal system efficiency is no longer true. This has resulted from modern systems being reconfigurable and super-proportional. The energy efficiency of the system at its peak performing point can be significantly less than the best that it can achieve. Consequently, aiming to attain EP may result in significant lost opportunities in improving efficiency. Instead, system designers should aim to attain EOP.

This also means that comparisons between systems based on the energy consumptions at their peak performing points can be misleading since that may not be the best energy efficiency realizable on either system. EOP can form a basis for such comparisons.

Further, the policy of race-to-halt (running the system at its highest speed and then shutting it down to save power) can be suboptimal in terms of energy consumption since race-to-halt aims for attaining the EP power-performance profile, not the EOP profile. Race-to-halt is thus more of a performance-optimal policy rather than an energy-optimal one. As we will demonstrate with our new reactive governors for SPECpower [205] in Chapter 3, it may be more energy efficient to control processing speed so that the system is never under-utilized while still serving the offered load. This strategy minimizes idle time whereas the race-to-halt policy maximizes idle time. Our new governors employ a "jog-to-halt" policy that subsumes race-to-halt by selecting processing speeds that minimize energy consumption while meeting performance targets or power caps.

Secondly, while the intense focus on datacenter PUE has led to significant reductions in datacenter cooling overheads, it has also resulted in IT equipment inefficiencies being one of the largest contributors to energy waste in modern datacenters. With increasing computational demands and electricity costs and reducing PUE numbers, optimizing

computational energy through smart server reconfigurations and load balancing will result in more overall savings than it has in the past. It is important to have useful metrics that can guide such operating decisions. Our new metrics—CPUE, LUE, and RUE—can help to analyze and guide those decisions.

Thirdly, existing governors in Linux as well as RAPL capabilities in modern processors are inadequate for achieving maximum energy efficiency. Currently, Linux governors do not consider the full range of reconfiguration capabilities present in the system. Currently, RAPL also has the same limitation and additionally only guarantees a maximum power cap, but ignores performance considerations. Thus, any performance, including suboptimal ones, is possible within that cap. Decoupling power management from performance management risks missing performance goals or energy goals or both.

Finally, we hope and believe that the concepts and models presented in this dissertation will help improve the energy efficiency of future data centers, that in turn will lower energy needs and associated environmental impacts, increase computational capacity within existing budgets, and promote job growth through improved profit margins.

2 ENERGY EFFICIENCY IDEALS AND THE IRON LAW

> *Energy efficiency is the new fundamental limiter of processor performance, way beyond numbers of processors.*
>
> — Shekhar Borkar and Andrew A. Chien [35]

2.1 Overview

Energy efficiency is the work done per unit amount of energy consumed. Maximizing energy efficiency is important as it allows more work to be done for a given energy budget and also allows work to be done faster for a given power budget. This has economic and environmental benefits as it minimizes the energy needed to do a given computation.

While compute capability, in terms of the number of transistors per chip, has steadily increased (Moore's Law [158]), operating voltage has not reduced in proportion (limited Dennard scaling [63]). Borkar and Chien [35] observed that although Pollack's rule [170] predicts a speedup potential (beyond speedups in transistor switching) in proportion to the square root of the number of transistors in a processor, energy-efficiency concerns discourage many microarchitectural techniques that can enable those performance gains. Thus, improving energy efficiency will also improve processor performance.

Energy-proportional computing, that uses energy in proportion to the work done, is an important concept for energy-efficient systems since it seeks to eliminate energy waste by only using as much energy as the work done. However, modern computers are not energy proportional. For example, they use non-trivial power when they are powered on but not used. This is due to processor leakage power, DRAM refresh, and power draw

by various components such as fans, hard disks, etc. Energy proportionality has been an useful goal for system designers to make their systems more energy-efficient.

One way to increase proportionality could be to use innovative power-delivery solutions. PowerNap [151] proposed a new power delivery system called RAILS that reduces idle power consumption and proposed to rapidly transition the system to a nap (sleep) state. The nap state retains volatile information, e.g., memory state. With an expected transition time of 10ms or less, the system should be able to save power during idle periods of short durations. The RAILS system consists of multiple power supplies to improve upon the low efficiency of individual power supply units. The RAILS supplies are provisioned such that the power consumption of the idle system is in the efficient operating range of a single supply. As server blades become active, more RAILS supplies get electrically connected so that all the supplies operate in their efficient ranges. However, the PowerNap approach does not work well for some workloads, e.g., OLDI workloads, because full system idleness is relatively rare [152]. Moreover, the napping opportunity decreases further as the number of cores increase [154].

Modern computers also have reconfigurable resources, e.g., processor voltage and frequency levels. We show that intelligent reconfiguration can cause these computers to exceed the efficiency of conventional energy-proportional machines when they are performing work. The original definition of energy-proportional computing, first proposed by Barroso and Hölzle, does not characterize the energy efficiency of recent reconfigurable computers, resulting in non-intuitive "super-proportional" behavior (more work done in proportion to the energy used). This chapter introduces a new definition of "ideal" energy-proportional computing and new metrics to help guide both system architects and operators to configure systems to operate close to this ideal efficiency.

We show that the traditional ideal of energy-proportional may be significantly energy

inefficient, and hence, not suited to be an ideal model. Instead, we propose Energy Optimal Proportional (EOP) as the new ideal model for system designers. An ideal EOP system has the maximum efficiency over its entire performance range. Making systems more EOP is a design goal for system architects.

Currently, real systems are not EOP any more than earlier systems were EP. We propose Dynamic EO (Dynamic Energy Optimal), which is the power-performance Pareto frontier and which can be realized on the current system, as the new ideal model for system operators. The Pareto frontier is a set of Pareto-optimal configurations. A system configuration is Pareto optimal if it is not possible to reconfigure the system to improve performance without also increasing power consumption or to reduce power consumption without also degrading performance. Pareto-optimal power-performance system configurations help enforce service-level objectives such as maximizing performance for a given power budget or minimizing power for a given performance target, both leading to energy savings. System operators should aim for Dynamic EO to achieve power-efficient performance for the current system.

This chapter focuses on defining new ideals for energy proportional computing and new metrics to quantify operational energy wastage of computing systems.
The main contributions of this chapter are:

1. We show that the conventional "ideal" model of energy proportionality does not fully describe the energy efficiency potential of modern super-proportional systems.

2. We propose new ideals for both system designers and system operators. EOP is the new design ideal that subsumes conventional "ideal" energy proportionality. Dynamic Energy Optimal (Dynamic EO), that is the power-performance Pareto frontier, is the new operational ideal.

3. We propose a new metric called Computational Power Usage Effectiveness (CPUE) to quantify excess computational energy used with respect to that by EOP.

4. We propose new metrics, Load Usage Effectiveness (LUE) and Resource Usage Effectiveness (RUE), that can help system operators to focus on load management and configuration management to make the system operate efficiently.

5. We develop the "Iron Law of Energy" that quantifies the impact of poor load management and poor configuration management on CPUE.

Section 2.2 defines energy efficiency and describes our experimental setup. Section 2.3 shows why the conventional "ideal" model is inadequate for modern systems. Section 2.4 proposes our new design and operational ideals. Section 2.5 discusses several properties of the power-performance Pareto frontier and their implications on managing for efficient operations. Section 2.6 proposes a new metric for quantifying energy waste and its decomposition into two components, pertaining to load management and configuration management. Section 2.8 describes how Pareto frontiers of individual systems can be composed to determine the Pareto frontier for a collection of systems. Section 2.7 discusses some of the overheads and challenges involved in energy-efficient scheduling.

2.2 Terminology and Infrastructure

Similar to Barroso and Hölzle [22, 104], we define energy efficiency as $\frac{\text{Work}}{\text{Energy}}$, or equivalently, $\frac{\text{Performance}}{\text{Power}}$. The performance of a system is measured as the rate of doing work, e.g., the load serviced, or transactions completed per unit time. Performance normalized to that at peak load levels is the system utilization [22].

The system that we use in this work is a single-socket quad-core Haswell-based Xeon E3-1275 v3 server with 32 GB memory (DDR3-1600), henceforth referred to as HS. HS

runs RHEL with kernel version 2.6.32. It has a frequency range of 0.8–3.9 GHz with 3.5^{+}–3.9 GHz being the turbo boost region. The turbo boost plan is 2/3/4/4 meaning that the maximum frequency can be 3.5 + 0.1*2 = 3.7 GHz with all four cores active, 3.5 + 0.1*3 = 3.8 GHz with three cores active, and 3.5 + 0.1*4 = 3.9 GHz with two or one cores active. We run the system with all four cores, hyperthreading (2 hardware threads per core, that is, 8 hardware threads per socket), and cache prefetching enabled by default. All cores run at the same frequency (except perhaps in turbo mode where individual cores may be throttled differently). The socket frequency can changed in steps of 100 MHz by writing to Model Specific Registers. Any value for the turbo region implies a limit on the maximum frequency. HS has a socket TDP of 84W and a remarkably low socket power of ~0.27W when idle. DRAM idle power is ~4.3W.

We use the SPECpower benchmark [205] in this chapter. This Java workload simulates warehouse transaction processing, with (by default) as many warehouses as logical processors on the system under test, that is, the server. Transaction requests to each warehouse arrive in batches of 1000 transactions each. The batches have (negative) exponentially distributed interarrival times. The server load is measured in total transactions per second. The workload first calibrates the maximum, or 100%, load. Next, it does measurement intervals by varying the load offered to the system under test from 100% (max. utilization) to 0% (no utilization) in decrements of 10%. In these intervals, the load served must be within 2% (up to 2.5% shortfall for the 100% and 90% intervals is allowed) of the offered load. We use a Watts Up? (.net) meter [107] for system (wall) power measurements. SPECpower uses its own software utility (daemon) for periodically measuring and reporting system power. SPECpower reports power numbers only for the measurement intervals. This is what is plotted against performance in all the graphs for SPECpower profiles.

We refer to 100% load as the maximum load achieved for the Peak Performance Configuration (all cores at the highest frequency and prefetching enabled). All loads are normalized with respect to that peak load.

2.3 Inadequacy of Conventional Energy Efficiency Ideals

> *We see that peak energy efficiency occurs at peak utilization and drops quickly as utilization decreases.*
>
> — Luiz André Barroso and Urs Hölzle [22]
>
> *The average efficiency is always less than the peak efficiency; modern servers are only maximally efficient at 100%.*
>
> — David Meisner and Thomas F. Wenisch [153]

Barroso and Hölzle observed that real systems—at that time—attain peak efficiency at peak utilization, but quickly lose efficiency as utilization drops as they are unable to proportionately reduce power consumption. They posit that an "ideal" energy-proportional system should always use energy in proportion to the work done, by maintaining this peak efficiency even at reduced load.

Figures 2.1 and 2.2 illustrate this original model for HS running SPECpower. Figure 2.1 shows the server's power-performance profile at different load levels with the highest processor frequency. We label these points with *Peak Performance Configuration*) since the machine can serve maximum load (peak performance) with this configuration.

The *EP* line represents Barroso and Hölzle's "ideal" energy-proportional profile where performance is linearly proportional to power. We consider this a *design ideal* for future systems, since current systems have unavoidable idle power consumption. The *Dynamic EP* line accounts for idle power [141], and represents an *operational ideal* for the current

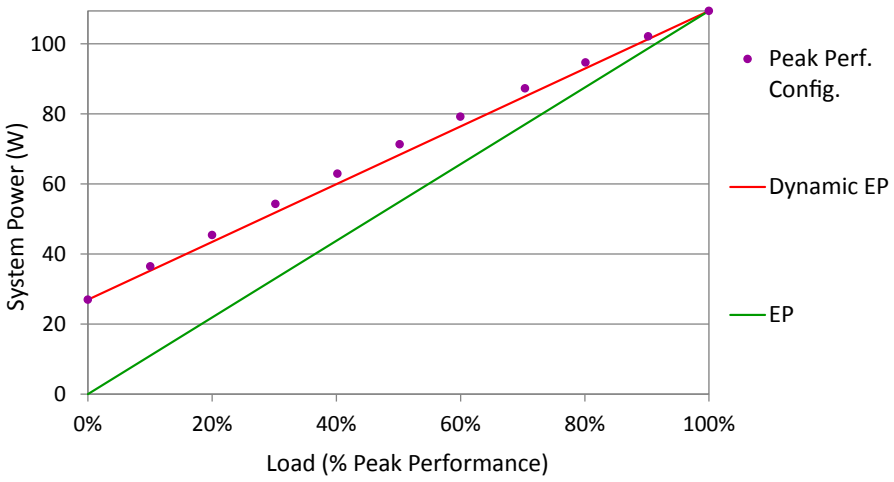


Figure 2.1: Power-Performance profile with conventional server configuration.

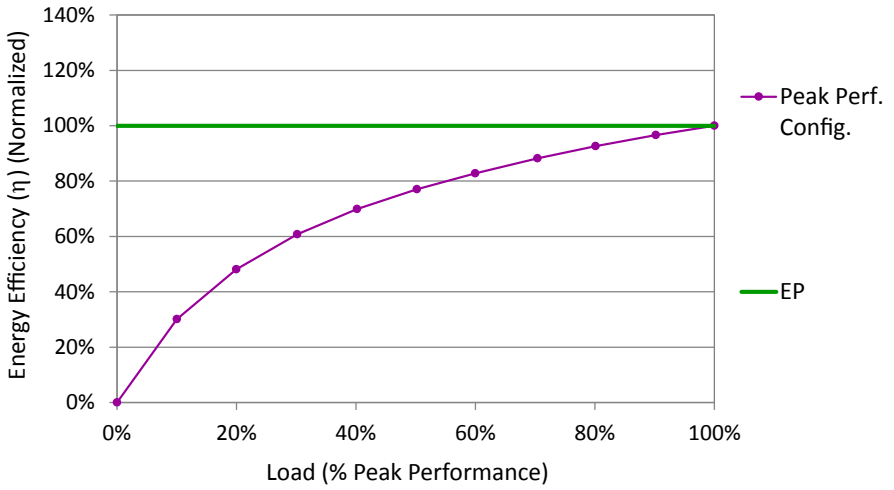


Figure 2.2: Conventional efficiency model of servers.

system. This server's Peak Performance Configuration achieves power-performance very close to Dynamic EP. Figure 2.2 shows that the corresponding energy efficiency (η), normalized to that at peak performance, reduces quickly from 100% as performance drops. In contrast, an EP system is always 100% efficient.

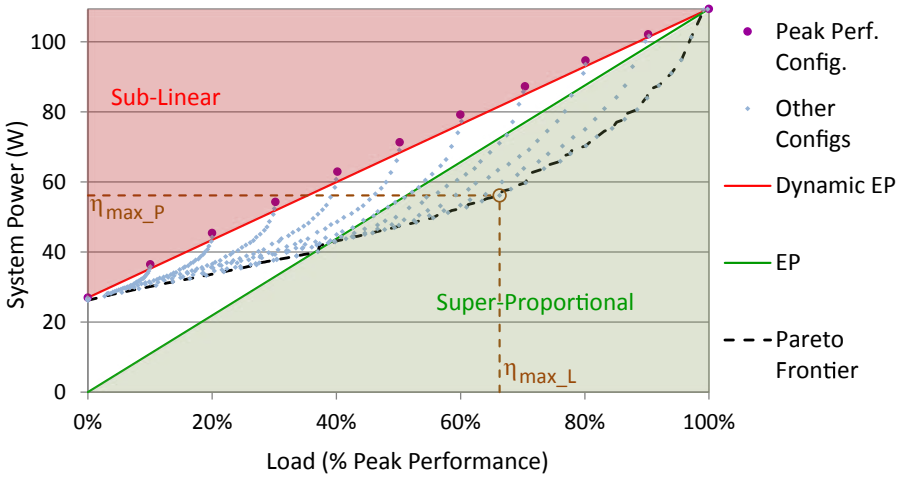


Figure 2.3: Power-Performance profile for super-proportional systems.

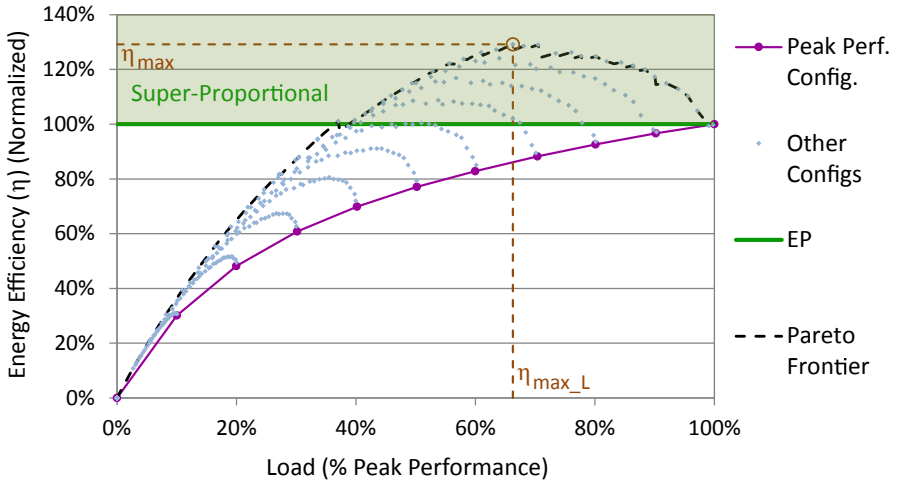


Figure 2.4: Performance (Load) vs Efficiency for super-proportional systems.

Barroso and Hölzle's observation has been instrumental in helping drive recent system designs to have lower idle power and a wide dynamic power range. However, their model describes systems with *fixed resources*, while these modern, more-efficient processors have *reconfigurable resources*—e.g., core frequencies, voltages, number of active cores, threads

per core, etc. that can be varied at runtime.

Operating with fixed resources can be inefficient when a server faces variable loads, either due to fluctuating demands, or service consolidation and load balancing among other servers [47, 60, 148].

Servers are usually configured for maximum performance (that is, the Peak Performance Configuration), but other configurations can trade performance for greater energy efficiency. Figure 2.3 shows that changing just the socket frequency (and consequently voltage) results in energy efficiency that exceeds the "ideal" EP profile. Specifically, by varying the frequency from 3.9 to 0.8 GHz, the Haswell server can achieve super-proportional efficiency over almost 60% of the performance range (points in the shaded Super-Proportional region—where performance is super-proportional to power). Figure 2.4 shows that the maximum efficiency (η_{max}, occurring at approximately two-thirds load) is 29% higher relative to the EP energy efficiency, for this server.

Reconfigurable systems create opportunities for increased efficiency even outside the super-proportional region. For example, Figure 2.4 shows that the Peak Performance Configuration attains a relative efficiency of 61% at 30% load, while a different configuration achieves a relative efficiency of 88% at the same load. In other words, the usual server configuration uses 44% more energy than necessary to satisfy the same load, despite being nearly on the Dynamic EP line.

> *Ideally systems would exhibit energy-proportionality, wherein servers consume power in proportion to their load.*
>
> — David Meisner et al. [152]
>
> *In an energy-proportional system, explicit power management is unnecessary, as power consumption varies naturally with utilization.*
>
> — David Meisner et al. [151]
>
> *energy-proportional computing must be the ultimate goal for both hardware architecture and software-application design.*
>
> — Shekhar Borkar and Andrew A. Chien [35]

As we have demonstrated, neither EP nor Dynamic EP (that is, the conventional ideal models) describes the full potential of modern computing systems. While non-linearity with reconfiguration is well-known, e.g., with frequency (and voltage) control, the existing ideal models do not consider its impact on peak efficiency. New models are needed to aid operating system schedulers and system administrators to configure systems to deliver maximum efficiency.

2.4 Redefining EP and Dynamic EP

The EP model assumes that maximum energy efficiency occurs at maximum (100%) load and argues that an ideal system should achieve that efficiency for all loads. Yet Figure 2.4 shows that a reconfigurable server actually attains maximum efficiency (η_{max}) at a lower load ($\eta_{max\_L} < 100\%$). We argue that a better ideal model is one that achieves this optimal efficiency η_{max} for all loads.

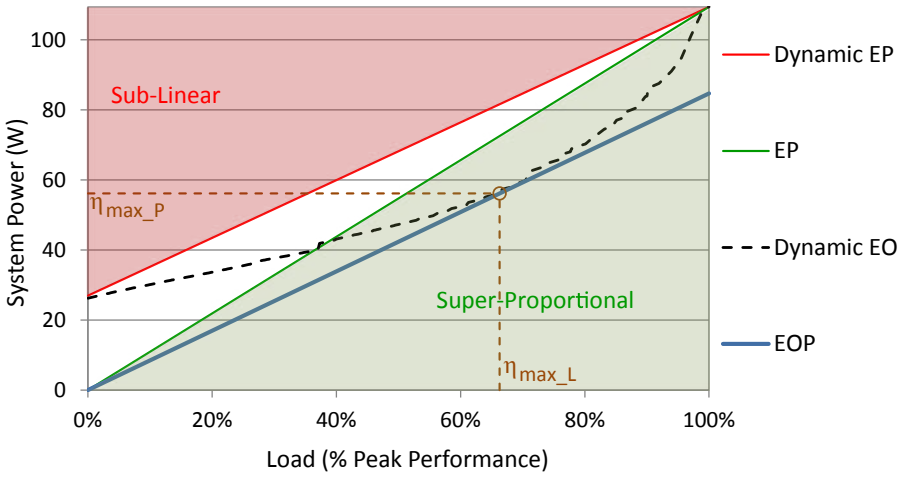


Figure 2.5: EOP and Dynamic EO models.

Similar to the EP model, the ideal system should have maximum efficiency (η_{max}) at every load. This implies that for a given computation, it will use minimum energy (E_{min}) to do it irrespective of the computing rate (performance or load). Figure 2.5 shows its geometric interpretation as a straight line passing through the points (0, 0) and (η_{max_L}, η_{max_P}). This ideal system, that is energy optimal at every load, uses power linearly proportional to load (l/η_{max} power at load l). Energy optimality at every load implies energy proportionality, but the converse is not true, e.g., EP is proportional but not optimal at all loads.

We call this new model *EOP* (Energy Optimal Proportional) since it is both optimal and proportional. EOP is a *design ideal* that gives system designers a way to measure how far the energy efficiency of a target design differs from the best possible design, hopefully leading to more energy-efficient systems. EOP subsumes the EP model for all systems—it improves upon EP for super-proportional systems and is identical to it for all others.

Of course real systems are unlikely to achieve this design ideal, e.g., due to unavoidable idle power, so system software needs an operational model that characterizes the maximum efficiency that can be realized by the current system at different loads. We address this using the well-known power-performance *Pareto frontier* [19, 27, 183], shown as a dashed line in Figures 2.3–2.6. The Pareto frontier represents configurations in the current system that use the lowest power, and hence are the most efficient, among all configurations that can serve a given load. These configurations are Pareto optimal in the sense that, among these configurations, one cannot reduce power without also reducing load or increase load without also increasing power.

We call this model *Dynamic EO*. Like Dynamic EP, it is an operational ideal that seeks to characterize the best energy efficiency that can be achieved for a given system. But it differs from Dynamic EP in two aspects—it characterizes optimality that can already be realized by some among the multitude of configurations in the current system and it does not assume linearity of the power-performance profile.

Figure 2.5 illustrates the different models. These are the

- *design ideals*: conventional (EP), new (EOP), and
- *operational ideals*: conventional (Dynamic EP), new (Dynamic EO).

The EOP line meets (is tangential to) the Dynamic EO line only at points having the maximum efficiency (η_{max}).The following energy efficiency relations hold for any system:

$$\text{Dynamic EP} \leqslant \text{EP} \leqslant \text{EOP}$$

$$\text{Dynamic EO} \leqslant \text{EOP}$$

where $\leqslant$ means less than or equal to for values of efficiency. Systems, like our server, that can operate in the non-Sub-Linear region for any portion of their performance range have Dynamic EP $\leqslant$ Dynamic EO for all such loads.

2.5 Power-Performance Pareto Frontier (Dynamic EO)

In this Section we describe some properties of Dynamic EO and their implications for optimal system operations.

Every configuration of the system can be characterized by its performance and power consumption. We call each such (Configuration, Performance, Power) tuple a system state. The Pareto frontier is determined by only those states that use the lowest power among all states having at least that performance. It is a subset of the set of system states. The governors that we develop in Chapters 3 and 5 seek to constrain system operations to Pareto-optimal states.

Let Π denote the set of system states with Π_i representing the i^{th} state having performance $\Pi_i.Perf$ and power consumption $\Pi_i.Power$. Let the highest performing state be Π_0. We apply the well-known concepts of Pareto dominance and Pareto optimality. State Π_i Pareto-dominates state Π_j if $(\Pi_i.Perf \geqslant \Pi_j.Perf) \wedge (\Pi_i.Power \leqslant \Pi_j.Power)$.

Property 1: *The Pareto frontier is the set of non-dominated states*.

In Figures 2.3 and 2.4, the Pareto frontier is the set of states represented by the dashed line. The states that lie on the EP line in the Super-Proportional region are dominated by the states on the frontier.

Implication: Constraining system operation to the Pareto frontier is important since dominated states are less efficient than dominating states (also see Figure 2.4). The state with the maximum efficiency (η_{max}) lies on the Pareto frontier.

Property 2: *States on the Pareto frontier have the same total order in both power and performance*.

Let Π_i, Π_j be states on the Pareto frontier. Then $(\Pi_i.Perf > \Pi_j.Perf) \iff (\Pi_i.Power > \Pi_j.Power)$. We number the states in decreasing order of performance. The ordering relation for states on the frontier is thus: $i < j \iff (\Pi_i.Perf > \Pi_j.Perf) \wedge (\Pi_i.Power >$

$\Pi_j.\mathrm{Power}$).

Implication: While the state space is inherently two-dimensional, the Pareto frontier is more constrained allowing system operators to qualitatively reason about the other dimension from looking at one dimension alone. For example, increasing the power budget *will* improve performance at the Pareto frontier if the power is used. This is not true for the whole state space where states with less performance can use more power. This positive correlation between the two dimensions exists at the Pareto frontier.

Property 3: *System states that optimize power-performance metrics are located at the Pareto frontier*.

Consider a state Π_i that is not on the frontier. So there exists at least one other state Π_j such that $\Pi_i.\mathrm{Perf} \geqslant \Pi_j.\mathrm{Perf}$ and $\Pi_i.\mathrm{Power} \leqslant \Pi_j.\mathrm{Power}$ with at least one of the inequalities being strict. This implies that the highest performing state with/without a (maximum) power cap and the lowest power state with/without a (minimum) performance bound lie on the Pareto frontier.

In this work we assume that $\mathrm{performance} \propto \mathrm{delay}^{-1}$. Since energy is power multiplied by time (delay), it implies that the lowest energy point with/without a delay cap must lie on the Pareto frontier. Since the state corresponding to the highest performance-per-watt is the same as the state with the lowest energy, that state will also be on the Pareto frontier. Moreover, according to the above condition, states corresponding to the minimum energy-delay (ED) product or ED^2 product or, in fact, any $\mathrm{ED}^n, n \geqslant 0$ must also lie on the Pareto frontier.

Since states on the Pareto frontier are more efficient than other states, the highest performing state with/without a maximum power cap, the lowest power state with/without a minimum performance bound, the highest performance-per-watt state, the lowest energy state, the lowest energy-delay state, etc. will lie on the Pareto frontier.

Implication: Optimizing system operations for commonly used power-performance or energy efficiency metric necessitates operating it at the Pareto frontier.

Property 4: *The points of contact between the frontier,* $\mathrm{Power} = f(\mathrm{Perf})$*, and the tangent curve* $\mathrm{Power} = c_n(\mathrm{Perf})^{n+1}, n+1 \geqslant 0$ *and some constant* c_n*, represent configurations that optimize (minimize) metric* ED^n*. (*$n = 0$ *means energy E.)*

Let Π_i be a state that optimizes (minimizes) metric ED^n. By Property 3, Π_i must be on the frontier. Since $E = \mathrm{Power}(\mathrm{Perf})^{-1}$ and $\mathrm{ED}^n = \mathrm{Power}(\mathrm{Perf})^{-n-1}$, Π_i will be on the curve for the power function $\mathrm{Power} = c_n(\mathrm{Perf})^{n+1}$ if we choose $c_n = \Pi_i.\mathrm{Power}(\Pi_i.\mathrm{Perf})^{-n-1}$. c_n is thus the optimum value for ED^n. Moreover, every point on this power function curve will have the same value for ED^n, which is c_n. No part of the frontier can be below this curve, as then states on this part of the frontier will have lower power for the same performance compared to points on the power function curve directly above them and thus have a smaller value for ED^n than c_n which is a contradiction.

Note that all points on the curve above the linear tangent are suboptimal with respect to E, all points above the quadratic tangent are suboptimal with respect to ED, all points above the cubic tangent are suboptimal with respect to ED^2, and so on.

Implication: This forms the basis for the geometric interpretation of the Pareto Proportional line described in Section 2.4. Every point on the linear tangent has the same slope, which is equal to $\frac{\text{Power}}{\text{Performance}}$, that is, performance-per-watt^{-1} value of the most energy-efficient point.

Property 5: *The Pareto frontier is not necessarily convex (or concave).*

Let Π_i, Π_j, Π_k be states on the frontier with $i < j < k$. The ordering relations only imply $\Pi_i.\mathrm{Perf} > \Pi_j.\mathrm{Perf} > \Pi_k.\mathrm{Perf}$ and $\Pi_i.\mathrm{Power} > \Pi_j.\mathrm{Power} > \Pi_k.\mathrm{Power}$, not $\Pi_j.\mathrm{Power} \leqslant \Pi_k.\mathrm{Power} + \left(\frac{\Pi_j.\mathrm{Perf} - \Pi_k.\mathrm{Perf}}{\Pi_i.\mathrm{Perf} - \Pi_k.\mathrm{Perf}}\right)(\Pi_i.\mathrm{Power} - \Pi_k.\mathrm{Power})$.

Implication: Convex optimization approaches cannot be directly applied while composing multiple Pareto frontiers. Moreover, hill-climbing based search techniques at the frontier can get stuck in local optima instead of reaching global optima. However, as we show in Section 2.8, convex (polynomial) approximations to the Pareto frontier may work well enabling applications of efficient optimization techniques.

2.6 Computational PUE

Datacenters can satisfy a given load by distributing it to machines in different ways. Each machine can also be configured in a large number of ways. These modes for servicing the load differ in the amount of energy consumed, since some modes are more inefficient than others.

A hypothetical ideal system, that is, one that meets the design ideal EOP, achieves maximal energy efficiency (η_{max}) and thus minimizes the energy (E_{min}) needed for a given computation regardless of load. We would like a metric to quantify the excess energy used by a real system, compared to this ideal system.

Our new metric, *Computational Power Usage Effectiveness* (or, CPUE), measures how much energy a server uses with configuration c at load l compared to the energy used by EOP. We define

$$\text{CPUE}(c, l) = \frac{\text{Actual server energy with } c \text{ at } l}{\text{EOP energy at } l}, \qquad l > 0 \tag{2.1}$$

$$= \frac{E(c, l)}{E_{min}}, \qquad l > 0 \tag{2.2}$$

$$\text{Thus, } E(c, l) = \text{CPUE}(c, l) \times E_{min}, \qquad l > 0 \tag{2.3}$$

CPUE(c, l) is inspired by the well-known PUE metric [16] that tracks energy waste for datacenters by taking the ratio of facility energy consumption to energy consumption by IT equipment. PUE > 1 quantifies excess relative energy used by the datacenter due to the non-IT infrastructure. Similarly, CPUE$(c, l) > 1$ quantifies excess relative computational energy used whenever efficiency drops below η_{max}.

We have seen that there are two major factors that lead to energy inefficiencies: i) running the system at a non-optimal load and ii) for a given load, running the system with a non-optimal configuration. We can decompose CPUE(c, l) to isolate these two factors.

We defined CPUE(c, l) as $E(c, l)/E_{min}$. For a given amount of work, energy consumed is inversely proportional to efficiency. Thus,

$$\text{CPUE}(c, l) = \frac{\eta_{max}}{\eta(c, l)}, \qquad l > 0 \tag{2.4}$$

$$= \left(\frac{\eta_{max}}{\eta_{Pareto}(l)}\right) \times \left(\frac{\eta_{Pareto}(l)}{\eta(c, l)}\right), \qquad l > 0 \tag{2.5}$$

$$= \text{LUE}(l) \times \text{RUE}(c, l), \qquad l > 0 \tag{2.6}$$

$$\text{Thus, } E(c, l) = \text{LUE}(l) \times \text{RUE}(c, l) \times E_{min}, \qquad l > 0 \tag{2.7}$$

where LUE(l) denotes *Load Usage Effectiveness* at load l and RUE(c, l) denotes *Resource Usage Effectiveness* of configuration c and load l.

LUE(l) is the efficiency of EOP(η_{max}) relative to that of of Dynamic EO at load l. LUE$(l) \geqslant 1$ with LUE$(l) = 1 \iff l$ can be served at maximum efficiency (η_{max}). Since energy consumed is inversely proportional to efficiency, LUE$(l) > 1$ quantifies excess energy used, relative to E_{min}, due to non-optimal loads assuming that the Pareto-optimal configuration has been chosen to serve load l.

RUE(c, l) is the efficiency of Dynamic EO relative to that of configuration c, both

at load l. $\text{RUE}(c,l) \geqslant 1$ with $\text{RUE}(c,l) = 1 \iff c$ is a Pareto-optimal configuration. $\text{RUE}(c,l) > 1$ quantifies excess energy used, relative to Dynamic EO at load l, due to using non-optimal (Pareto-dominated) configuration c for serving load l.

Inspired by the "Iron Law of Performance", we call Equation 2.7 the "Iron Law of Energy". System designers will focus on minimizing E_{min} whereas system operators will focus on minimizing LUE and RUE.

Both $\text{LUE}(l)$ and $\text{RUE}(c,l)$ can be expressed in terms of $\text{CPUE}(c,l)$. Since $\text{RUE}_{\text{Pareto}}(l) = 1$ for every l, $\text{LUE}(l) = \text{CPUE}_{\text{Pareto}}(l)$ and $\text{RUE}(c,l) = \text{CPUE}(c,l)/\text{CPUE}_{\text{Pareto}}(l)$.

Our proposed RUE and LUE metrics can help system operators isolate the sources of energy inefficiency and guide new policies to reduce it. LUE is important for load management of Pareto-optimal configurations. RUE is important for configuration management for Pareto-dominated configurations. While LUE is applicable to all systems, both old and new, it only partially quantifies energy waste in reconfigurable systems that can be configured in a plurality of ways. RUE completes the quantification.

2.7 Load and Configuration Management

Most data centers are provisioned to meet peak load, but normally operate at much lower load levels. The LUE metric can help operators quantify the potential benefit of deploying load management policies [47, 60, 148], e.g., concentrating load on some servers and shutting down others. Of course, any such policy must also ensure that service-level agreements are still satisfied [171].

Figure 2.6 shows that CPUE for the Peak Performance Configuration is always > 1 (wastes energy) and increases as load decreases. The best CPUE for this configuration is 1.29, occurs at peak load, and implies 29% excess energy used relative to E_{min}. LUE (that is, CPUE for Dynamic EO), on the other hand, first decreases to 1, then increases, revealing

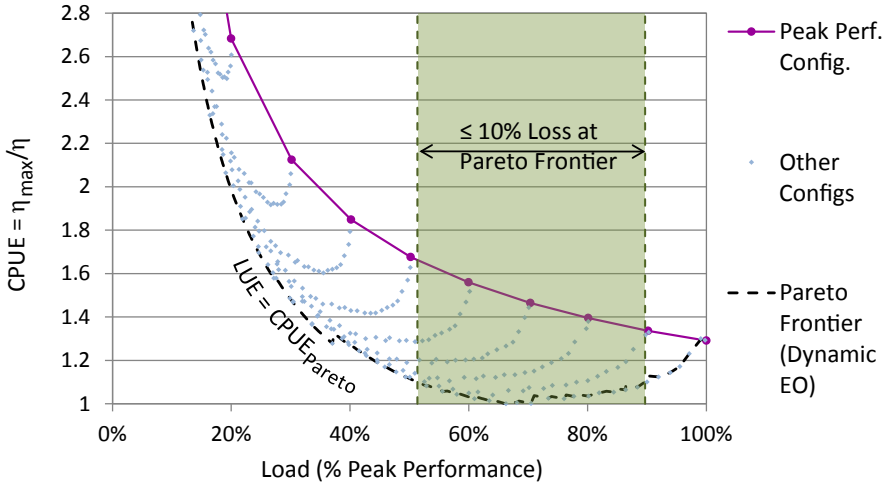


Figure 2.6: $\mathrm{CPUE}(c, l)$ and $\mathrm{LUE}(l)$. These $\to \infty$ as load $l \to 0$ due to non-zero idle power. For any configuration c and load l, $\mathrm{CPUE}(c, l) \geqslant \mathrm{CPUE}_{\mathrm{Pareto}}(l) = \mathrm{LUE}(l) \geqslant 1$.

a sweet spot of $\leqslant 10\%$ excess energy used at around 51%–90% of peak performance.

Barroso and Hölzle [22] observed that servers typically operate at 10%–50% load. The LUE curve for SPECpower (Figure 2.6), shows excess energy used due to suboptimal load of approximately 10% at the higher end of this range, to over 250% (not shown) at the lower end. The steep slope of the LUE curve at low loads makes even modest load management very attractive. For example, increasing load from 10% to 20% of peak reduces LUE from 3.55 (255% excess) to 1.99 (99% excess) and a further increase to 25% peak load reduces LUE to 1.68 (68% excess).

Even in a data center with perfect load balancing, reconfigurable servers may be misconfigured, wasting significant energy even at optimal load. Figure 2.7 shows RUE for SPECpower for all system configurations and loads. Operating with the Peak Performance Configuration is significantly wasteful even at low loads, e.g., 21% excess energy used at 10% load compared to operating at Dynamic EO. The excess increases to 51% before decreasing to zero at peak load. Not all Pareto-dominated configurations are as wasteful—

the shaded band identifies configurations that have an RUE of $\leqslant 1.1$ and hence limit the extra energy used to 10%.

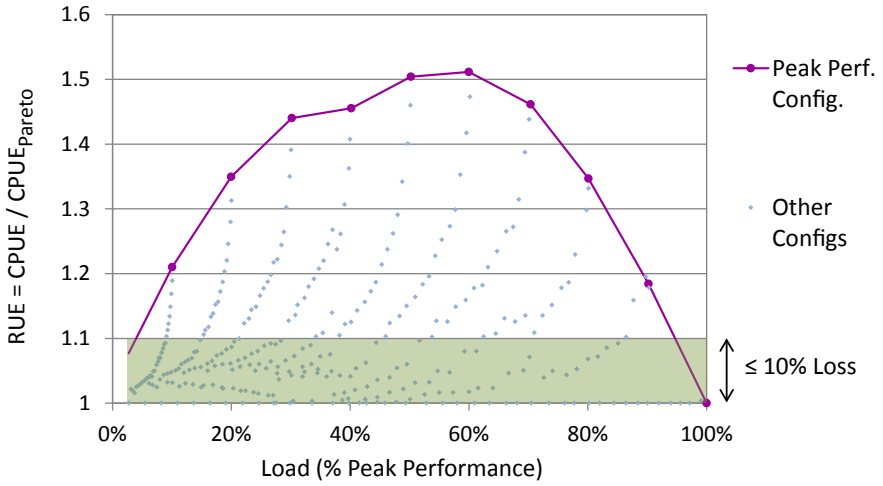


Figure 2.7: Resource Usage Effectiveness.

Configuration management (to reduce RUE) may incur costs, e.g., due to transition times while changing configurations. System designers are making great strides in reducing these costs. For example, processor frequency transitions complete within a few hundred microseconds today.

Calculating LUE and RUE (as well as determining EOP and Dynamic EO) requires knowledge of the Pareto frontier. In this chapter, we determine the frontier offline by running the workload multiple times with the server configured to different frequencies. Offline characterization is also used in prior work [19, 183], but may not be feasible in an online setting with unknown workloads. In Chapter 3 we introduce an online policy that closely approximates the frontier by controlling processor frequency and cache prefetching.

Workload characterization incurs overheads, but researchers have demonstrated [60,

148] its feasibility and utility in large-scale computing environments, e.g., at Google datacenters and Amazon EC2. With modern systems showing trends of increasingly making components reconfigurable, we expect further applications of such techniques to infer characteristics that are relevant to these components.

2.8 The Π-dashboard

With one or more reconfiguration knobs in the system, the user is faced with the daunting task of choosing the right configuration that meets a desired power-performance or energy-efficiency criteria. The Π-dashboard attempts to bridge the gap between high-level power-performance goals and system resource configurations—a mapping capability that is largely missing in today's systems. Π-dashboards enable selection of a variety of power-performance profiles for the system. The user or operating system can select a desired power-performance profile from the Π-dashboard resulting in a "one-shot" transition of the system to the corresponding configuration.

As discussed in Section 2.5, we denote the collection of system states as Π-states. Each state is characterized by the performance and power consumption of the system when operating with that configuration. Pareto-optimal Π-states can be totally ordered (Section 2.5, Property 2). The Π-dashboard is a tabular representation of this totally ordered list of Pareto-optimal Π-states.

Chapter 3 shows how we use power-performance predictors, using hardware counters, to characterize the expected impact of different configurations and subsequently identify Pareto-optimal configurations. A controller/coordinator creates the dashboard from the predictions and interfaces with the user or operating system. It updates the dashboard periodically as execution profiles change over time. The coordinator may be implemented as a software routine (ISR) that runs on one or more cores, or as a specialized unit such

as a PCU in modern systems [181].

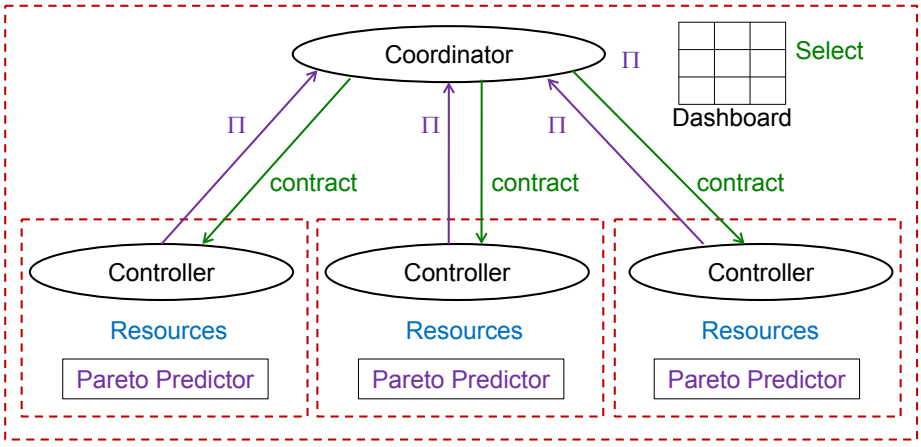


Figure 2.8: Coordination architecture.

Figure 2.8 shows a schematic overview of hierarchical coordination across multiple systems. The Pareto Predictors of individual systems predict power-performance profiles of the reconfigurable Resources of their systems. The local controller/coordinator communicates the Pareto-optimal Π-states to the upper-level coordinator. This coordinator composes the Pareto frontiers to get the overall Pareto frontier and exposes the Π-dashboard to the user. The configurations for the selected profile are then communicated back to the local controllers/coordinators as a "contract" that should be honored by the individual systems for subsequent execution.

The coordinator can compose Pareto frontiers using an optimization program. Let there be n machines, numbered 1..n. Let $x_{i,j}$ and $y_{i,j}$ respectively denote the performance and power consumption in the j^{th} Pareto-optimal state, Π_j, in machine i. The overall performance range that can be supported is $[\min_{i,j}(x_{i,j}), \sum_{i=1}^{n} x_{i,0}]$. The optimal power consumption, p, for any performance l in this range can be determined by solving the following program:

Model	R^2
Linear	0.9138
Quadratic	0.9813
Cubic	0.9959
Quartic	0.9988
Quintic	0.999

Table 2.1: R^2 values for polynomial fits to SPECpower Pareto frontier.

$$\text{minimize } p = \sum_{i=1}^{n} \sum_{j} I_{i,j} * y_{i,j} \tag{2.8}$$

$$\text{such that } l \leqslant \sum_{i=1}^{n} \sum_{j} I_{i,j} * x_{i,j} \tag{2.9}$$

$$I_{i,j} \in \{0, 1\} \quad \forall i, j \tag{2.10}$$

$$\sum_{j} I_{i,j} = 1 \quad \forall i \tag{2.11}$$

In the above, condition 2.9 requires that the desired performance be met, condition 2.10 allows any state to be either fully selected or not selected, and condition 2.11 requires exactly one state to be selected per system.

In general, the Pareto frontier is not convex (Section 2.5, Property 5). So, a local optima in an optimization program dealing with Pareto frontiers is not necessarily a global optima. But for many real systems, convex (e.g., polynomial) approximations may work well. Table 2.1 shows the coefficient of determination values (R^2 values, best fit value=1) for several polynomial fits to the SPECpower Pareto frontier on HS. A quadratic or higher order approximation works quite well. The approximation errors may be higher if the models are required to include specific points. Convex approximations may reduce the computational effort required to solve the optimization program.

2.9 Conclusion

In this chapter, we explored the relation between two well-known but dissimilar concepts, power-performance Pareto optimality and energy proportionality, both of which share the end goal of making computing more energy efficient. We demonstrated that the conventional model of energy proportionality is inadequate for reconfigurable systems since it does not guarantee energy optimality. We defined a new model, EOP, that guarantees both optimality and proportionality and established its relation to the Pareto frontier.

Real systems are not ideal and hence use more energy than that used by the ideal EOP system (E_{min}). We proposed a new metric, Computational PUE (CPUE), that quantifies how much excess computational energy is used by the system relative to that by EOP. This depends on both the load served and the system configuration used to serve that load.

Our new Iron Law of Energy shows that CPUE can be decomposed into three terms—LUE, RUE, and E_{min}. The LUE and RUE metrics separate the load and configuration aspects of suboptimality. LUE answers the question: how suboptimal is a given load? RUE answers the question: how suboptimal is a given configuration with respect to the most efficient configuration that can also serve that load? LUE is affected by demand fluctuations and inter-server load management whereas RUE is affected by intra-server configuration management.

While system components are increasingly being designed to be reconfigurable, identifying the Pareto frontier is challenging, particularly with multiple reconfigurable resources and dynamically changing runtime environments. Scheduling frameworks that carefully choose configurations and operating ranges will unlock the full potential of current and future reconfigurable systems. This will be our focus in Chapter 3.

3 PARETO GOVERNORS

3.1 Overview

In this chapter we develop new operating system governors (resource managers) that seek to operate the system at or close to Dynamic EO (power-performance Pareto frontier) so that Service-Level Agreements (SLAs) are satisfied. We call such governors *Pareto governors*.

We consider the following SLAs in this chapter. (See Chapter 1, Section 1.2, for more background on these SLAs.)

- **SLAee**: Maximize energy efficiency.
- **SLApower**: Maximize performance given a power cap/budget.
- **SLAperf**: Maximize power savings given a performance target.

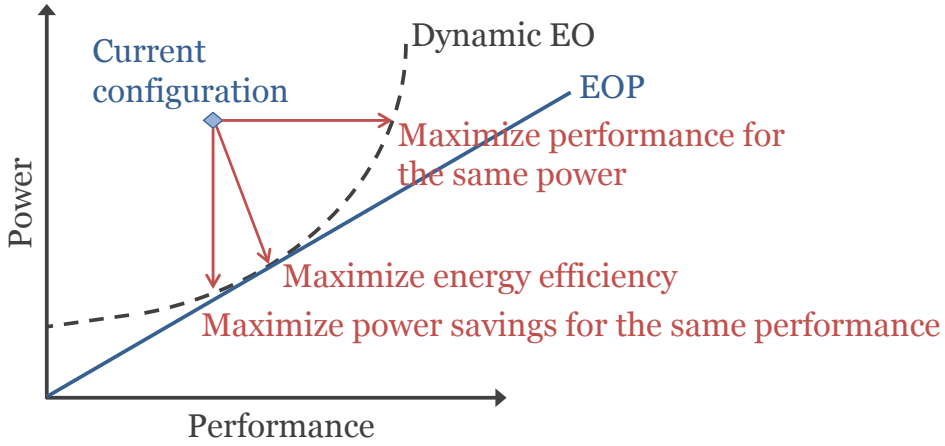


Figure 3.1: State transitions to Dynamic EO for meeting SLAs.

Figure 3.1 shows what transitions the Pareto governors must make to the current operating point to meet various SLAs.

Our governors manipulate two dynamic reconfiguration knobs—processor frequency (and voltage) scaling and cache prefetching. Processor caches improve performance by keeping recently or frequently used data on chip so that when the data is needed again, a long offchip access to main memory is avoided. However, caches are of finite size and previously used data may need to be evicted to make room for other data before being needed again. Cache misses happen when data is needed but is not in the cache either because it was never needed before, or was evicted from the cache. Cache prefetchers try to reduce cache misses by predicting data that is likely to be needed in the near future and proactively fetching it into the cache [201]. Unfortunately, the prediction may be inaccurate, leading to cache pollution, or not timely, leading to reduced or negative benefits [17, 201, 209].

Our governors use the BIPS (Billion Instructions Per Second) throughput metric to determine current application or to set performance targets. We assume the existence of user-supplied software routines that convert between BIPS and high-level performance metrics. Similar to existing governors in Linux, our new governors do not keep track of higher-level application constructs such transactions, queries, etc. The workloads that we consider for our experiments do not have any latency constraints. Other workloads that have latency constraints on high-level constructs should estimate their BIPS requirement and communicate that to the governors.

Our new governor for SLAee significantly improves BIPS-per-watt (energy efficiency), with a maximum improvement of 67% and an average (geometric mean) improvement of 30% (Section 3.8) compared to the performance-per-watt of the highest-frequency configuration (EP energy efficiency). Improving performance-per-watt is important as it translates into more work being done for the same energy cost or less energy being used for the same amount of work.

Our new governor for SLApower improves over traditional RAPL-based governors by controlling prefetch enable and governing for full system power (Section 3.9).

The SPECpower benchmark calculates a figure of merit called "overall ssj_ops/watt" ($= \frac{\text{avg. load}}{\text{avg. power}}$). This is the overall energy efficiency over all the load levels. With only static frequency selection (single frequency for the entire run), this figure of merit is maximized at 3 GHz achieving 16% higher figure of merit compared to that for the configuration with the maximum frequency. However, our new governor for SLAperf (Section 3.10) results in 31% higher figure of merit by carefully controlling frequency and prefetch settings dynamically. Note that the power meter that we use is not accepted by SPEC as valid for submitting reports and the figure of merit calculations mentioned above may be approximate.

The main contributions of this chapter are:

1. We develop a new OS governor that seeks Pareto-optimality for socket frequency (and associated voltage) scaling and hardware cache prefetching. To the best of our knowledge, this is the first work to develop governors that simultaneously control for these two knobs on a real system. OS governors currently do not seek to control for hardware prefetching.

2. We propose a two-level design for constructing SLA-aware governors. The first level predicts the Pareto frontier (Dynamic EO) and the second level chooses a state on the frontier that targets the desired SLA. This makes governors easily retargetable to different SLAs.

Section 3.3 describes existing governors in Linux. Section 3.6 develops a new governor for maximizing energy efficiency using frequency (and associated voltage) control. Section 3.7 extends this to include cache prefetch control. Section 3.8 discusses how to control

for wall (full system) power in addition to socket and memory power. Sections 3.9 and 3.10 develop governors for maximizing performance under a power cap and minimizing power for a performance target.

3.2 Infrastructure

We use the Intel Haswell server, described in Section 2.2 of Chapter 2 and henceforth referred to as HS, for our experimental evaluations in this chapter. The OS acpi-cpufreq interface allows controlling frequency in 15 steps from 0.8–3.5 GHz (0.8–2.0 GHz and 2.1 GHz–3.5 GHz in steps of 200 MHz) and enabling/disabling the turbo boost region (3.5^+ GHz). Although writing MSRs (Model Specific Registers) directly provides greater control, we use this interface for a fair comparison with the existing governors. (We make an exception to this rule in Section 3.10.3 where we consider all frequencies for our new reactive governor that seeks to minimize idle time.)

We measure socket power and DRAM power using an additional software thread that reads available RAPL (Runtime Average Power Limit) counters [52, 113] at 1 second intervals. This runs as a thread separate from application threads and any governor threads. The governors that we develop also read RAPL counters for power calculations. We measure wall power with a Watts Up? (.net) meter [107] at 1 second intervals. This also runs as a separate additional thread for experiments where we use wall power.

We consider 14 workloads from SPECOMP2012 [206], graph500 [88], hpcg [66, 188], and SPECpower [205]. Of these, graph500 and SPECpower are run to completion whereas the other workloads are run for the first 1200 seconds of their executions (a few runs of kdtree complete within this time at high frequencies).

3.3 Governors in Linux

The Linux acpi-cpufreq module includes the following governors [37] that control the operating frequency. The goal is to manage power-performance by either setting the frequency statically, or by varying it in response to processor utilization. The governors available in our system are shown below. The root user can dynamically change the governor.

- **PowerSave** (**S**): Sets all cores to the lowest frequency. The idea is to use the least amount of power to do the work, but performance may be less than what could be achieved on this machine.
- **OnDemand** (**O**): Periodically samples (default: 10 ms interval) cores to adjust frequencies based on core utilization. The idea is to reduce power by lowering frequency when the CPU is not fully utilized and increase frequency as utilization increases so that the performance impact is minimal. The **Conservative** (**C**) governor is a variant of the OnDemand governor with more conservative utilization thresholds for changing frequencies.
- **UserSpace** (**U**): The idea is to give the root user control of the frequency settings. On HS, the root user can set the socket frequency (all cores together) to any of the allowed frequencies. The interface does not allow per-core settings for HS. All cores transition to the highest frequency of any core in the socket. This mode is useful only if the workload is known well in advance so that it can be run with different frequencies to determine the best setting. This is not practical in most deployments but is useful in reasoning about improvement opportunities.
- **Performance** (**P**): Sets all cores to the highest frequency. The idea is to get the maximum performance. This governor also uses the maximum power.

To further distinguish between modes, we constrain **U** mode to exclude **S** or **P** mode frequencies, i.e., it operates in the range of 1.0–3.5 GHz.

While these governors attempt to control knobs (e.g., processor frequency) in the system, none of them seek to meet SLAs that deal with energy consumption, power limits, or performance targets.

3.4 Two-level governor design

There are two challenges in developing governors that seek to optimize for SLAs—dealing with multiple hardware reconfiguration knobs (DVFS, prefetching, and possibly more to be made available in future) and dealing with different SLAs. To simplify governor design and make them retargetable to different deployment scenarios, we propose the following two-level governor design:

1. **Pareto Predictor**: This predicts the power-performance Pareto frontier for the system and currently observed execution profile.

2. **Objective Selector**: This level selects the desired operating state from the Pareto frontier according to the SLA to be achieved.

The objective selector remains unchanged if the available knobs change and the Pareto predictor remains unchanged if new SLAs are targeted. We believe that this simplifies governor construction and portability.

The above separation is possible because of a few properties of the Pareto-optimal frontier:

- Configurations that optimize power-performance metrics lie on the Pareto frontier (Section 2.5, Property 3). This makes it sufficient to focus only on the frontier to meet SLAs.

- Power and performance of states on the Pareto frontier have the same monotonic ordering relation (Section 2.5, Property 2). This makes predicting effects of system configurations easier, e.g., reducing a power cap will reduce performance.

Section 3.6 describes the basic sampling and interpolation schemes used by the Pareto predictor. This is slightly extended in Section 3.7 to include an additional reconfiguration knob (cache prefetching). The objective selector for SLAee computes performance-per-watt for each predicted point on the frontier and selects the next system state to be the one that is expected to maximize performance-per-watt. Objective selector mechanisms for SLApower and SLAperf are described in Sections 3.9 and 3.10 respectively.

Modern systems [181] often include a centralized power-control unit (PCU) that collects telemetry information from functional blocks and performs control actions. Our proposed Pareto predictor can be colocated with or implemented by such a PCU to reduce runtime overheads.

Our new governors do not control frequency in the turbo boost region (3.5+ GHz) except for the SLAperf governor for SPECpower (Section 3.10). This is because we cannot control frequency exactly in that region, but a vendor implemented version of our work would not have this difficulty. It is possible to control performance indirectly by limiting power in this region at a fine granularity (e.g., 0.125W [141]) through the RAPL capability (also see Section 3.9). However, as we shall show, energy-efficient operations are usually at much lower frequencies. Hence we do not include the additional complexity of finely controlling the turbo region in our governors.

3.5 Deployment Scenarios

One deployment scenario makes the simplifying assumption that the system can be completely shutdown (zero power draw) when there is no work and (near-)instantly fully enabled when work arrives. As an example, Figure 3.2 shows the performance in Billion Instructions-Per-Second (BIPS) and power consumption in Watts (W) of graph500 [88] for different socket frequency settings on HS in this deployment scenario.

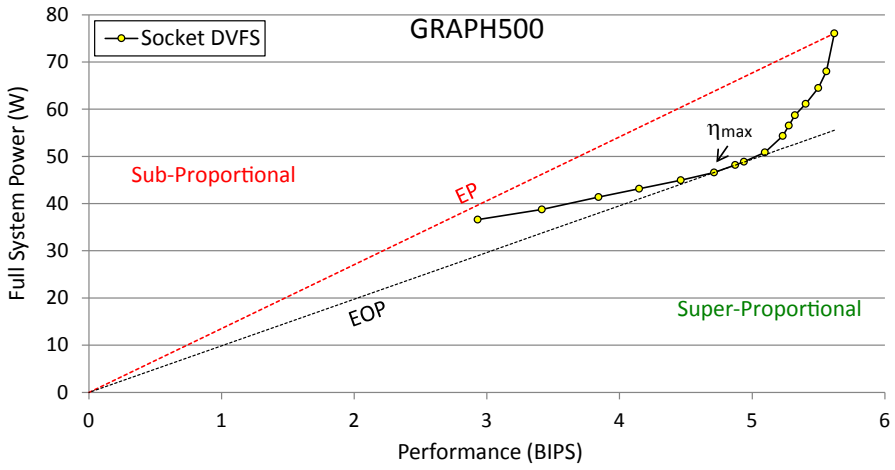


Figure 3.2: Example power-performance profile with Wall Power.

The SPECpower workload also measures the wall power of the system as the power cost to complete work. Additionally, SPECpower includes power measurements when the system is under zero load, that is, no work done.

In practice, this deployment scenario may not be realizable as there is usually a non-trivial latency cost while booting the system. An alternate deployment scenario assumes that the system transitions to active idle (non-zero idle power) when there is no work and is (near-)instantly fully enabled when work arrives. This minimize delays due to system wakeup when a workload is dispatched. In this scenario we are interested

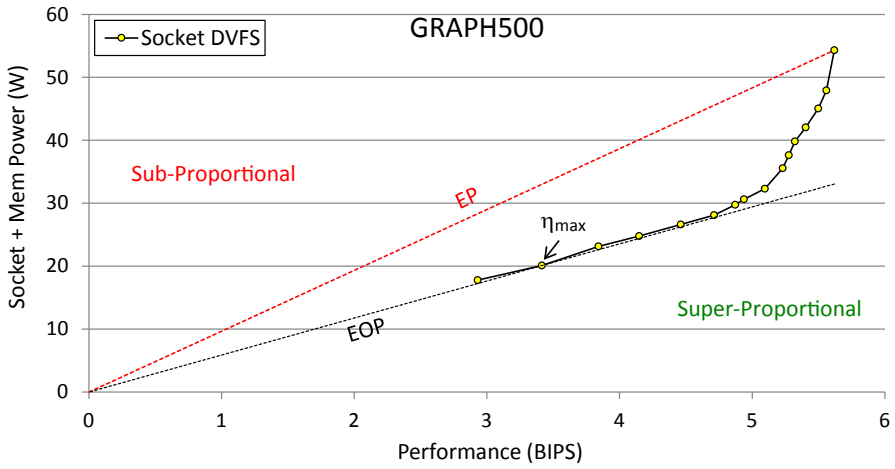


Figure 3.3: Example power-performance profile with Socket + Mem Power.

in the extra cost, measured by the socket and memory RAPL (Running Average Power Limit [113]) energy counters, to execute the workload. We will start with this deployment scenario for developing our governors. Figure 3.3 shows the power-performance profile for graph500 in this scenario.

For both scenarios, the EP line connects the origin (no extra power used for no work done) to the highest performing point with the default Peak Performance Configuration (all cores at their highest frequencies, prefetching enabled). While performance remains the same in both scenarios, the power consumption accounted for varies. Section 3.8 shows how to convert between the two power models. The states having the maximum efficiency (η_{max}) are higher-performing states in the first scenario, than in the second scenario, to compensate for idle power.

EP delivers performance in proportion to the extra power used. The EP line divides the power-performance landscape into two regions—"Sub-Proportional", where the performance is less than proportional to the extra power (equivalently, lower energy efficiency than that of EP), and "Super-Proportional", where the performance is more

than proportional to the extra power (equivalently, higher energy efficiency than that of EP). Our governors aim to operate the system at Dynamic EO (power-performance Pareto frontier) that automatically accounts for super-proportional behavior, if it manifests.

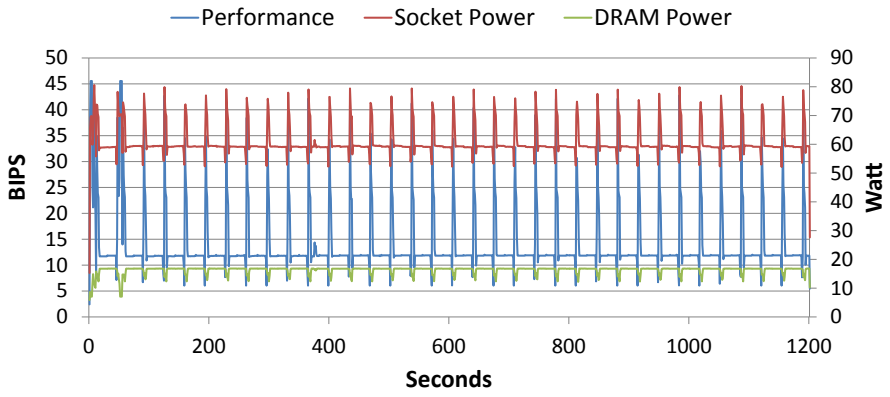
3.6 SLAee: Maximize energy efficiency

Our first goal is to develop a new governor that *seeks Pareto-optimal operation* and by doing so improves energy efficiency, measured as performance-per-watt (BIPS/Watt). For this section, we consider system power as the sum of socket power and DRAM power, both of which are estimated using RAPL counters. This corresponds to the second scenario (Figure 3.3) discussed in Section 3.5.

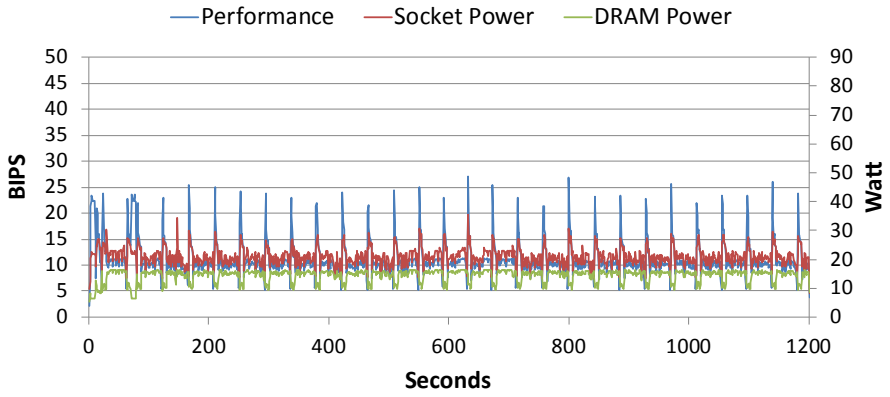
Figures 3.4a and 3.5a show example power-performance traces for `applu` (SPECOMP2012) and `graph500` in **P** mode. Figures 3.4b and 3.5b show energy-efficient (but lower-performing) executions using our new governor that we will describe shortly. Lower-performing executions that save energy may be desirable in situations with relaxed performance constraints, e.g., in batch executions, and when energy costs are important to the user.

Workload `applu` performs several iterations, each with a memory-intensive portion followed by a compute-intensive portion; the performance and power spikes indicate iteration boundaries. The DRAM power drops during the compute-intensive part of each iteration due to less memory accesses. Workload `graph500` runs 64 iterations of breadth-first search after initialization. Both `applu` and `graph500` exhibit long-term periodic behavior in both performance and power readings, with periods of tens of seconds corresponding to iteration lengths. Long-term stability in average power-performance profiles reduces differences between fixed-time and fixed-work experiments.

HS exhibits significant opportunities in improving BIPS /Watt (equivalently, Instruc-



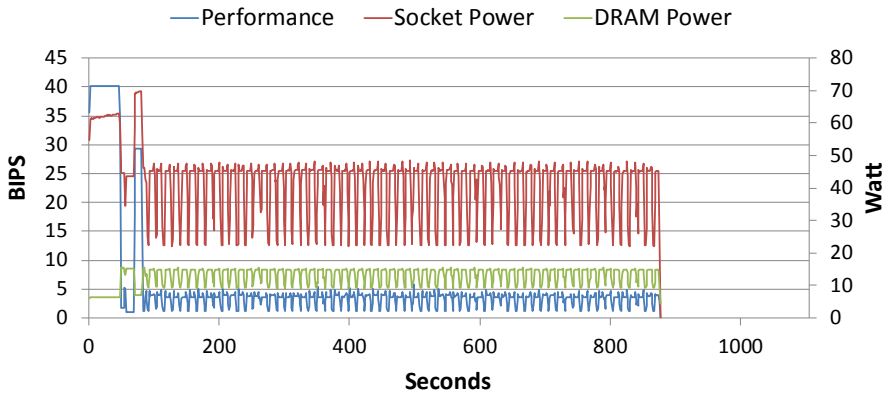
(a) applu. Instructions/nJ = 0.18.



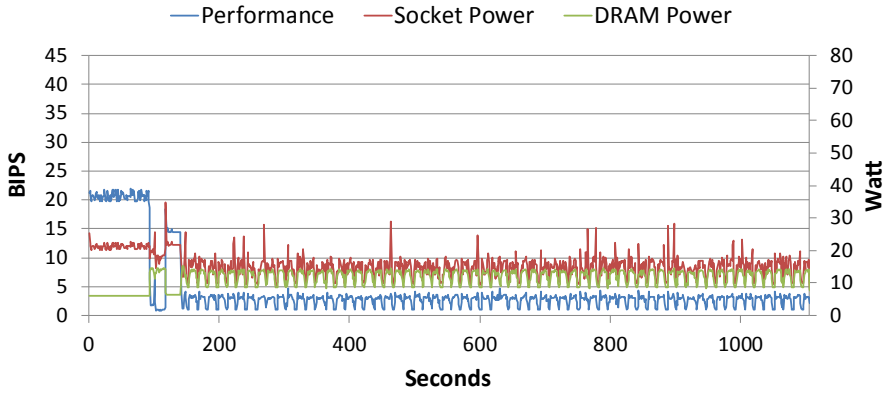
(b) applu. Instructions/nJ = 0.32.

Figure 3.4: Power-Performance traces for applu in **P**-mode and **R(10)**-mode on HS. Higher Instructions/nanoJoule implies more energy efficiency.

tions/nanoJoule) by changing frequency settings alone. BIPS changes between 1.18x (swim) to 4.86x (bwaves) in going from **S** to **P** modes whereas power changes between 2.52x (swim) to 5.67x (botsalgn), leading to a BIPS/Watt range of 1.29x (imagick) to 2.14x (swim) between best and worst values for that workload over all frequencies. For all these workloads, the minimum BIPS/Watt happens for **P** mode. applu and graph500 show a BIPS/Watt range of 1.84x and 1.67x respectively (also see Figure 3.6).



(a) graph500. Instructions/nJ = 0.10.



(b) graph500. Instructions/nJ = 0.16.

Figure 3.5: Power-Performance traces for graph500 in **P**-mode and **R(10)**-mode on HS. Higher Instructions/nanoJoule implies more energy efficiency.

We implement a simple reactive, **R(t)**, mode of operation to exploit the improvement potential. Our approach is to sample power and performance at a few different frequencies, then use that information to interpolate the frontier. Referring to Figure 3.2 as an example, we see that at least three samples are needed to target super-proportionality. In contrast, aiming for proportionality would require only two points, but the non-linearity in system behavior between the points could not be predicted or controlled.

We implement two power-performance predictors (in software)—one for the socket subsystem and the other for the memory subsystem. The socket predictor sets the frequency to 0.8 GHz (lowest frequency), 2.1 GHz (midpoint frequency) and 3.5 GHz (nominal frequency) in three consecutive intervals of t ms each and observes the power, performance, memory read and write bandwidths for each setting. It then interpolates (quadratic or piecewise linear) the effects for the other frequencies.

A software coordination module, running on one of the cores, reads the socket predictions and DRAM predictions every 51t ms (immediately after the 3t socket sampling), composes the predictions, estimates the frontier and selects the best frequency. The length of the interval that the system runs in this state is 48t. It is during this time that the DRAM predictor is periodically invoked (every 12t ms) to adjust a computed linear regression between DRAM power and read and write bandwidths (two variables) based on current readings. The regression is reset every 17 observations (204t ms) to react faster to phase changes. We choose this value since 17 is not divisible by 4 ($48t/12t = 4$), so the regression will not be reset at the time when the readings are needed for the power estimations with the interpolated values.

Since the optimal frequency will be in of the high/mid/low ranges, the sampling overhead is approximately $\frac{2t}{51t}$~4%. The workload continues to execute, although sub-optimally, in those two sampling intervals, so the temporal overhead is usually $< 4\%$. Figures 3.4b and 3.5b show power-performance traces for `applu` (SPECOMP2012) and `graph500`, in **R(10)** and the improvement in BIPS/Watt.

For all of the timing intervals mentioned above, we do not account for additional governor overheads due to system calls, interpolation, estimations, etc. So, the actual intervals will be slightly longer. The governor in Section 3.10.3 accounts for these overheads.

There are three main issues in implementing the interpolant for the socket predictor:

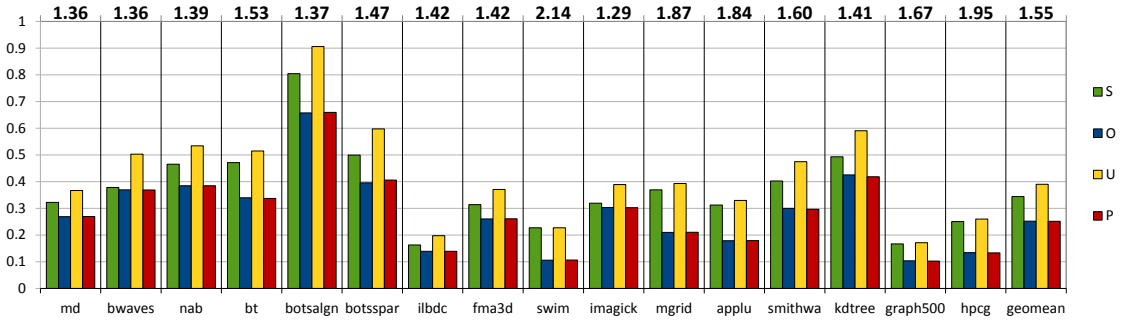
1. Getting successive sample points that show non-decreasing performance and power with increasing frequency.
2. Getting sample points with acceptable measurement noise/jitter.
3. Dealing with non-convexity of the frontier.

The first issue arises when the workload exhibits local phase behavior. The second issue arises with rapid sampling that makes the jitter in the energy measurements seem to be higher than that in the timing measurements leading to occasionally unrealistic power calculations. The third issue arises when the number of samples is not enough to correctly estimate the shape of the frontier.

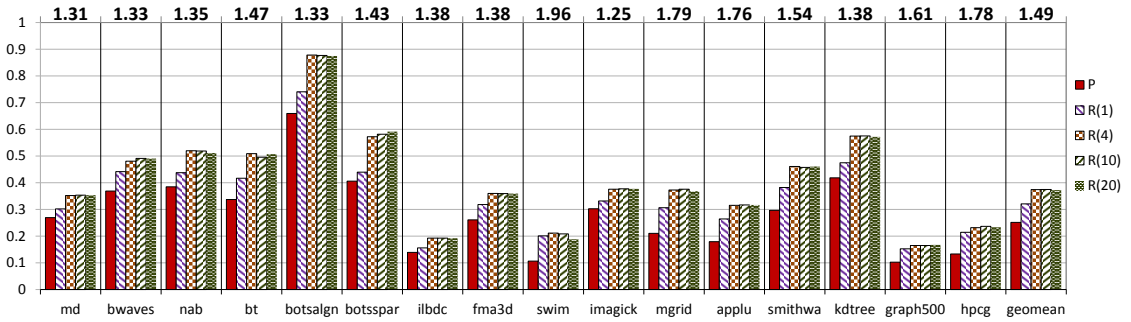
To deal with the first two issues, we disregard samples if either decreasing values are found or if power readings differ in more than 10x between the three samples and the coordinator transitions to 3.5 GHz. While other default actions are possible, we choose to penalize ourselves when we are not confident about the interpolation. On average (geometric mean) less than 2% of samples are discarded, but the frequency can occasionally be high, e.g., ~10% for bwaves in **R(1)**. We do not correct for the third issue and our results will be suboptimal for non-convex frontiers.

Figure 3.6 compares the energy efficiency (performance-per-watt) with different modes of operation. For all these workloads, the **P** mode has the lowest efficiency. The numbers at the top show maximum gains (e.g., 1.36 implies 36% gains) in energy efficiency over **P**-mode by selecting the optimal frequency in **U** or **S** modes (Figure 3.6a) and **R(10)** mode (Figure 3.6b). We observe that:

- *The potential rewards for selecting optimal configurations are significant*: The efficiency improvements were 28.6% (imagick) to 113.7% (swim) over **P** (geometric mean: 55%



(a) The Top line shows Performance-per-Watt of the best **U** mode frequency relative to **P** mode



(b) The Top line shows Performance-per-Watt of **R(10)** relative to **P** mode

Figure 3.6: BIPS-per-Watt on HS with different policies.

over **P**, 13.4% over **S**). The **O** and **P** modes are suboptimal for this metric for every workload.

However, the improvements come at a performance cost. Compared to **P** mode, the most energy-efficient frequency setting for each workload resulted in a reduction of BIPS of around 12.8% (mgrid) to 45.3% (botsspar), with a geometric mean of 35.4%. So, there is a tradeoff between energy savings and performance loss. For batch executions the performance loss may be tolerable. For other executions, Section 3.10 discusses a governor that try to reduce energy while meeting a given performance constraint.

- *There is no single best static frequency setting*: The best static frequency settings for the different workloads were 0.8 GHz (swim), 1.0 GHz (applu, graph500, hpcg), 1.4 GHz (mgrid), 1.8 GHz (bt, botsspar), 2.0 GHz (ilbdc, smithwa, kdtree), 2.1 GHz (md, nab, botsalgn, fma3d), 2.3 GHz (bwaves, imagick).

- *Rapid profiling and reconfigurations are not necessary for long running workloads*: We did a sensitivity analysis with **t**=1 msec, 4 msec, 10 msec, and 20 msec. The resulting performance-per-watt numbers indicate that **R(20)** (geometric mean: ~48% over **P**, 8.3% over **S**), **R(10)** (geometric mean: ~49% over **P**, ~9% over **S**), and **R(4)** (geometric mean: 48.7% over **P**, 8.8% over **S**) improved over **R(1)** (geometric mean: 27.5% over **P**, -6.7% over **S**). For the rest of our discussion on this governor we will focus only on t=10 msec, that is, **R(10)**.

We observe that many workloads exhibit long-term variation and periodic behavior. For example, applu shows ~34.3 sec periodicity in **P** mode (Figure 3.4a) and ~42.3 sec with **R(10)** (Figure 3.4b). graph500 exhibits ~12.3 sec periodicity in **P**-mode (Figure 3.5a) and ~15 sec with **R(10)** (Figure 3.5b). We expect long training intervals that track considerable execution history to work well with such workloads.

The socket predictor could use a variety of interpolants, e.g., piecewise linear or quadratic, to predict power and performance for different frequencies from the profiled data. The choice of the interpolant trades off accuracy with computation cost.

We use quadratic interpolation for the socket predictor. A piecewise linear interpolation would be faster, but for the performance-per-watt metric, only one of the sample frequencies (0.8/2.1/3.5 GHz) would get chosen as the optimal frequency. This is because $Perf(f) = af + b,\ Pwr(f) = cf + d \implies Perf(f)/Pwr(f)$ is monotonic in f. So, the maxima will always occur among the end points of the interval. This is a generic result and is not limited to using frequency as the independent variable.

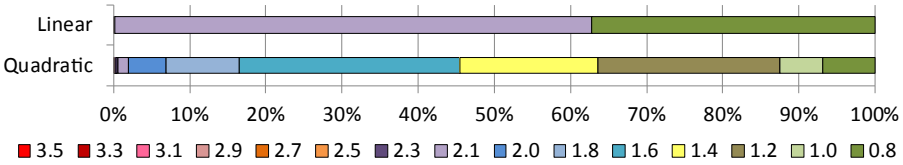


Figure 3.7: R(10) freq. distribution for applu (0.8–3.5 GHz).

To overcome this, we also evaluated quadratic interpolation for an alternative socket impact predictor without needing to change the coordinator. Figure 3.7 shows the frequency distribution for both schemes for applu. While Linear fluctuates mostly between 0.8 and 2.1 GHz, resulting in ~66% improvement over **P**-mode, Quadratic selects more frequencies in between resulting in 76.4% improvement. mgrid showed similar improvements.

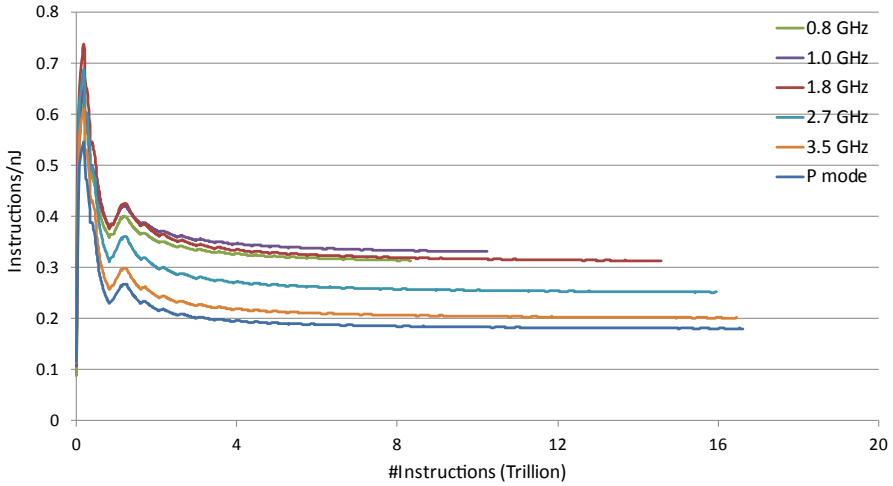


Figure 3.8: Average energy efficiency of applu as a function of the number of instructions executed and processor frequency.

For our evaluations, we run SPECOMP workloads for the first 1200 seconds. Only a few runs (when run at high frequencies) of kdtree finish within this time while other runs and all runs of other workloads do not finish. Doing fixed-time experiments, as

opposed to fixed-work experiments, runs the risk of comparing efficiency numbers from incomparable runs. However, for these workloads, the differences are small. For example, Figure 3.8 shows that, for any operating frequency, the average energy efficiency (over the total instructions executed so far) of applu stabilizes after a sufficiently large number of instructions have executed. So comparing across runs having different, but large, number of instructions is possible.

We re-evaluated results using the same number of instructions (minimum across all policies from the fixed-time runs) for each workload. In terms of **U**-mode gains over **P**-mode, applu changed from 84% to 80% whereas botsspar changed from 47% to 52%. The geometric mean changed < 2% for all policies. All trends remained the same.

3.7 SLAee: Adding L2 Prefetch Control

Hardware prefetching on Intel x86 machines can be enabled or disabled by writing specific values to Model-Specific Registers (MSRs) [112]. All prefetchers are enabled by default. In this study we keep the DCU (L1 Data Cache) prefetchers enabled, but dynamically enable or disable the L2 prefetchers. We set the prefetching mode for all cores identically.

Figure 3.9 shows one example workload each for beneficial prefetch (mgrid) and harmful prefetch (md) with all possible socket DVFS settings. When prefetching is disabled, md shows 14% improvement in peak performance and 13.6% in maximum energy efficiency whereas mgrid shows 12.3% loss in peak performance and 14.9% loss in maximum energy efficiency. Since prefetching benefits are workload dependent, a static prefetch setting will always be suboptimal for some workloads.

We extend the frequency governor in the following simple way: Instead of taking one sample at 2.1 GHz, we take two samples—once with prefetching enabled and once

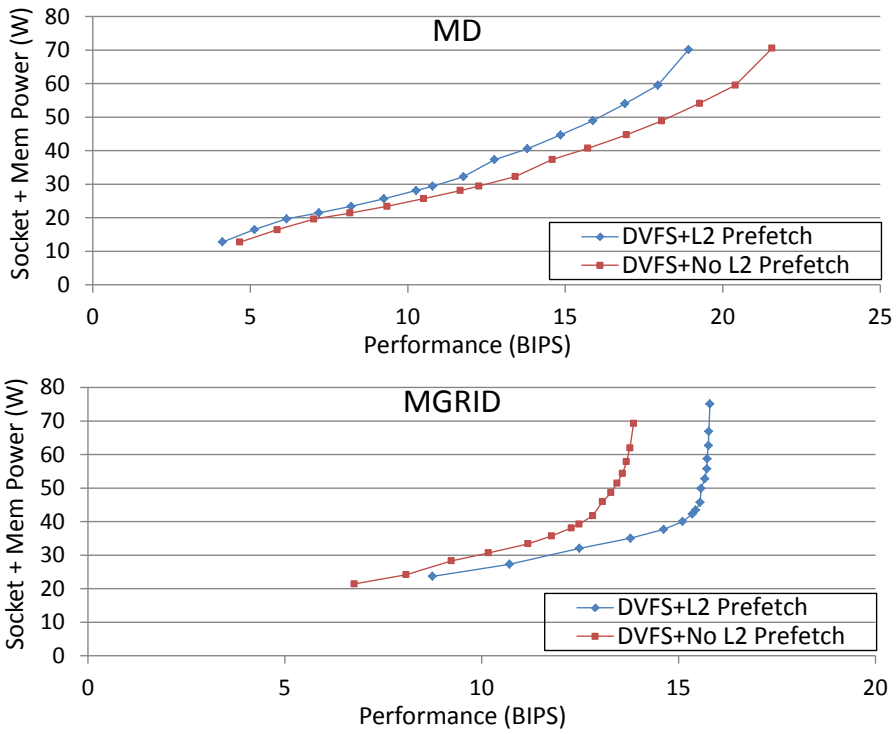


Figure 3.9: Example profiles. States at the Pareto frontier have L2 prefetching disabled for md, but enabled for mgrid.

disabled. We choose the prefetching mode that gives better performance and continue with that for the remaining two samples and estimating the frontier for that interval. Our choice of 2.1 GHz for taking the initial two samples is motivated by the need to keep the overhead of taking an extra sample small. A mid-range frequency, such as 2.1 GHz, is likely to incur a lower additional overhead for this than a high frequency, such as 3.5 GHz since the energy-efficient operations for most workloads are not at high frequencies.

Similar to **R(t)**, we name the new governor **RF(t)** (Reactive with prefetch control), parametrized by **t**, the length of the profiling interval in milliseconds. Figure 3.10 shows the distribution of prefetch modes (enabled/disabled) selected by **RF(10)** for our workloads. As expected, md ran with prefetching mostly disabled whereas mgrid ran

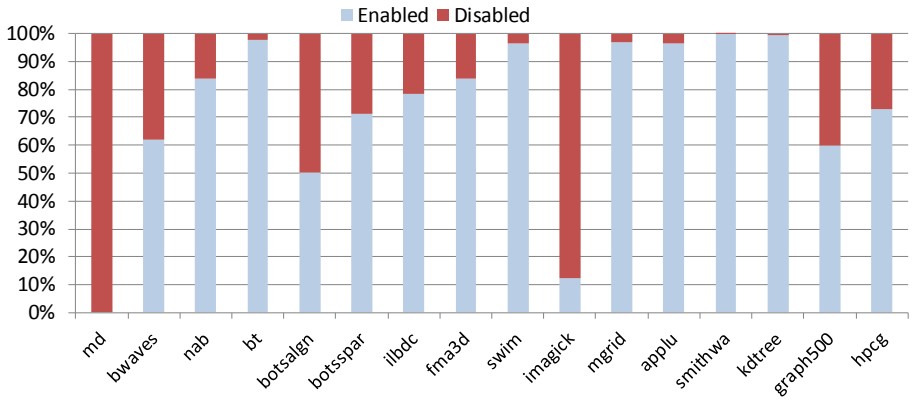


Figure 3.10: L2 Prefetch mode distribution by **RF**(10).

with prefetching mostly enabled. Apart from mgrid, workloads bt, swim, applu, smithwa and kdtree also predominantly chose to keep prefetching enabled. Other workloads chose between both enabled and disabled modes.

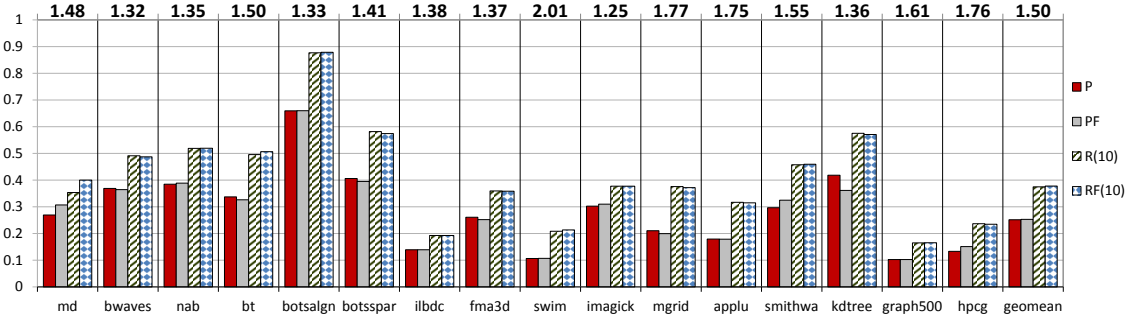


Figure 3.11: BIPS-per-watt of governors with (**RF**(10)) and without (**P**, **PF**, **R**(10)) dynamic control for L2 Prefetching.

Figure 3.11 shows performance-per-watt for four governors—**P**, **R(10)** (prefetching always enabled), **PF** (prefetching always disabled) and **RF(10)** (prefetching dynamically controlled). The secondary horizontal axis (top) shows the improvement (e.g., 1.48 implies 48% improvement) in performance-per-watt of **RF(10)** compared to **P**. **RF(10)**

improved performance-per-watt beyond **R(10)** for md (48% instead of 31%) but did not create significant differences for other workloads.

To summarize our results so far, we find that the **P**, **PF** and **O** modes can always be improved by the other policies. **S** works best for swim, well for hpcg, but can be improved by **U**, **R** and **RF** for the other workloads. **U** is a good policy to use provided that workload is known in advance and profiling experiments can be carried out. **R** reaches close to **U** but is unable to outperform it. This is likely because these workloads have long-term stable behavior (see for example, Figure 3.8) making the best static frequency not a bad choice. On the other hand, **R** suffers from runtime profiling overheads and prediction errors due to sampling inconsistency and non-convexity of the frontier. The situation changes for SPECpower (see Sections 3.10.2 and 3.10.3), where reactive governors do significantly better than static frequency settings. **RF** further improves upon **R** if disabling prefetch is useful but does not hurt energy efficiency if not, so it represents the best of both prefetch modes.

3.8 SLAee: Adding Control for Wall Power

For the experiments in Sections 3.6 and 3.7, we consider system power as the sum of processor and memory power as estimated by the RAPL counters. We do not consider the power consumptions for other components (e.g., power supply, network interfaces, hard disks, etc.) that account for 20-30W power, with system idle power of HS at ~26W. HS has a 350W, 80 Plus Gold PSU (Power Supply Unit) [218, 230]. We will now correct for the extra system power considering a deployment scenario where the cost to execute a workload includes the wall (full-system) power.

Measuring wall power requires external power meters (e.g., Watts Up? meters) and are usually available only at long measurement granularities (e.g., minimum 1 sec intervals)

as opposed to the millisecond granularity of RAPL counter measurements used by our governors (**R(10)** requires a measurement interval granularity of 10 msec).

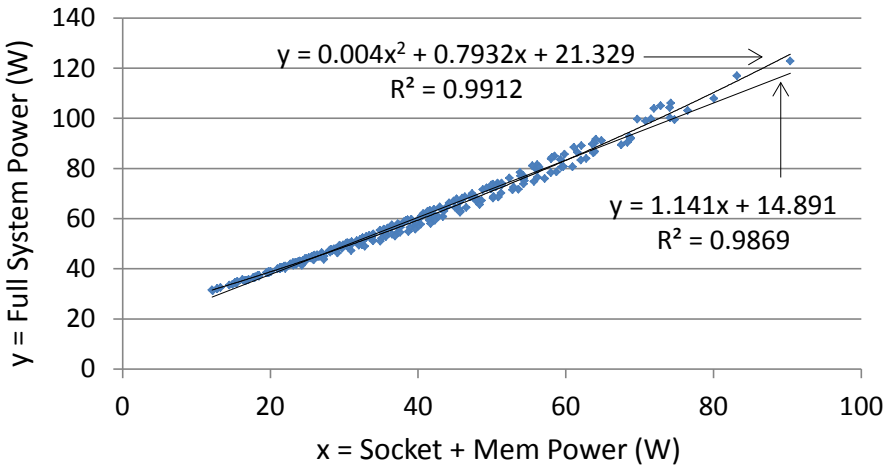


Figure 3.12: RAPL and Wall Power correlation.

To correct for the extra power we need to model the relation between the socket+mem power (measured by RAPL counters) and wall power (measured by a Watts Up? meter). Figure 3.12 shows the two values for all frequency settings for our workloads. We then fit a linear model and a quadratic model to the data. The slope of the straight line is consistent with the power efficiency of 87% (1.141^{-1}~0.876) of this PSU at <50% load [218, 230]. However, the y-intercept suggests a system idle power of ~20W (@x=~4.6W, socket+mem idle power) instead of the ~26W actually observed. Conversely, the quadratic model gives a better fit for idle power as well as a slightly better fit overall. One reason for this could be that load-dependent variations in the power efficiency of the PSU give rise to some non-linearity. In that case the SPUE (Server PUE) metric [104] that tracks power supply overheads may be more accurately characterized by a formulation that includes both load-dependent and load-independent factors than by a single number, e.g., 1/(rated power efficiency of the PSU). We use the quadratic model to estimate wall

power from socket+mem power.

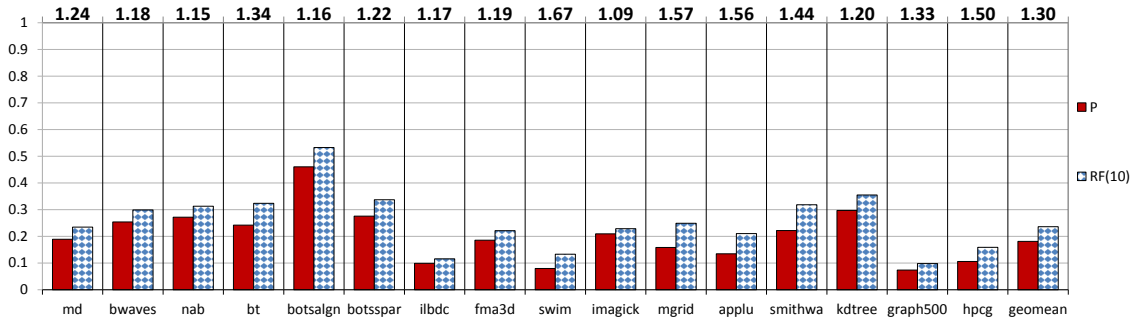


Figure 3.13: BIPS-per-watt of governors **P** and **RF**(10) with wall (full-system) power.

Figure 3.13 shows the performance-per-watt for the baseline **P** governor and our new **RF** governor, with wall power for the run measured using a Watts Up? meter. During the run, **RF** uses RAPL counters for profiling, then applies the wall power model mentioned above to the interpolated power numbers.

The maximum gains in performance-per-watt achieved by **RF** over **P** is 67% (swim) while the geometric mean over all workloads is 30.2%. Although the improvements are somewhat smaller than those in Figure 3.11 due to consideration of the extra system power for both the governors, this is significant improvement in energy efficiency realized on a real server machine. Since CPUE is directly proportional to energy consumption and inversely proportional to energy efficiency, we have $\frac{CPUE(\mathbf{P})}{CPUE(\mathbf{RF})} = \frac{E(\mathbf{P})}{E(\mathbf{RF})} = \frac{\eta(\mathbf{RF})}{\eta(\mathbf{P})} = 1.302$. So the average energy savings of **RF** over **P** is $\frac{E(\mathbf{P})-E(\mathbf{RF})}{E(\mathbf{P})} = 1 - \frac{E(\mathbf{RF})}{E(\mathbf{P})} = 1 - 1.302^{-1} = 23.2\%$.

Energy savings reduce operational expenses that in turn reduces TCO for datacenters. **RF** thus opens up opportunities either for cost savings by using ~23% less energy on average to do the same work or for revenue generation by doing ~30% more work on average for the same energy cost.

While it improves energy-efficiency, **RF** loses performance, with respect to **P** mode,

ranging from 0.2% (md) to 30% (bt) with a geometric mean of 19.5%.

3.9 SLApower: Maximize performance within a power cap/budget

None of the standard Linux governors **S**, **C**, **O**, **U**, **P** deal with power caps/limits. There is no way for the user to specify power caps to these governors.

The RAPL [113] capabilities include mechanisms to enforce a limit on the power consumption. One advantage of RAPL limits over frequency settings is that they can be fine-grained (e.g., units of 1/8 W) leading to greater control of the state space. Another advantage is that since RAPL limits are enforced by the hardware, the management overhead is lower than that of a software-controlled governor. Prior works [141, 213] have used power limiting as a mechanism to improve energy efficiency.

There are two main disadvantages of the RAPL power-capping mechanisms:

1. Capping of wall power cannot be directly specified. One needs to use a model, similar to the one that we developed in Section 3.8, to convert wall power limits to RAPL domain power limits.

2. Management of non-frequency resources does not automatically happen through the RAPL mechanisms. For example, setting or clearing RAPL limits does not affect prefetching status (enabled/disabled). So, workloads such as md that benefit significantly from prefetch control would not see those advantages with the RAPL approach. From this perspective, RAPL guarantees a power cap, *not best performance within that power cap*. Pareto optimality provides the stronger guarantee.

The first limitation can be easily overcome, but the second limitation is more profound and needs more effort to address. This limitation is likely to be accentuated with the

addition of more reconfigurable knobs in future systems. Our new governor for SLApower improves upon the RAPL governor in this aspect. We demonstrate it by controlling prefetch settings as well as socket frequency to get the maximum performance within a specified power budget.

For the RAPL experiments in this section, we limit average socket power over 1 second intervals from 10W to 80W in steps of 5W. For our system, we could enforce a power cap only for the entire socket, not for the memory or for other components.

We develop a new governor (by modifying the objective selector) to select the next state that is predicted to use the highest power among all states with less power consumption than the SLA target. We name this governor **RF_SLApower(t)**. There are three differences between this governor and the RAPL governor:

1. We specify limits on full system power or socket power as needed. The Pareto predictor uses the model shown in Figure 3.12 to estimate wall power from socket and memory RAPL energy counter measurements. We specify power limits in steps of 5W from 35W to 80W for graph500 and 105W for md. In contrast, for the RAPL governor we capped socket power and measured the resulting system power. Since the RAPL interface does not allow specification of full-system power caps, the two power limits do not have an exact correspondence.

2. We impose a power limit on every reconfiguration interval, which is 510 msec long for **RF_SLApower(10)**. So it is somewhat more strict than the sliding window of 1 second that we used for the RAPL experiments.

3. We do not set turbo mode frequencies. This is because our Pareto predictor has limited control over frequency selection and hence limited insight on power consumption in turbo mode. So, the maximum frequency that we set is the nominal

frequency (3.5 GHz). However, this limits the maximum performance that can be attained.

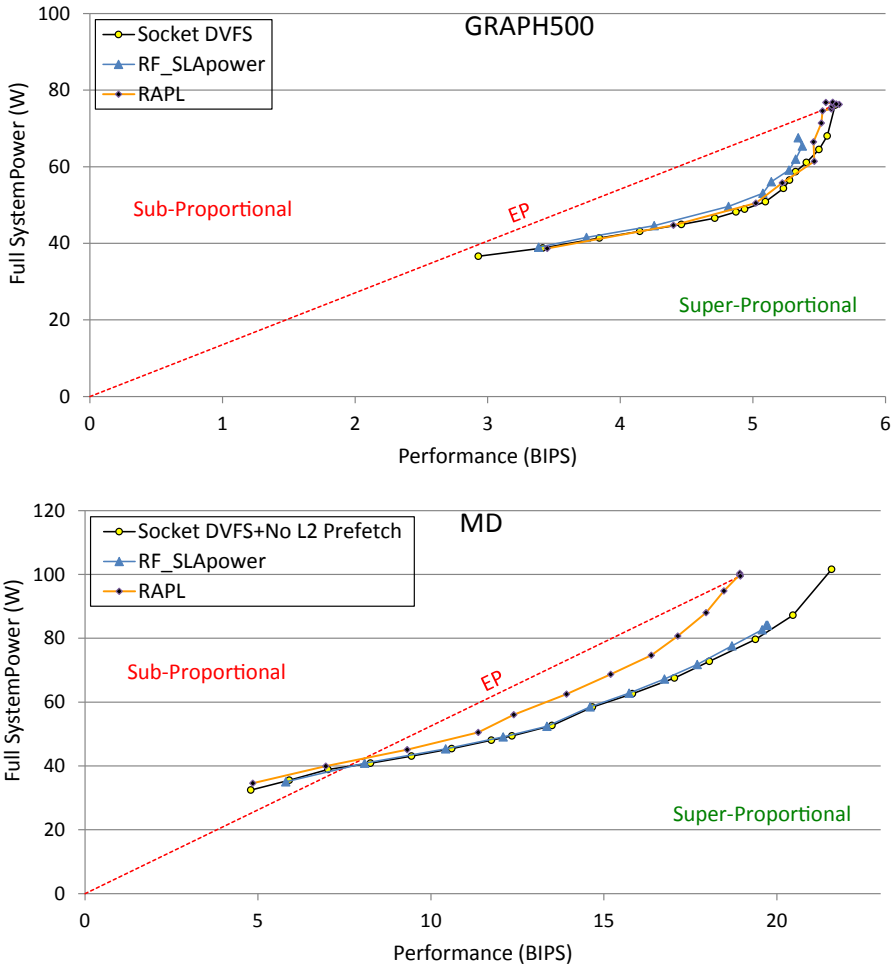


Figure 3.14: Power-performance profiles for graph500 and md for SLApower.

We investigate enforcement of SLApower using **RF_SLApower(10)** and the RAPL governor for two of our workloads: graph500 and md. Figure 3.14 shows the power-performance profiles for both approaches. Both workloads exhibited behavior close to Pareto optimal with **RF_SLApower(10)**. The RAPL governor works well for graph500,

but causes Pareto-dominated (hence suboptimal) operations for md as it does not control prefetch settings.

We also observe that **RF_SLApower(10)** falls short of achieving the maximum performance at the upper end of operating range. This is because the maximum allowable frequency in this governor is 3.5 GHz (nominal frequency) since we cannot effectively control the turbo range. One way to get around this limitation could be to use RAPL to control DVFS settings and have **RF_SLApower** control other settings, e.g., prefetch settings. Of course, one needs to to calculate the appropriate RAPL power caps from system power caps using the inverse of the power mapping function shown in Figure 3.12.

3.10 SLAperf: Maximize power savings given a performance target

None of the standard governors **S**, **C**, **O**, **U**, **P** deal with performance targets. There is no way for the user to specify performance targets to these governors. The RAPL capabilities (see Section 3.9) allow power caps to be specified, but not performance targets to reach.

We name our new governor **RF_SLAperf(t)**. It allows the user to specify performance targets in absolute or relative (with respect to peak) BIPS. The governor is agnostic of higher-level performance goals, e.g., transactions per second or response latency distributions. The user needs to have a mapping between such performance goals to one of the performance targets mentioned above. We will now describe the governor designs for these two types of performance targets.

3.10.1 Governing for absolute performance targets

To govern for this SLA, we keep the Pareto predictor intact, but modify the objective selector to additionally keep track of the performance so far (time elapsed and instructions executed). This allows it to set the desired performance target for the next interval so that if the target for the interval is met, then the average performance so far would be that required by the SLA. The objective selector makes one of three possible choices:

1. If the average performance so far is greater than the SLA, the lowest performing point is chosen.

2. Otherwise, if the next interval target is greater than the best performance predicted for 3.5 GHz, turbo mode is chosen.

3. Otherwise, the point on the frontier that meets or just exceeds the next interval target is chosen.

The above is a simple policy for the objective selector. Other policies are possible and they will be useful particularly if they include workload semantics and predictions of future workload behavior since the Pareto predictor lacks both these dimensions.

We investigate enforcement of SLAperf using **RF_SLAperf(10)** for two of our workloads: `md` and `graph500`. `md` has mostly homogeneous behavior during its execution and we will show that it can be governed well to meet the SLA. On the other hand, `graph500` has significant heterogeneity (different execution phases) and, as we shall show, cannot be governed well without prior knowledge of the phase behavior.

`md` has an average performance range of 4.8–21.6 BIPS at the frontier depending on the frequency setting and L2 prefetching disabled. We select SLA targets of 5.0, 7.5, 10.0, 12.5, 15.0, 17.5, 20.0 and 22.5 BIPS (unreachable). `graph500` has an average

performance range of 2.9–5.6 BIPS at the frontier depending on the frequency setting and L2 prefetching enabled. We select SLA targets of 3.0, 3.5, 4.0, 4.5, 5.0 and 5.5 BIPS.

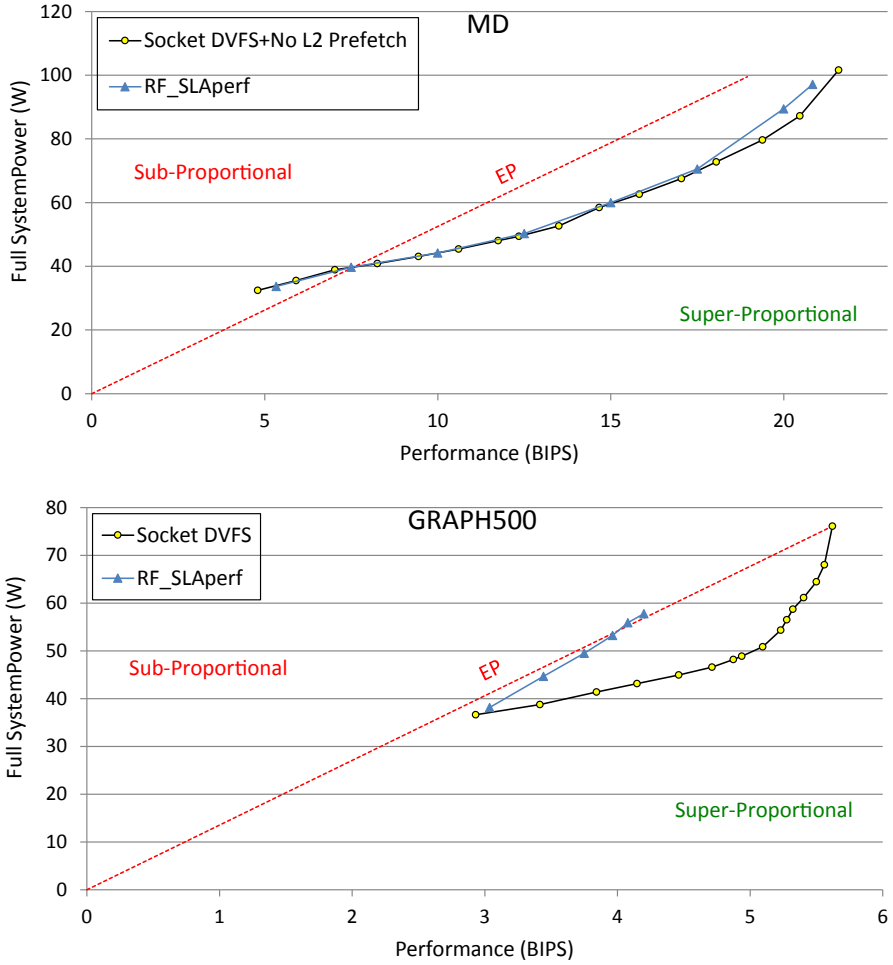


Figure 3.15: Power-performance profiles for md and graph500 for SLAperf.

Figure 3.15 shows the power-performance profiles and expected behavior for both workloads. The profile for md was at the frontier except at high performance targets. Its highest performance is around 3.5% less than the maximum possible. This is due to sampling/profiling and prediction overheads in the governor. Also, when it is given a

target close to its highest performance, it tries to compensate for the loss by transitioning more into turbo mode resulting in more power consumption and consequently, Pareto-dominated states.

The governor failed to meet the SLA for most points of graph500 and the profile was quite suboptimal, being closer to proportional than Pareto optimal. However, as we explain below, this is primarily due to the non-homogeneous nature of the workload rather than incorrect state transitions chosen by the governor.

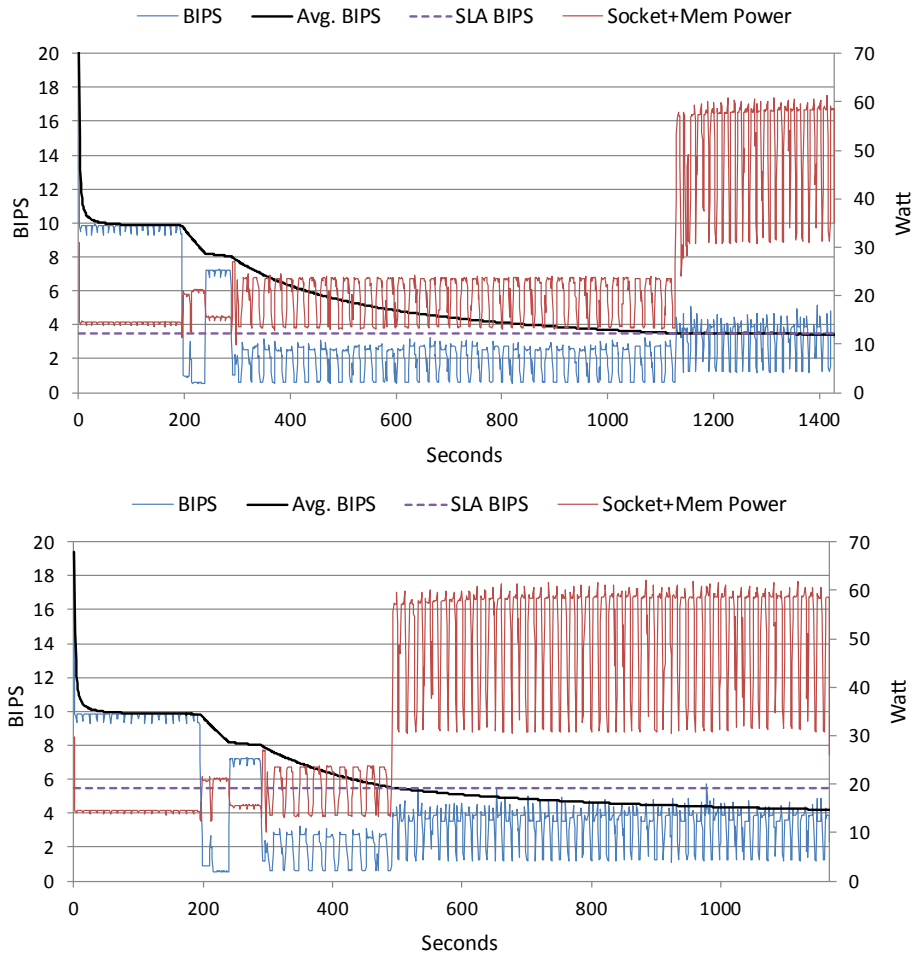


Figure 3.16: Execution profiles for graph500 with SLA = 3.5 BIPS and 5.5 BIPS.

Figure 3.16 shows detailed execution profiles for graph500 for two SLAs: 3.5 and 5.5 BIPS. The primary vertical axis (left axis) and the blue line show the number of instructions executed per second in BIPS. The secondary vertical axis (right axis) and the red line show the power (socket+memory) in Watts. The Avg. BIPS line shows the average BIPS attained by the workload execution so far. The dashed line shows the average BIPS expected to be reached over the entire execution. The governor seeks state transitions that will maintain the average BIPS equal to the SLA BIPS.

graph500 has non-homogeneous behavior—the initial ~290 seconds is mostly a high performance phase where high BIPS is possible whereas the remainder of the execution (successive iterations of breadth-first search) consists of a low performance phase. Initially, the average BIPS is higher than the SLA BIPS, so the governor reduces frequency to the lowest possible to save power. Eventually, execution enters the low performance phase and the average BIPS starts dropping more rapidly. However, the governor continues with a low frequency execution as the SLA BIPS is still lower than the average BIPS. After a while (1152 seconds for SLA=3.5, 491 seconds for SLA=5.5), the average always remains below the desired SLA although the governor transitions to higher frequencies. By now it is too late to take corrective action because of the nature of the low performance phase. The governor ends up transitioning to turbo mode (observe the sharp increase in power consumption), but average BIPS continues to drop. This is more pronounced for SLA=5.5, where the SLA is breached earlier in the execution, than for SLA=3.5.

This case study highlights a challenge with targeting this SLA for non-homogeneous workloads. There was no way that the governor could have known about future execution characteristics without such information being provided to the objective selector. The executions were doomed to miss SLA quite some time before the violations started to appear. Timely prediction of maximum attainable performance in future execution

intervals is needed to resolve this issue.

Another challenge for targeting this SLA is that it is highly sensitive to the accuracy of online performance predictors. For our governors, we sample execution at different operating points and use that data to predict performance for the next interval using interpolation for intermediate points assuming convex behavior. This also assumes that the behavior in the next interval will closely match the characteristics of the sampled execution. Some inaccuracy in performance prediction is inevitable when one or more of these assumptions are violated. These issues are not specific to our governor design and must be addressed by any control policy seeking to target this SLA.

3.10.2 Governing for relative performance targets

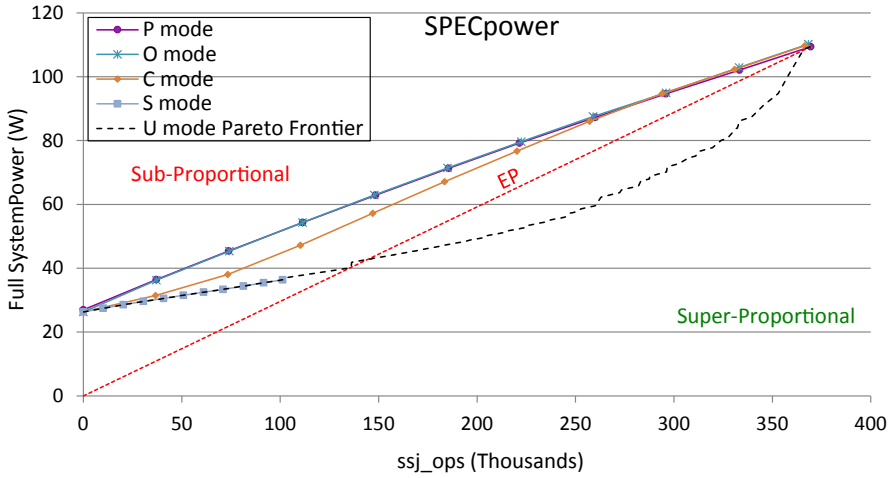


Figure 3.17: Power-performance profiles for SPECpower with Linux governors.

Figure 3.17 shows power-performance profiles for SPECpower with the default governors. **P**-mode and **O**-mode perform similarly and use the highest power for the achieved load. **S**-mode uses the lowest power for the achieved load, but the maximum load

achievable is low (see below). **C**-mode works better than **P**-mode or **O**-mode at low loads but in general consumes significantly more power than **S**-mode or the **U**-mode Pareto frontier for the same load. For these experiments we "niced" the power measurement daemon and set the ignore_nice_load parameter of the **O** and **C** governors to discount activity by the daemon while calculating processor utilization.

Even though the **S**-mode profile lies on the Pareto frontier, there are two limitations with this policy:

1. It can only serve up to around 27% of peak load. So it will fail the performance requirement (load served must be within a certain limit of the offered load; see Chapter 2, Section 2.2, for details) at higher loads. The other governors in Figure 3.17 can serve high as well as low loads.

2. Although RUE = 1 for this governor, LUE $\geqslant$ 1.58. This means at least 58% excess energy used compared to operating with the best load even though it is operating at the Pareto frontier. Actually, its profile does not even enter the Super-Proportional region.

So, restricting servers to this policy is not a good idea. The other governor profiles are distant from Dynamic EO, thus having large RUE values. So, using these policies will lead to significant energy waste.

To govern for this SLA, we modify the Pareto predictor to also sample turbo mode so that peak BIPS can be estimated. Instead of limiting the maximum profiling frequency to 3.5 GHz, we now have the maximum profiling frequency as 3.7 GHz (midpoint of the turbo range 3.5–3.9 GHz) since we do not know the actual temperature-dependent operating frequency.

For any load, the target relative BIPS is set to be equal to that load level (load relative

to maximum load). For example, attempting to service 70% load level sets the target relative BIPS to 0.7. The objective selector selects the lowest-power configuration that is predicted to have relative BIPS greater than or equal to the target relative BIPS.

We call this new governor **RF_SLAperf(t)**. The parameter *t*, in milliseconds, determines the length of the intervals as follows. For 100% target performance,

1. Turn prefetching off and profile for t msec.
2. Turn prefetching on and profile for t msec. Choose and set prefetching mode that performed better.
3. Run with that setting for the next 48t msec.

For this target performance, the frequency is kept at 3.7 GHz and not changed.

For a lower target performance, the plan is

1. Set frequency to 2.1 GHz (midpoint frequency). Turn prefetching off and profile for t msec.
2. Turn prefetching on and profile for t msec. Choose and set prefetching mode that performed better.
3. Set frequency to 3.7 GHz (turbo frequency). Profile for t msec.
4. Set frequency to 0.8 GHz (lowest frequency). Profile for t msec.
5. Estimate the Pareto frontier, select the best frequency, run with that setting for the next 48t msec.

DRAM bandwidth and energy consumption are profiled as before.

Figure 3.18 shows power-performance profiles with governors **R_SLAperf(10)** and **RF_SLAperf(10)**. **R_SLAperf(t)** has prefetching always enabled whereas **RF_SLAperf(t)**

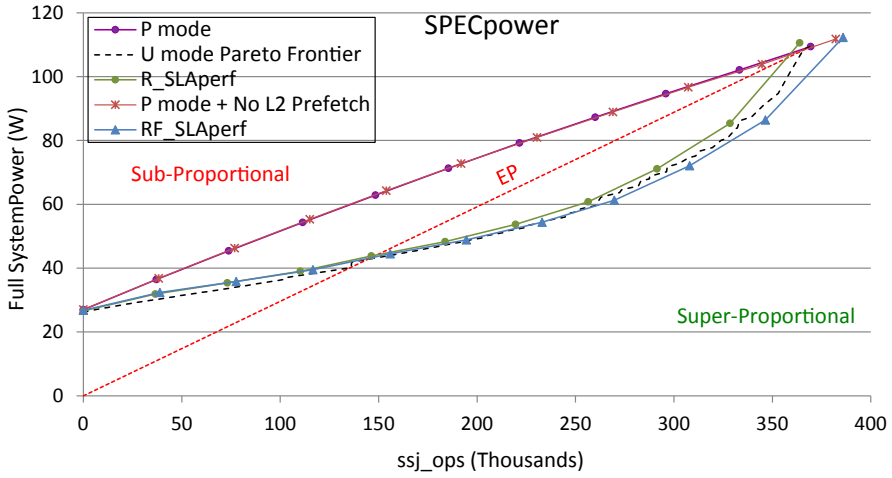


Figure 3.18: Power-performance profiles for SPECpower with **RF_SLAperf(10)** and **R_SLAperf(10)**.

dynamically controls it. Controlling prefetch settings increases the maximum load achievable compared to **P**-mode that always has prefetching enabled. **RF_SLAperf(10)** outperforms **P**-mode by around 4%. This outcome is qualitatively similar to md (See Figure 3.14). The increase in maximum performance can also be observed by running in **P**-mode with prefetching disabled (**P**-mode + No L2 Prefetch profile).

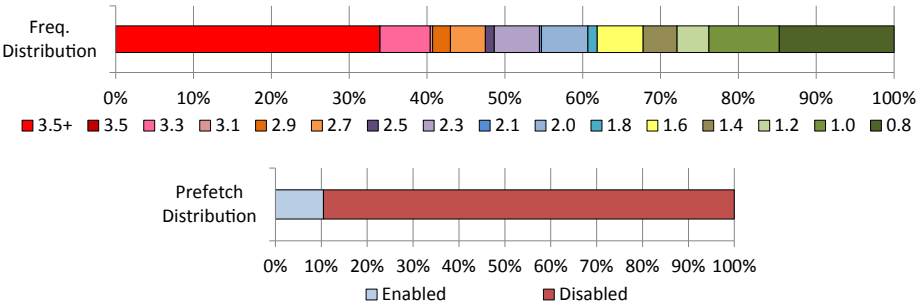


Figure 3.19: Distributions of frequency and prefetch settings for SPECpower with **RF_SLAperf(10)**.

Figure 3.19 shows distributions of frequency and prefetch settings selected by governor

RF_SLAperf(10) over the entire run that includes both calibration and measurement intervals of SPECpower (see [205], or, a brief description of SPECpower intervals in Section 2.2). As expected, it disabled prefetching for most of the time.

RF_SLAperf(10) also chose a number of different frequencies for each run depending on the load serviced. By default, SPECpower does 3 calibration intervals followed by 11 measurement intervals. During the calibration intervals and the first measurement interval, the server is offered very high load. So we expect that during these times the maximum frequency would be chosen by the governor. Thus for at least (3+1)/(3+11) = 28.6% of the time, the maximum frequency should be chosen. The last measurement interval is "Active Idle", that is zero load is offered, so we expect to select the lowest frequency during this interval which is around 1/(3+11) = 7.1% of total time. We observe from Figure 3.19 that the 3.5+ GHz setting (that is, maximum frequency in turbo mode) was chosen for around 34% of the time and 0.8 GHz (the lowest frequency) was chosen for around 15% of the time.

3.10.3 Governing to minimize idle time

So far, a performance target (absolute or relative) needed to be specified to the governors so that they could ensure that SLAperf is satisfied. We will now discuss a new governor that aims to service the offered load without keeping any processing contexts idle. We will demonstrate its action in the context of the SPECpower workload execution.

The key idea is to predict the highest frequency such that there are no idle cycles. Let α denote the number of active (that is, not idle) cycles per second in the last interval. The governor estimates the optimal value of target frequency for the next interval to be $\frac{\alpha}{8}$. The division by 8 is done since there are 8 logical threads on HS. This assumes that in the next interval,

1. the load will remain the same (or at least, not increase) and

2. all threads will be equally active

In case these assumptions are not true, the system may not be able to serve the offered load. To protect against this situation, the governor increases the estimated target frequency by a step whenever it equals the current frequency and doubles the value of the step. This facilitates an exponential ramp-up of frequency over successive intervals so that the offered load is served. On the other hand, if the estimated target frequency is less than the current frequency, the frequency is set to the target frequency and the step is re-initialized. Additionally, our governor selects the best prefetch setting for the next interval.

We call our new governor **RF_Active(t,p)**. Following are the three main steps undertaken by our governor in each interval.

1. Turn prefetching off and profile for p/2 msec

2. Turn prefetching on and profile for p/2 msec. Choose prefetching mode that performed better.

3. Estimate and set target frequency for the remaining interval, so that the total interval is t msec.

Figure 3.20 shows the power-performance profile of our new governor with different parameter values: **RF_Active(10,2)**, **RF_Active(100,20)**, and **RF_Active(500,20)**. We statically determine the length of the interval in Step 3 by taking into account governor overheads for system calls, profiling, and estimations. Note that this governor does not predict either power or performance for any configuration, but manages to operate the system very close to Dynamic EO.

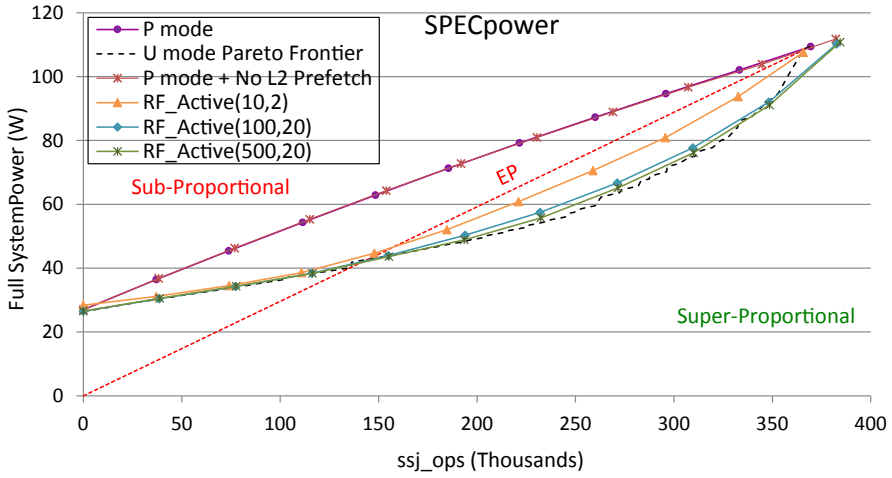


Figure 3.20: Power-performance profiles for SPECpower with **RF_Active(10,2)**, **RF_Active(100,20)**, and **RF_Active(500,20)**.

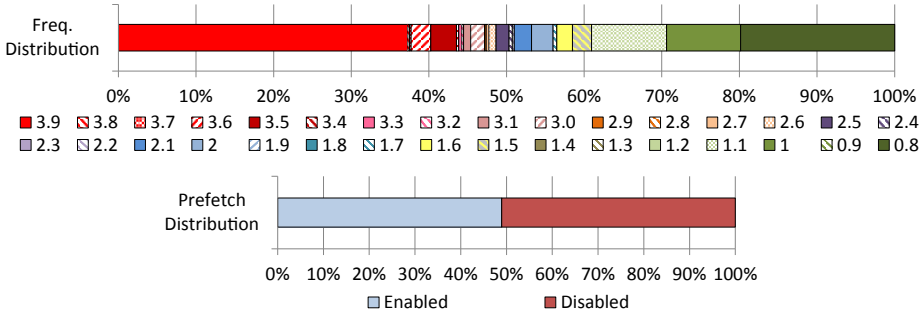


Figure 3.21: Distributions of frequency and prefetch settings for SPECpower with **RF_Active(10,2)**.

Figures 3.21–3.23 show the distributions of frequency and prefetch selections made by **RF_Active(10,2)**, **RF_Active(100,20)**, and **RF_Active(500,20)** respectively. For these experiments, we consider all frequencies, not just the ones available through the OS acpi-cpufreq interface. **RF_Active(10,2)** suffers from more overheads and less accurate selection of resource settings with shorter intervals—it keeps prefetching enabled for a larger fraction of time and also selects the maximum frequency more often than

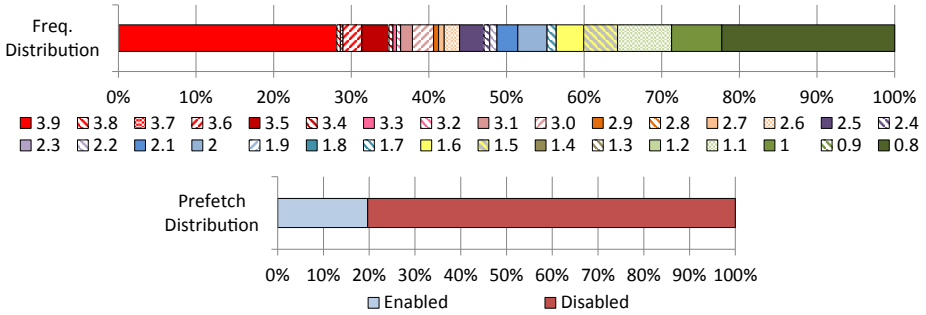


Figure 3.22: Distributions of frequency and prefetch settings for SPECpower with **RF_Active(100,20)**.

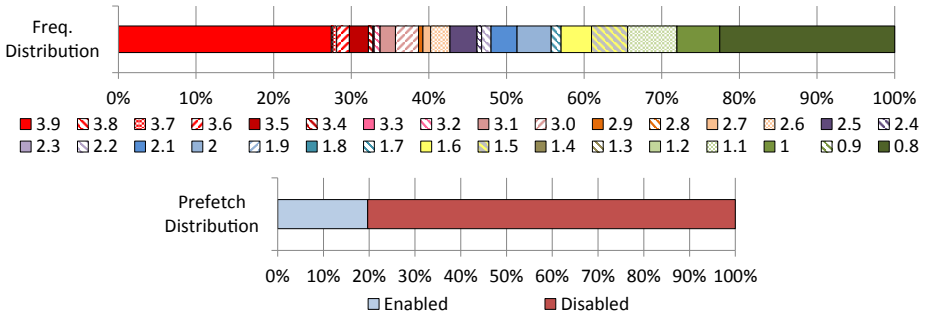


Figure 3.23: Distributions of frequency and prefetch settings for SPECpower with **RF_Active(500,20)**.

RF_Active(100,20) or **RF_Active(500,20)**. Compared to **RF_SLAperf(10)** (Figure 3.19), **RF_Active(100,20)** and **RF_Active(500,20)** select the lowest frequency more often and the maximum frequency less often but keep prefetching enabled for longer.

3.11 Limitations

We will now discuss a few limitations of our governor designs—socket-wide control, intrusive profiling, sampling inconsistency and non-representativeness, and non-zero reaction times.

3.11.1 Socket-Wide Control

Our governors select the same resource setting for the entire socket. This works well for HS, but newer machines offer capabilities for more fine-grained control, e.g., per-core DVFS settings. Having the same frequency setting for all cores may be suboptimal for non-homogeneous workloads if the hardware supports different per-core settings.

Even for machines supporting per-core control, we expect that we can continue with the current profiling strategy (low frequency, middle frequency, high frequency) for all cores. Our idea is to predict power-performance Pareto frontiers for each core and use those to select optimal per-core settings. However, to do that we need to know the performance and energy consumption of each core for each frequency setting profiled.

Current processors already report per-core performance. On HS we measure the energy consumption of the entire socket (HS does not report per-core energy consumption) but this will not be sufficient to determine per-core frontiers. If future processors report per-core energy consumption, then those values can just be used. Otherwise, per-core energy consumption needs to be predicted. One way to do this is to use per-core performance counter values, e.g., BIPS and generated memory bandwidth. Offline regression models that correlate performance counts with energy consumption could help to identify which counters are significant for the given machine.

Determining per-core prefetch settings is more difficult. Whether or not prefetching is useful depends on many factors such as the cache access patterns, the prefetching algorithm, and the timeliness of initiating prefetch. At this point we cannot suggest any efficient user-level or OS-level strategy to accurately determine optimal settings without additional hardware support from the vendor.

Constructing Pareto frontiers, using interpolation, individually for many cores can be costly in software. Hardware support for doing this in the PCU would be very

useful. Heuristics that trade off accuracy vs cost can be used if a software solution is required. For example, since piecewise linear interpolation only selects end points for optimal values (Section 3.6), interpolation is not required and only the end points can be considered. This would be faster than quadratic interpolation, but may be suboptimal for some workloads.

3.11.2 Intrusive Profiling

Our governors try out a few resource configurations (e.g., socket frequencies) to determine their effectiveness before selecting the predicted optimal configuration. This intrusive profiling may be costly both in terms of reconfiguration latency and energy for some resources, e.g., cache configurations. In Chapter 4 we describe a novel method for predicting cache performance using reuse distance distributions of cache accesses. That method does non-intrusive profiling, but requires new hardware support.

3.11.3 Sampling Inconsistency and Non-representativeness

Our governors infer the impact of DVFS on the Pareto frontier by sampling execution with three frequencies—lowest, high, and intermediate—over three consecutive intervals and then fitting a quadratic polynomial to the measured power-performance values. This strategy works if the samples have consistent properties and are not drawn from different execution phases. To protect against making invalid interpolations we implement simple checks on the sampled power-performance values, e.g., performance at the lowest frequency should not exceed that at the high frequency, etc. The sample inconsistency problem can be avoided by an alternate approach [202, 207] that samples performance counters once, then predicts power and performance for other configurations using a precharacterized model. However, this approach is limited by the number of performance

counters that can be concurrently read on real systems (can affect prediction accuracy) and also by the availability of the particular counters on different platforms (can affect portability).

3.11.4 Non-Zero Reaction Times

Our **R(t)** governors logically partition execution time into epochs with each epoch consisting of a profiling phase followed by a prediction phase (execution with the predicted optimal settings). The plan is t-t-t-48t without prefetch control and t-t-t-t-48t with prefetch control. The profiling phase lasts for t-t-t (total: 3t) or t-t-t-t (total: 4t) time, resulting in the epoch time being 51t or 52t. A small amount of extra delays exist due to overheads in system calls and calls to library functions to read system state and to execute the governor code.

Disregarding schedule variance (exact schedule timing may not be possible) and resource reconfiguration latency, the epoch time represents the worst-case time that the system needs to react to changes in execution characteristics. Shortening the prediction phase will allow faster reaction times at the cost of increasing profiling overhead.

The unit of time, t, cannot be made very small in part due to limited update frequency of the RAPL counters, usually about once every millisecond [96, 113]. In contrast, the DVFS transition time is typically a few tens to few hundreds of microseconds [95, 168] which is about one to three orders of magnitude smaller than t. However, these limits are due to hardware constraints and will affect other governors as well.

3.12 Conclusion

This chapter focused on online mechanisms to reduce RUE by constraining the system to operate close to the Pareto frontier. We developed new OS governors that seek

Pareto optimality in the presence of frequency scaling and cache prefetching and thereby improve the energy efficiency of a modern Intel Haswell server machine. We demonstrated improvements in performance-per-watt by up to 67% (maximum gains) and 30% on average (geometric mean over all workloads). This opens up significant opportunities for revenue generation or cost savings.

We proposed a two-level design to construct governors that are aware of Service-Level Agreements (SLAs) and can be easily retargeted to optimize for different SLAs. We also presented case studies and discussed challenges in governing for maximizing performance within a power cap and minimizing power for a performance target.

4 CACHE REUSE MODELS

4.1 Overview

Processor caches are critical components of the memory hierarchy that exploit locality to keep frequently-accessed data on chip. Caches can significantly boost performance and reduce energy usage, but their benefit is highly workload dependent. In modern power and energy constrained computer systems, understanding a workload's dynamic cache behavior is important for making critical resource allocation and scheduling decisions. For example, allocating excess cache capacity to a workload wastes power, as large caches dissipate significant leakage power, while allocating insufficient cache capacity hurts performance and increases main memory power.

Caches can be made dynamically reconfigurable to enable energy savings by exploiting workload characteristics. Previous research has explored placing some or all of a cache in low-power mode [10, 67, 77] or dynamically partitioning the cache to eliminate resource contention [175, 214]. A recent Intel processor [116] can dynamically reduce its LLC's capacity to save power. Researchers [240] have also explored mechanisms to dynamically reconfigure both the number of sets and the associativity of set-associative caches.

Being able to exploit cache reconfiguration capability will enable our governors to make the system operate more efficiently if possible. But real systems today offer limited or no support for user-/OS-driven cache reconfigurations. So, in this chapter and the next one, we will study cache reconfigurations using a simulator.

Cache reconfigurations incur significant overheads, so we will not consider the expensive approach of trying out various configurations before deciding on the best configuration. Instead, we will use online predictors that estimate the temporal locality in cache accesses and predict the performance of different cache configurations. These

models will be our focus in this chapter whereas the next chapter will focus on using these models to develop governors that control both core frequency and cache size to meet SLApower.

We develop a reuse distance/stack distance based analytical modeling framework for efficient, online prediction of cache performance for dynamically reconfigurable set-associative caches that use LRU/PLRU/RANDOM/NMRU replacement policies. Our framework is inspired by two foundational works: Mattson's stack distance characterization [150] (also used later as reuse distance [29, 65]) and Smith's associativity model [103, 200] for LRU caches.

The central theme of our framework is to decouple temporal characteristics in the cache access stream from characteristics of the replacement policy. We propose a novel low-cost hardware circuit, that uses Bloom Filters and sampling techniques, to estimate cache reuse distance distributions online. These distributions are then used as inputs to analytical models of replacement policy performance to predict miss ratios. This separation of aspects brings the advantage of being able to easily refine either aspect in isolation without affecting the other.

Our work differs from prior art in being suited for online predictions [103, 200], working with practical replacement policies other than LRU [103, 174, 175, 200, 215], allowing reconfigurability in the number of sets in addition to associativity [127, 174, 175], and being very low cost [84]. Our framework unifies existing cache miss rate prediction techniques such as Smith's associativity model, Poisson variants, and hardware way-counter based schemes.

The main contributions of this chapter are:

1. We formulate an analytical framework based on generalized stochastic Binomial Matrices [212] for transforming reuse distance distributions (Sections 4.4, 4.5).

2. We formulate new miss ratio prediction models for RANDOM (Section 4.5.2), NMRU (Section 4.5.3), PLRU (Section 4.5.4) replacement policies.

3. We show that the traditional hardware way-counter based prediction [215] for varying associativity is a special instance of our unified framework (Section 4.6.2). Further, we show how way-counter data for LRU may be transformed to apply to caches with a different number of sets. (Section 4.6.2.2)

4. We propose a novel hardware scheme for efficient online estimation of reuse distance/stack distance distributions (Section 4.6.1).

The rest of this chapter is organized as follows:

Evaluation Infrastructure: Section 4.2 describes the workloads and simulators that we used for this study.

Temporal locality metrics: Section 4.3 defines reuse distributions that capture the temporal locality of address streams. Section 4.4 shows how to modify these to apply for a cache with a different number of sets.

Replacement policy models: Section 4.5 introduces the notion of cache hit-functions that, when multiplied with the per-set reuse distribution, produce expected cache hit ratios. Sections 4.5.1.1 and 4.4.3 consider optimizations for LRU hit ratio prediction. Sections 4.5.2, 4.5.3 and 4.5.4 develops new prediction models for RANDOM, NMRU, PLRU respectively. Section 4.5.5 discusses prediction accuracy and computation overheads.

Hardware Support: Section 4.6.1 presents the novel, low-cost hardware for estimating reuse distributions. We also discuss two traditional hardware mechanisms—set-counters and way-counters (Section 4.6.2).

Index Hashing: Section 4.7 describes the index-hashing scheme that we used to map addresses to cache sets.

size \ assoc.	2	4	8	16	32
2MB	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}
4MB	2^{15}	2^{14}	2^{13}	2^{12}	2^{11}
8MB	2^{16}	2^{15}	2^{14}	2^{13}	2^{12}
16MB	2^{17}	2^{16}	2^{15}	2^{14}	2^{13}
32MB	2^{18}	2^{17}	2^{16}	2^{15}	2^{14}

Table 4.1: Relation between number of sets and associativity for different cache sizes. Assuming some cache configuration is the current configuration, there are a total of 25-1=24 possible other cache configurations. An inspection of the table reveals that *at most* 4 of these possible 24 configurations can have the same number of sets as the current configuration. For example, with 32MB 32-way as the current configuration, other configurations with the same number of sets (2^{14}) are: 2MB 2-way, 4MB 4-way, 8MB 8-way and 16MB-16-way. Thus, way-counters (Section 4.6.2) can predict for *at most* 4 of 24 possible other configurations at any time.

4.2 Infrastructure

In our study, caches are characterized by the number of sets S, associativity A, and replacement policy. We generally use S' and A' while referring to the target cache configuration for which we want to predict the miss ratio. We assume a fixed line size of 64 bytes. Table 4.1 shows the relation between S, A and cache size for the configurations of the Last-Level Cache (LLC) that we study.

Our models estimate hit ratio (hit/access). This is easily converted into other measures: miss ratio=1-hit ratio; miss rate=miss ratio*access/instruction. For evaluating prediction quality, we obtain address traces of accesses to a 32MB 32-way LLC in a simulated system (Table 4.2) for our workloads, run the traces through a standalone cache simulator (that does not model timing) and compare measured against predicted metrics.

Table 4.2 describes the 8-core CMP we use for gathering traces. We assume an 8-banked L3 cache that is dynamically re-configurable for a total of 25 configurations (see

Core configuration		4-wide out-of-order, 128-entry window, 32-entry scheduler		
Number of cores		8	**On-chip frequency**	2132 MHz
Technology Generation		32nm	**Temperature**	340K
Functional Units		4 integer, 2 floating-point, 2 mem units		
Branch Prediction		YAGS 4K PHT 2K Exception Table, 2KB BTB, 16-entry RAS		
Disambiguation		NoSQ 1024-entry predictor, 1024-entry double-buffered SSBF		
Fetch		32-entry buffer, Min. 7 cycles fetch-dispatch time		
Inclusive	**L1I Cache**	private 32KB 4-way per core, 2 cycle hit latency, ITRS-HP		
	L1D Cache	private 32KB 4-way per core, 2 cycle hit latency, ITRS-HP		
	L2 Cache	private 256KB 8-way per core, 6 cycle access latency, PLRU, ITRS-LOP		
	L3 Cache	shared, configurable 2–32 MB 2–32 way, 8 banks, 18 cycle access latency, PLRU, ITRS-LOP, serial		
Coherence protocol		MESI (`Modified`, `Exclusive`, `Shared`, `Invalid`), directory		
On-Chip Interconnect		2D Mesh, 16B bidirectional links		
Main Memory		4GB DDR3-1066, 75ns zero-load off-chip latency, 2 memory controllers, closed page, pre-stdby		

Table 4.2: System configuration.

Workload	**Time** (seconds)	**#LLC accesses** $\times 10^6$	**#unique line addresses** $\times 10^6$	**#pages in LLC accesses** $\times 10^6$
`apache`	0.562	177.764	3.829	0.136
`jbb`	0.260	35.474	5.831	0.123
`oltp`	0.410	150.126	2.401	0.138
`zeus`	0.322	10.488	1.003	0.048

Table 4.3: Workload Characteristics. Cache line size = 64 bytes. Page size = 4K bytes.

Table 4.1). The cache uses a hashed indexing scheme to map addresses to cache sets (see Section 4.7). We conservatively assume a constant access latency for all configurations.

We use 4 Wisconsin commercial workloads [4] (`apache`, `jbb`, `oltp`, `zeus`). Each workload uses 8 threads and runs for a fixed amount of work (e.g. #transactions or loop iterations [6]) that corresponds to ~4B instructions per workload. Each simulation run starts from a mid-execution checkpoint that includes cache warmup. Table 4.3 shows a summary of the characteristics of the workload executions.

Figure 4.1 shows the average miss ratios for a 32 MB 32-way LLC over the execution

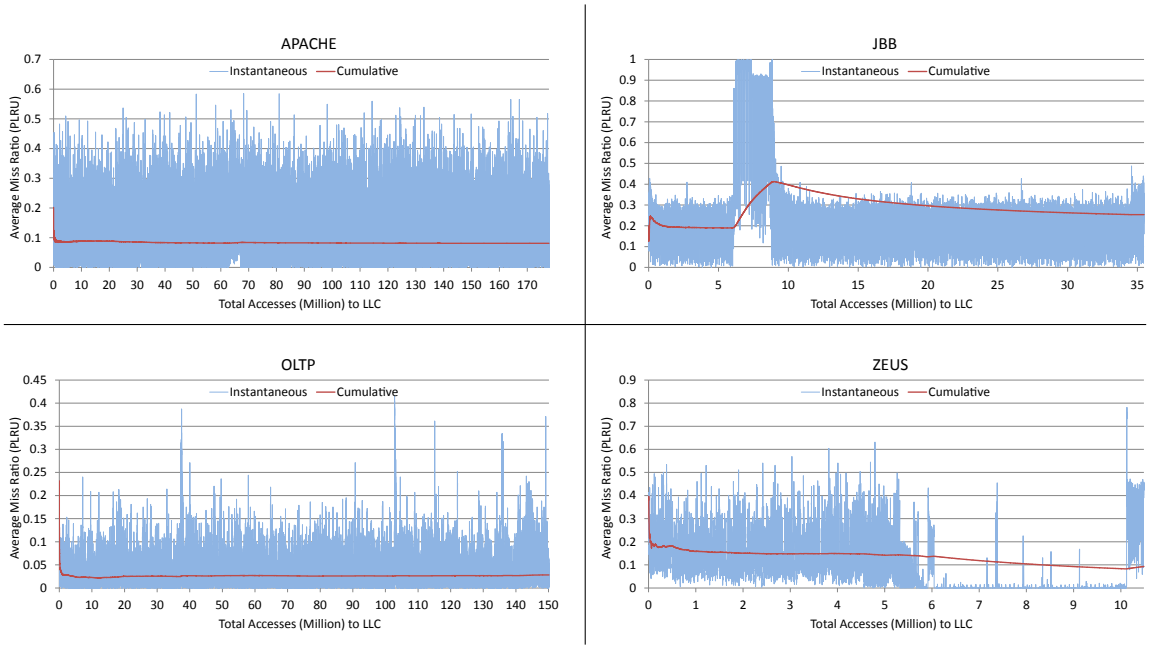


Figure 4.1: "Instantaneous" and cumulative miss ratios. Granularity is 1000 LLC accesses.

of the workloads. The "instantaneous" miss ratios are computed at the end of every 1000 accesses to the LLC and show a lot of variation for every workload. The cumulative miss ratio shows the average miss ratio of all LLC accesses in the execution till that point. It is much more stable than the "instantaneous" ratios. Its final value at the end of the execution is the overall/long-term average miss ratio. Our miss ratio prediction models aim to predict this long-term average.

Table 4.4 show why it is useful to consider reconfigurability in both the number of sets and ways (associativity) for the LLC. For a given cache size, M_* compares the maximum to minimum miss ratio for all configurations with that cache size (see Table 4.1 for the configurations). We obtain these numbers by running the access traces through our standalone cache simulator. As an example, `oltp` sees up to 57% increase in miss ratio with a suboptimal configuration for a 2MB LRU cache. M_S indicates what happens

Policy	Workload	2MB		4MB		8MB		16MB		32MB	
		M_*	M_S	M_*	M_S	M_*	M_S	M_*	M_S	M_*	M_S
LRU	apache	1.07	1.07	1.04	1.01	1.19	1.04	1.69	1.04	1.33	1.00
	jbb	1.03	1.03	1.14	1.08	1.13	1.02	1.09	1.00	1.08	1.00
	oltp	1.57	1.57	1.80	1.30	1.53	1.04	1.45	1.01	1.44	1.00
	zeus	1.13	1.13	1.09	1.03	1.07	1.01	1.08	1.00	1.07	1.00
RANDOM	apache	1.01	1.01	1.02	1.00	1.07	1.01	1.17	1.01	1.13	1.00
	jbb	1.01	1.01	1.02	1.01	1.03	1.01	1.02	1.00	1.02	1.00
	oltp	1.20	1.20	1.34	1.13	1.30	1.03	1.29	1.01	1.23	1.00
	zeus	1.02	1.02	1.03	1.01	1.02	1.00	1.02	1.00	1.02	1.00
NMRU	apache	1.03	1.00	1.02	1.00	1.02	1.00	1.07	1.01	1.06	1.05
	jbb	1.02	1.00	1.02	1.00	1.03	1.01	1.04	1.03	1.04	1.04
	oltp	1.12	1.12	1.21	1.03	1.18	1.00	1.17	1.01	1.11	1.01
	zeus	1.04	1.00	1.04	1.00	1.03	1.01	1.03	1.02	1.03	1.03
PLRU	apache	1.06	1.06	1.04	1.01	1.15	1.03	1.49	1.03	1.30	1.00
	jbb	1.03	1.03	1.10	1.05	1.11	1.02	1.07	1.00	1.05	1.00
	oltp	1.43	1.43	1.59	1.20	1.46	1.04	1.38	1.01	1.34	1.00
	zeus	1.10	1.10	1.08	1.02	1.05	1.00	1.05	1.00	1.05	1.00

Table 4.4: Relative miss ratios for difference cache sizes and replacement policies. M_* shows the max-to-min miss ratio over configurations having all possible number of sets for the given cache size. Relative ratios $\geqslant 1.05$ are shown in red. M_S shows the relative miss ratio of the configuration having the same number of sets (=2^{14}) as that of the largest cache (32 MB, 32-way) compared to the minimum miss ratio over all configurations for the given cache size. Entries with relative ratios $\geqslant 1.05$ are shaded.

if only way configurability is present. It assumes the same number of sets (2^{14}) as that for the 32MB 32-way cache. For `oltp`, this turns out to be the worst configuration for a 2MB cache (configuration: 2MB, 2-way)—it has 57% more misses than for the best configuration (2MB 32-way, number of sets = 2^{10}). Set configurability is more important at small cache sizes than at larger sizes.

4.3 Measures of Temporal Locality

In this section we develop metrics of temporal locality in the address stream that are independent of the cache configuration. These metrics will be used for estimating the miss ratios for arbitrary cache configurations. For our study, all addresses are (hashed) line addresses of cache accesses.

Consider an address trace T as a mapping of consecutive integers in increasing order, representing successive positions in the trace, to tuples (x, m) where x identifies the address and m identifies its repetition number. The first occurrence of address x in the trace is represented by $(x, 0)$. Let $t = T^{-1}$ denote the inverse function. $t(x, m)$ denotes the position of the m^{th} occurrence of address x in the trace. We now introduce a few more definitions.

Reuse Interval: The **reuse interval** (RI) is defined only when $m > 0$ and denotes the portion of the trace enclosed between the m^{th} and $(m-1)^{th}$ occurrence of x. Formally,

$$RI(x, m) = \begin{cases} \{(z, m') | t(x, m-1) < t(z, m') < t(x, m)\} & \text{if } m > 0 \\ \text{undefined} & \text{otherwise} \end{cases}$$

Unique Reuse Distance: This denotes the total number of unique addresses between two occurrences of the same address in the trace. Thus,

$$URD(x, m) = \begin{cases} \left| \{z | (z, m') \in RI(x, m)\} \right| & \text{if } m > 0 \\ \infty & \text{otherwise} \end{cases}$$

Numerically, this is 1 less than Mattson's much earlier stack distance [150].

Absolute Reuse Distance: This denotes the total number of positions between two

occurrences of the same address in the trace. Thus, $ARD(x, m) =$

$$\begin{cases} \left|RI(x, m)\right| = t(x, m) - t(x, m-1) - 1 & \text{if } m > 0 \\ \infty & \text{otherwise} \end{cases}$$

As an example, in the access sequence a b b c d b a, $URD(a, 1) = 3$ and $ARD(a, 1) = 5$.

4.3.1 Reuse Distance Distributions

Our study is concerned with average-case behavior. So instead of focusing on each individual point in T, we characterize it using probability vectors that reflect average/expected distributions.

- The **unique reuse distance distribution** of trace T is a probability distribution that we denote by row vector $\mathbf{r}(T)$ such that the k^{th} component,
  $\mathbf{r}_k(T) = P(URD(x, m) = k), \forall (x, m) \in image(T)$.

- The **expected absolute distance distribution** of trace T is a row vector that we denote by $\mathbf{d}(T)$ such that the k^{th} component,
  $\mathbf{d}_k(T) = E(ARD(x, m) | URD(x, m) = k), \forall (x, m) \in image(T)$

4.3.2 $T \rightarrow \mathbf{r}(T)$ is a Lossy Transformation

The characterization is lossy in the sense that in general, T cannot be recovered from $\mathbf{r}(T)$ even up to permutation of entity identifiers.

Consider two traces T_A and T_B such that they have disjoint sets of entities and different values of reuse metrics. Let T_{AB} denote a new trace formed from concatenating, in order, sequences represented by T_A and T_B. This operation is not commutative, that is, T_{AB} and T_{BA} are distinct, yet have the same values for the reuse metrics. So the reverse

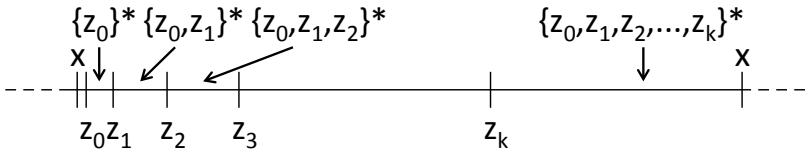


Figure 4.2: Unique elements z_0, z_1, ... z_{k-1} partition $\mathbf{d}_k(t)$ into subintervals.

mapping from $\mathbf{r}(T)$ to T is not unique. The argument can be extended to show that any trace characterization using position-agnostic metrics must be lossy.

4.3.3 $\mathbf{d}(T)$ Estimation

It is obvious that $ARD(x, m) \geqslant URD(x, m), \forall x, m$. It then follows that $\mathbf{d}_k(T) \geqslant k, \forall k$ such that $\mathbf{r}_k(T) > 0$. Also, $\mathbf{d}_0(T) = 0$. We now show how to compute (an approximation to) $\mathbf{d}(T)$ given $\mathbf{r}(T)$.

Figure 4.2 shows a schematic of a trace and organization of URDs within a reuse interval for some address x. z_0, z_1,...z_k denote distinct addresses. This is just a conceptual tool and does not constrain the actual permutation of addresses in a particular reuse interval. The immediate next access after reference address x must be something other than x (otherwise the reuse interval would immediately terminate with $k = 0$). Between this first address z_0 and the next different address z_1, the only possible URDs of accesses must be 0. Between z_1 and z_2, the only possible URDs can be 0 and 1. Extending this reasoning till z_{k-1} and z_k we observe that $\mathbf{d}_k(T)$ and $\mathbf{d}_{k-1}(T)$ differ only in the last sub-sequence which consists of a run of accesses with URDs in $\{0, 1, .., k-1\}$. We approximate the length of this run with the expected number of trials to success in a geometric distribution with success probability $\sum_{i=k}^{\infty} \mathbf{r}_i(T)$.

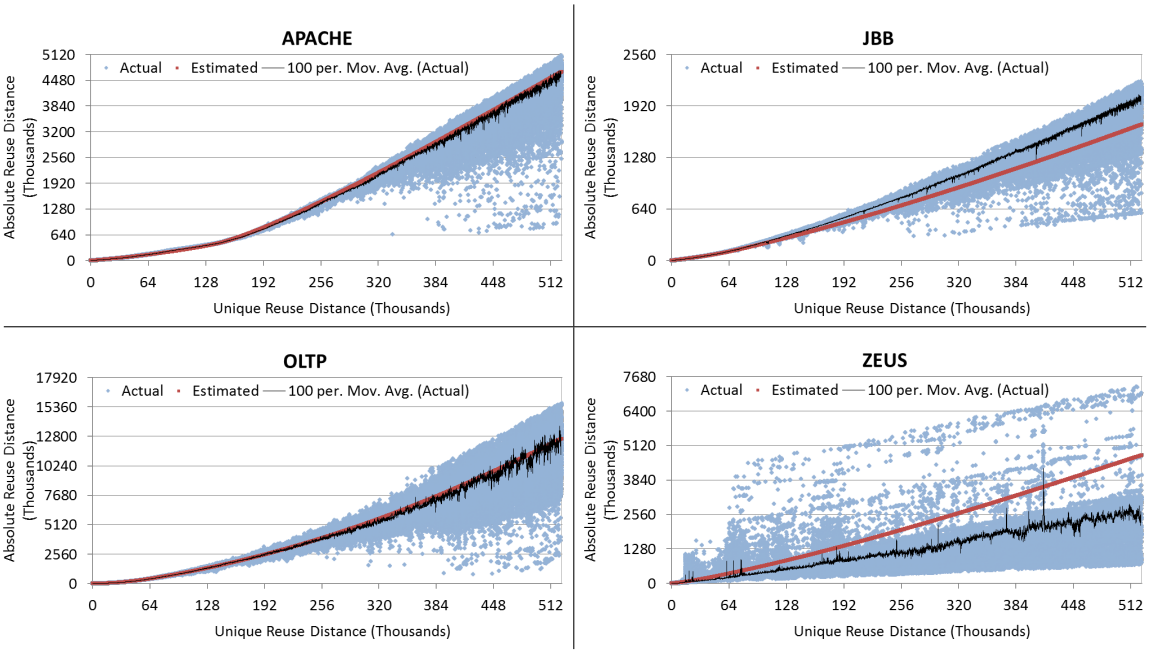


Figure 4.3: Actual, Moving Average (window size = 100) of Actual, and Estimated $\mathbf{d}(T)$.

We thus arrive at the following recurrence:

$$\begin{aligned} \mathbf{d}_0(T) &= 0 \\ \mathbf{d}_k(T) &= \mathbf{d}_{k-1}(T) + \frac{1}{\sum_{i=k}^{\infty} \mathbf{r}_i(T)} \end{aligned} \tag{4.1}$$

Expanding the recurrence gives us

$$\mathbf{d}_k(T) = \sum_{j=1}^{k} \frac{1}{\sum_{i=j}^{\infty} \mathbf{r}_i(T)} = \sum_{j=1}^{k} \frac{1}{1 - \sum_{i=0}^{j-1} \mathbf{r}_i(T)}$$

This is similar to known approximations for the coupon collector's problem assuming a given order of coupons [34]. We find good agreement in trends between observed (Moving Average) and estimated values of $\mathbf{d}(T)$ as illustrated in Figure 4.3. The moving

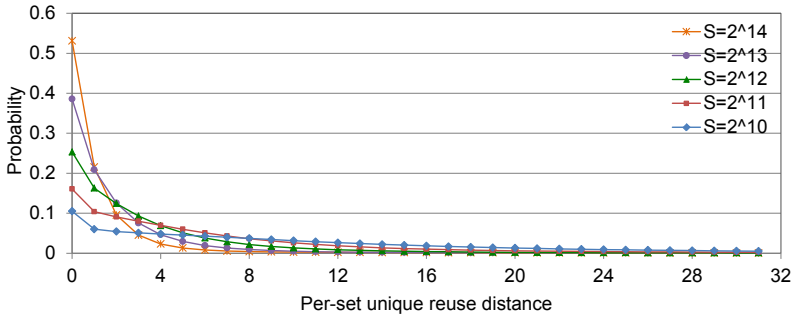


Figure 4.4: Effect of the number of sets (S) on per-set locality for `oltp`.

average calculation acts like a low-pass filter that removes short-term variation to reveal long-term trends. Our estimates are good for workloads with long traces, e.g., `apache` and `oltp`. We expect the differences to reduce for `jbb` and `zeus` with longer traces.

4.4 Per-set Locality

Replacement policy decisions (determining which cache line to evict) in traditional caches happen for each individual set. This in turn influences the miss ratio. So it is essential to determine the locality in the address stream that each individual set sees on average. We refer to the temporal locality in the per-set address stream as the per-set locality.

Per-set locality is strongly influenced by the number of sets (S) in the cache. The set of *unique* addresses in the address stream is split among the sets based on the index mapping function. The address stream that any individual set sees is the subset of the original address stream consisting of *all* accesses to the addresses mapping to that set. S thus determines the degree to which the address stream is split. Decreasing S increases URDs of the per-set address streams since addresses that hitherto mapped to other sets now get mapped to the reference set, and vice versa.

Accordingly, we extend our previous notations of locality metrics to additionally

include S as a parameter. Thus $\mathbf{r}(T, S)$ denotes the unique reuse distribution of the sub-sequence of T that a single set in the cache observes on average. $\mathbf{r}(T, 1)$ is the temporal locality of the original address stream, which is also the per-set locality of a fully-associative cache ($S = 1$). Figure 4.4 illustrates how $\mathbf{r}(T, S)$ changes with S for `oltp`. $\mathbf{d}(T)$ is adapted to $\mathbf{d}(T, S)$ similarly and can be estimated from $\mathbf{r}(T, S)$ using Equation 4.1. For brevity of notation, we will omit specifying one or more parameters when their values are clear from the context.

As Table 4.1 shows, cache configurations in our study have a range of number of sets (2^{10} to 2^{18}). For efficiently predicting miss ratios it is essential to be able to determine how $\mathbf{r}(S)$ can be transformed to $\mathbf{r}(S')$ for $S' \neq S$. The rest of this section develops a (new) methodology for this.

4.4.1 $\mathbf{r}(S')$ Estimation

For set-associative caches with $S' > 1$ we make the simplifying assumption, similar to Smith's model [103, 200], that the mapping of unique lines to cache sets are independent of each other. While this assumption does not always hold with the traditional bit selection index function, some processors use simple XOR hashing functions that increase uniformity [134]. The uniformity assumption enables both the following model and the use of uniform set-sampling techniques.

Accesses to a given set can thus be modeled as successive Bernoulli trials with the success of each trial having probability $\frac{1}{S'}$. While computing $\mathbf{r}(S')$ from $\mathbf{r}(1)$, we note that $\mathbf{r}_j(S')$ is the sum of the probability of exactly j successes (j addresses mapping to the reference set) from $\mathbf{r}_k(1)$, $\forall k$. The generalized stochastic Binomial Matrix [212] $\mathbf{B}(x, y)$ has the value ${}^kC_j y^j x^{k-j}$ in row k, column j, where ${}^kC_j$ denotes the j^{th} binomial coefficient and $x + y = 1$. This is the same as the probability of exactly j successes in k Bernoulli

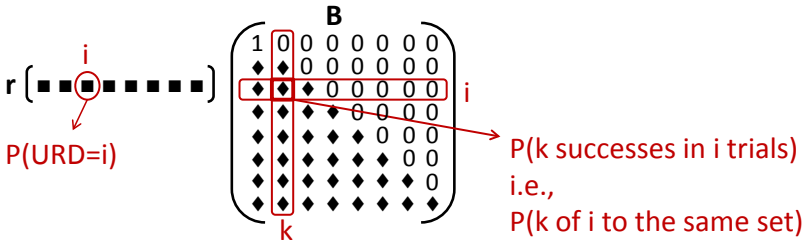


Figure 4.5: Reuse distribution transformations with stochastic Binomial Matrices.

trials with probability of each success being y. Viewing the computation of $\mathbf{r}(S')$ from $\mathbf{r}(1)$ through the lens of matrix multiplication, we recognize that the transformer is a generalized stochastic Binomial Matrix, $\mathbf{B}(1-\frac{1}{S'},\frac{1}{S'})$. Thus,

$$\mathbf{r}(S') = \mathbf{r}(1) \cdot \mathbf{B}(1-\frac{1}{S'},\frac{1}{S'}) \tag{4.2}$$

Figure 4.5 shows a schematic of the transformation. The transformer, B, is always a lower triangular matrix.

It is straight-forward to show that the transformation respects $\sum_{i=0}^{\infty}\mathbf{r}_i(S') = \sum_{i=0}^{\infty}\mathbf{r}_i(1) = 1$. Qualitatively, this transformation results in a re-distribution of mass with $\mathbf{r}(S')$ getting compressed as S' is increased and dilated as S' is decreased (see Figure 4.4).

We will now show how to compute $\mathbf{r}(S')$ from any starting cache configuration S. This shows how computations can be reused instead of always needing to start from the ground configuration ($S = 1$) and will also be useful in reasoning about way-counters (Section 4.6.2).

Binomial Matrices are invertible (when the second parameter is non-zero) and closed under multiplication within the same dimension [212]. Using identities $\mathbf{B}(x,y)\mathbf{B}(w,z) =$

$\mathbf{B}(x + yw, yz)$ and $\mathbf{B}(x, y)^{-1} = \mathbf{B}(-xy^{-1}, y^{-1})$, [212], we get

$$\begin{aligned}
\mathbf{r}(S') &= \mathbf{r}(1) \cdot \mathbf{B}(1 - \frac{1}{S'}, \frac{1}{S'}) \\
&= \mathbf{r}(S) \cdot (\mathbf{B}(1 - \frac{1}{S}, \frac{1}{S}))^{-1} \cdot \mathbf{B}(1 - \frac{1}{S'}, \frac{1}{S'}) \\
&= \mathbf{r}(S) \cdot \mathbf{B}(1 - \frac{S}{S'}, \frac{S}{S'}) \qquad (4.3)
\end{aligned}$$

Equation 4.3 is a general form of Equation 4.2. The transformer depends only on the ratio of the number of the sets in the current cache to that in the target cache. There are two cases to consider depending on the value of this ratio:

Case 1, $\mathbf{S'} \geqslant \mathbf{S}$: The transformation is always safe in that the computed probabilities are valid ($\in [0, 1]$) *even if* $\mathbf{r}(S)$ *has not been computed binomially*. Moreover, this allows intermediate steps; for example, computing $\mathbf{r}(2^{14})$ from $\mathbf{r}(1)$ is equivalent to first computing $\mathbf{r}(2^{10})$ from $\mathbf{r}(1)$ and then computing $\mathbf{r}(2^{14})$ from $\mathbf{r}(2^{10})$. This provides an opportunity to *reuse intermediate computations*. So, $\mathbf{r}(S)$ can be computed once from $\mathbf{r}(1)$ for the smallest S (2^{10} in our study, see Table 4.1) and used for all other target configurations.

Case 2, $\mathbf{S'} < \mathbf{S}$: Since $\mathbf{B}(1 - \frac{S}{S'}, \frac{S}{S'}) = (\mathbf{B}(1 - \frac{S'}{S}, \frac{S'}{S}))^{-1}$, Case 2 transforms can invert Case 1 transforms provided Case 1 results have not been truncated (see below). Otherwise, the computed probabilities may not be valid ($\notin [0, 1]$).

For an example transformation, consider $S' = 2S$. The components of the transformed reuse distribution, r', are computed as:

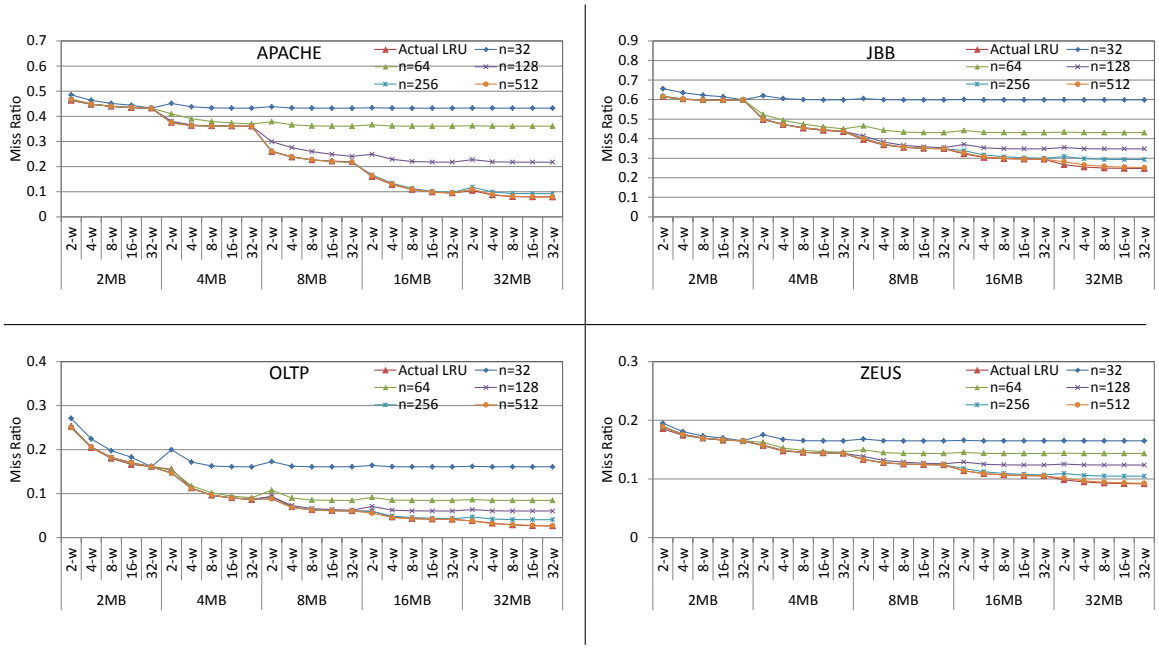


Figure 4.6: LRU prediction with reuse information limited to length n at $\mathbf{r}(2^{10})$ which is first computed from $\mathbf{r}(1)$ (Equation 4.2).

$$
\begin{aligned}
r'_0 = \quad & r_0 + \quad (1/2)\cdot r_1 + \quad (1/4)\cdot r_2 + \quad (1/8)\cdot r_3 + \quad \ldots \\
r'_1 = \quad & \quad\quad (1/2)\cdot r_1 + \quad (2/4)\cdot r_2 + \quad (3/8)\cdot r_3 + \quad \ldots \\
r'_2 = \quad & \quad\quad\quad\quad\quad (1/4)\cdot r_2 + \quad (3/8)\cdot r_3 + \quad \ldots \\
r'_3 = \quad & \quad\quad\quad\quad\quad\quad\quad\quad (1/8)\cdot r_3 + \quad \ldots \\
\ldots \quad &
\end{aligned}
$$

4.4.2 Matrix dimension and Truncation of $\mathbf{r}$

The dimension of $\mathbf{B}$ is determined by the maximum (per-set) URD that we are interested to maintain to avoid large computational costs. Let n denote the length of the $\mathbf{r}$ vector

that we maintain. That is, $\mathbf{r}(S)$ is computed for $\mathbf{r}_0(S)$ through $\mathbf{r}_{n-1}(S)$, with $\mathbf{r}_\infty(S)$ adjusted so that $\mathbf{r}_\infty(S) = 1 - \sum_{i=0}^{n-1} \mathbf{r}(S)$.

Assume $\mathbf{r}(2^{10})$ is available, computed from $\mathbf{r}(1)$ using Equation 4.2. Figure 4.6 shows predicted miss ratios with $\mathbf{r}(2^{10})$ maintained for various values of n. Section 4.5.1 explains LRU prediction. Although the maximum associativity that we consider is 32, Figure 4.6 shows that n has to be much larger than that ($\geqslant 512$) for good predictions for larger caches with $S' > 2^{10}$, such as 32MB caches (see Table 4.1).

While $n = 512$ is good for $\mathbf{r}(2^{10})$, the equivalent value for $\mathbf{r}(1)$ is very large, potentially up to $512 \cdot 2^{10}$. To appreciate this, consider the $\mathbf{r}(1)$ address stream as a merger of the 2^{10} mutually exclusive per-set address streams, each of which has reuse intervals of up to 512. Determining the long-tailed $\mathbf{r}(1)$ distribution or using large matrices to compute $\mathbf{r}(2^{10})$ from $\mathbf{r}(1)$ in software is time-consuming. Section 4.6.1 proposes low-cost hardware support to approximately estimate $\mathbf{r}(2^{10})$ with $n = 512$.

4.4.3 Poisson approximation to Binomial

Cypher [55, 56] uses a Poisson approximation to binomial for reducing computational costs – when i is large and $\frac{1}{S'}$ is small, the binomial distribution can be approximated by a Poisson distribution with parameter $\lambda = \frac{i}{S'}$. Computing this is faster than the binomial coefficient.

Figure 4.7 shows pseudo-code for the `compute_per_set_r` function that computes Equation 4.3. It uses Poisson approximation to Binomial and assumes that $\mathbf{r}(2^{10})$ up to $n = 512$ is available. $num\_set\_bits \in [1, 8] = \log_2(\frac{S'}{S})$. The computation is done for $2A'$ terms (see Section 4.5).

```
void init() {
    int i;
    for(i=0;i<9;i++)
        precomputed_exp_inc[i]=exp(-1.0/(1<<i));
    for(i=1;i<64;i++)
        precomputed_v[i]=1.0/i;
}

void compute_per_set_r(int num_set_bits, int max_assoc) {
    const double *ptr=&r_histogram[0];
    double s3=precomputed_exp_inc[num_set_bits];
    double s2=1.0;
    double base_lambda=1.0/(1<<num_set_bits);
    double lambda=0;
    int i, rd;
    for(i=0;i<512;i++) {
        double s1=s2;
        for(rd=0;rd<2*max_assoc;rd++) {
            per_set_r[num_set_bits][rd]+=ptr[i]*s1;
            s1*=lambda*precomputed_v[rd+1];
        }
        s2*=s3;
        lambda+=base_lambda;
    }
}
```

Figure 4.7: Equation 4.3 pseudo-code with Poisson approximation.

4.5 Cache Hit Functions

Given a target cache organization $(S', A', \mathtt{policy})$ and a trace T, our goal is to determine a vector $\boldsymbol{\phi}(\mathbf{r}(S'), S', A', \mathtt{policy})$ such that the expected hit ratio for the trace is

$$h = \mathbf{r} \cdot \boldsymbol{\phi} \tag{4.4}$$

The idea is to characterize workload traces by $\mathbf{r}$ and caches by $\boldsymbol{\phi}$ so that the effect on hit ratio for changes in traces or cache configurations can be readily estimated.

We call $\boldsymbol{\phi}$ the cache hit function. The value of the k^{th} component, $\boldsymbol{\phi}_k$ is the conditional

probability of a hit for accesses x such that $\mathrm{URD}(x, m) = k$ where m is the repetition count for x at that point in the trace when the access happens. $\boldsymbol{\phi}_k$ monotonically decreases with k in this model. This is because non-eviction of a cache-resident address after accesses involving k other unique addresses implies non-eviction after accesses involving $k-1$ unique addresses *and* the remaining accesses. If there are no intervening accesses ($k = 0$), the access must be a hit. Accesses hitherto never seen ($k = \infty$) must miss. So,

$$\boldsymbol{\phi}_k = \begin{cases} 1 & \text{if } k = 0 \\ \leqslant \boldsymbol{\phi}_{k-1} & \text{if } k \geqslant 1 \\ 0 & \text{at } k = \infty \end{cases} \tag{4.5}$$

Figure 4.8 shows $\boldsymbol{\phi}$ curves for common replacement policies. We consider the well-known, but rarely-implemented[1] LRU policy as well as the practical RANDOM, NMRU, and PLRU policies. In each figure, we superimpose the $\boldsymbol{\phi}$ curves for $S' = 2^{14}$ and $A' = 2$, 4, 8, 16, and 32. These five curves appear from left to right, in that order, in each figure. $\boldsymbol{\phi}(\mathrm{LRU})$ is always a step function, with the 1-to-0 transition happening at A'. We show $\boldsymbol{\phi}(\mathrm{LRU})$ on each figure for comparison. Note that $\boldsymbol{\phi}$ for RANDOM, NMRU, PLRU are non-zero beyond A'. So, computing $\mathbf{r} \cdot \boldsymbol{\phi}$ up to A' is not sufficient for these replacement policies. For our evaluations, we compute the dot-product for $2A'$ terms; longer than that has diminishing returns for our workloads.

Apart from LRU, $\boldsymbol{\phi}$ is not independent of $\mathbf{r}$ for different replacement policies. As we shall show later, $\boldsymbol{\phi}(\mathrm{RANDOM})$ depends on $\mathbf{d}$, while $\boldsymbol{\phi}(\mathrm{PLRU})$ may need more information.

[1] LRU is typically not implemented in real caches for associativity larger than 4 due to hardware complexity.

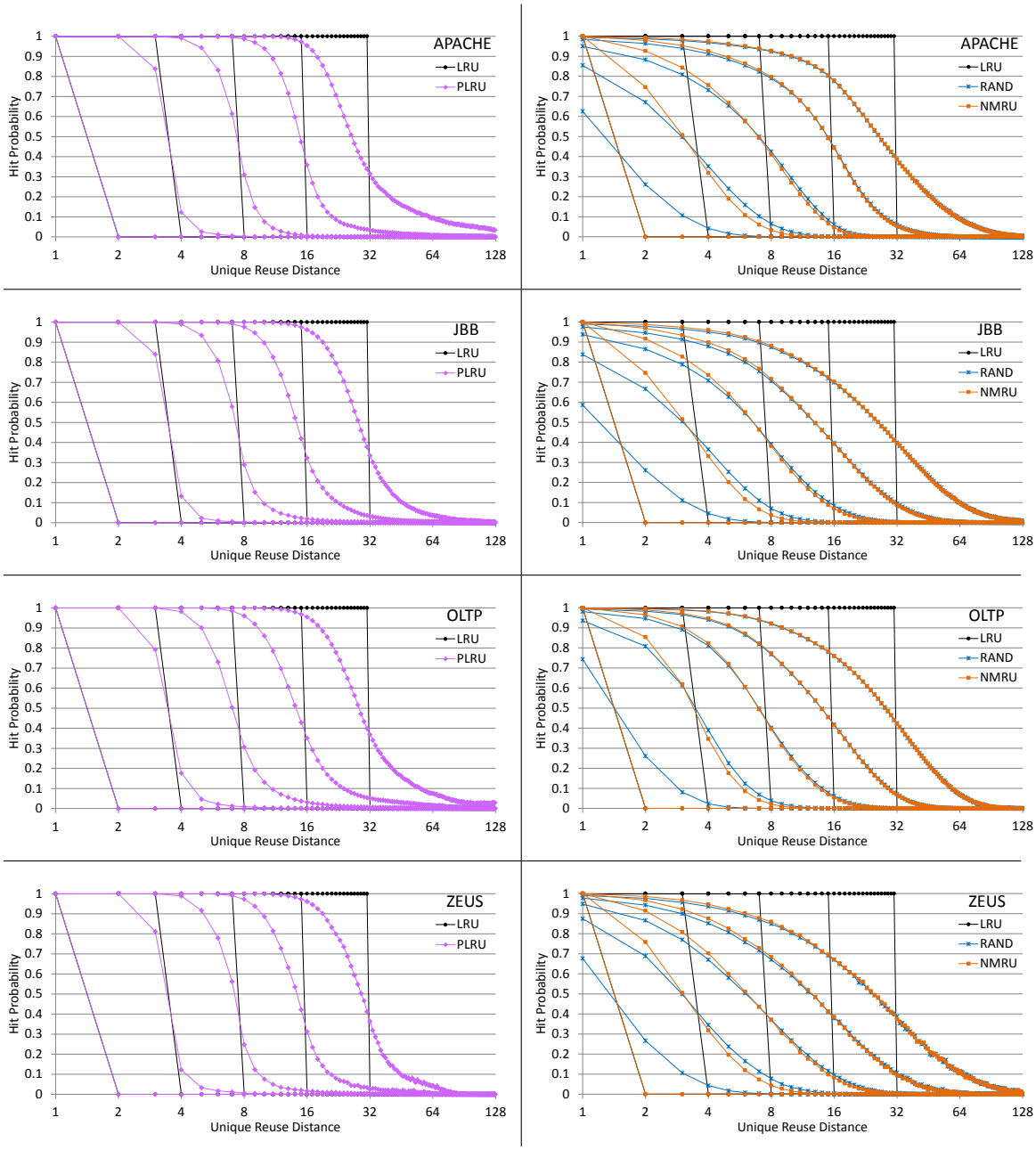


Figure 4.8: Representative hit ratio functions (Φ_k) with $S' = 2^{14}$ and $A' = 2, 4, 8, 16, 32$ with different replacement policies. $\Phi_0(\text{not shown}) = 1$ for all policies. Note that the x-axes are in $\log_2$ scale. Only LRU has a step function for any associativity.

4.5.1 Estimating $\phi(\mathbf{LRU})$

For a set-associative LRU cache with associativity A', it is well known that all accesses with addresses re-appearing with less than A' unique intervening elements must hit and all other accesses must miss. This leads us to the following characterization of the LRU hit ratio function.

$$\boldsymbol{\phi}_k(\mathrm{LRU}) = \begin{cases} 1 & \text{if } 0 \leqslant k < A' \\ 0 & \text{if } k \geqslant A' \end{cases} \tag{4.6}$$

Figure 4.6 shows actual vs estimated ($n = 512$) miss ratios with LRU using Equations 4.3, 4.6 and 4.4. As observed earlier by Hill and Smith [103], increasing A' yields diminishing returns.

4.5.1.1 Optimization (Smith's Model)

A naive combination of Equations 4.4, 4.6 and 4.3 results in $\left(\sum_{i=0}^{A'-1}(n-i)\right) - 1 = nA' - \frac{A'(A'-1)}{2} - 1$ multiplications with binomial computations to estimate the hit ratio for a cache with S' sets and associativity A'. The number of multiplications can be reduced by observing that due to the step-function nature of $\boldsymbol{\phi}(\mathrm{LRU})$, some of the coefficients will sum to 1. Expanding the computation and simplifying, we get

$$h(S') = \sum_{i=0}^{A'-1} \mathbf{r}_i(1) + \sum_{i=A'}^{n-1} \mathbf{r}_i(1) \cdot \sum_{k=0}^{A'-1} {}^{i}C_k \cdot \left(\frac{1}{S'}\right)^k \cdot \left(1 - \frac{1}{S'}\right)^{(i-k)} \tag{4.7}$$

Equation 4.7 is an optimized version of Smith's associativity model [103, 200]. It requires $(n - A')A'$ multiplications which is $\frac{A'(A'+1)}{2} - 1$ less than the naive combination. But computing binomial terms is costly and n is usually much larger than A', so the gains from this optimization are small.

4.5.2 Estimating ϕ(RANDOM)

The RAND replacement algorithm [25] (also popularly called RANDOM) chooses a line (uniformly) randomly from the lines in the set for eviction on a miss.

For an A'-way set-associative cache, the probability of replacement of a given line on a miss is $\frac{1}{A'}$. Accounting for the number of misses in between successive reuses of an address is therefore needed. For expected miss rate θ, the expected number of misses for a sequence of α accesses is $\alpha \cdot \theta$. This is why $\mathbf{d}$ is important for RANDOM whereas LRU works independent of such information.

We make the simplifying assumption that miss occurrences (not specific addresses) are independent and hence amenable to be modeled as a Bernoulli process. While this may not be accurate, it allows us to make reasonably good predictions without tracking additional state.

Let $\mathbf{d}_k = \alpha$. The probability of i misses is estimated by ${}^{\alpha}C_i \cdot \theta^i \cdot (1-\theta)^{(\alpha-i)}$. The probability that a specific line is not replaced after i misses is $\left(1 - \frac{1}{A'}\right)^i$. We thus have

$$\begin{aligned} h(\text{RANDOM}) &= \mathbf{r} \cdot \boldsymbol{\phi}(\text{RANDOM}) \\ \boldsymbol{\phi}_k(\text{RANDOM}) &= \sum_{i=0}^{\alpha | \mathbf{d}_k = \alpha} {}^{\alpha}C_i \cdot \theta^i \cdot (1-\theta)^{(\alpha-i)} \cdot \left(1 - \frac{1}{A'}\right)^i \\ \theta &= 1 - h(\text{RANDOM}) \end{aligned} \tag{4.8}$$

To simplify the computation, we approximate Binomial(α,θ) by Poisson($\lambda = \alpha \cdot \theta$). Let $q = \left(1 - \frac{1}{A'}\right)$. This gives

$$\begin{aligned}
\boldsymbol{\phi}_k(\text{RANDOM}) &= \sum_{i=0}^{\alpha|\mathbf{d}_k=\alpha} {}^{\alpha}C_i \cdot \theta^i \cdot (1-\theta)^{(\alpha-i)} \cdot q^i \\
&= \sum_{i=0}^{\infty} {}^{\alpha}C_i \cdot \theta^i \cdot (1-\theta)^{(\alpha-i)} \cdot q^i \\
&\approx \sum_{i=0}^{\infty} e^{-\lambda} \cdot \frac{\lambda^i}{i!} \cdot q^i \\
&= e^{-\lambda(1-q)} \sum_{i=0}^{\infty} e^{-\lambda q} \cdot \frac{(\lambda q)^i}{i!} \\
&= e^{\frac{-\alpha\theta}{A'}} \qquad (4.9)
\end{aligned}$$

The system of equations in 4.8 can now be approximated by the following system.

$$\begin{aligned}
h(\text{RANDOM}) &= \mathbf{r} \cdot \boldsymbol{\phi}(\text{RANDOM}) \\
\boldsymbol{\phi}_k(\text{RANDOM}) &= e^{\frac{-\mathbf{d}_k\theta}{A'}} \\
\theta &= 1 - h(\text{RANDOM}) \qquad (4.10)
\end{aligned}$$

We solve the system of equations in 4.10 with the initial value $h = \mathbf{r}_0$. $\mathbf{d}$ is estimated using equation 4.1. Usually 5 or fewer iterations suffice to reach within 1% of a fix-point.

4.5.2.1 Convergence for RANDOM

First note that if a fix-point exists, the solution satisfies the general conditions of $\boldsymbol{\phi}$ (Equation 4.5). This is because $\mathbf{d}_0 = 0$ (Equation 4.1) and from Equation 4.10,

$$\begin{aligned}
\frac{\boldsymbol{\phi}_k}{\boldsymbol{\phi}_{k-1}} &= e^{\frac{-(\mathbf{d}_k-\mathbf{d}_{k-1})\theta}{A'}} = e^{-\left(\frac{\theta/A'}{\sum_{i=k}^{\infty} r_i(T)}\right)} \\
&\leqslant 1 \qquad (4.11)
\end{aligned}$$

Let H denote a fix-point and h^0, h^1, h^2, ... denote successive approximations. By

re-arranging the system of equations in 4.10 we have

$$h^{j+1} = r_0 + \sum_{i=0}^{n-1} r_i e^{\frac{d_i(-1+h^j)}{A'}} \tag{4.12}$$

Since the exponential function is monotonic, H must be unique. Since $0 \leqslant r_i \leqslant 1, \forall i$, $r_0 \leqslant H \leqslant 1$.

Also, it is easy to show that $h^j \geqslant h^{j-1} \implies h^{j+1} \geqslant h^j$. Thus, successive iterations produce a chain of values $r_0 = h^0 \leqslant h^1 \leqslant h^2$....

We will now prove that $h^j \leqslant H, \forall j$. This is true at $j = 0$. For induction, let $h^j = H - \epsilon$ with $\epsilon \geqslant 0$. Then,

$$\begin{aligned} h^{j+1} &= r_0 + \sum_{i=0}^{n-1} r_i e^{\frac{d_i(-1+h^j)}{A'}} \\ &= r_0 + \sum_{i=0}^{n-1} r_i e^{\frac{d_i(-1+H)}{A'}} \cdot e^{-\frac{d_i\epsilon}{A'}} \\ &\leqslant r_0 + \sum_{i=0}^{n-1} r_i e^{\frac{d_i(-1+H)}{A'}} = H \end{aligned} \tag{4.13}$$

This shows a convergence chain $r_0 = h^0 \leqslant h^1 \leqslant h^2 ... \leqslant H$.

4.5.2.2 Optimization

A better approximation for $A' = 2$ can be obtained by using the fact that for the reference element not to be evicted at $URD \geqslant 2$, the previous element must be evicted (since the set can hold only 2 elements). The probability of the previous element to be evicted is $1 - \phi_1$. For the reference element to hit at $URD = k$, it must hit at $URD = k - 1$ and the

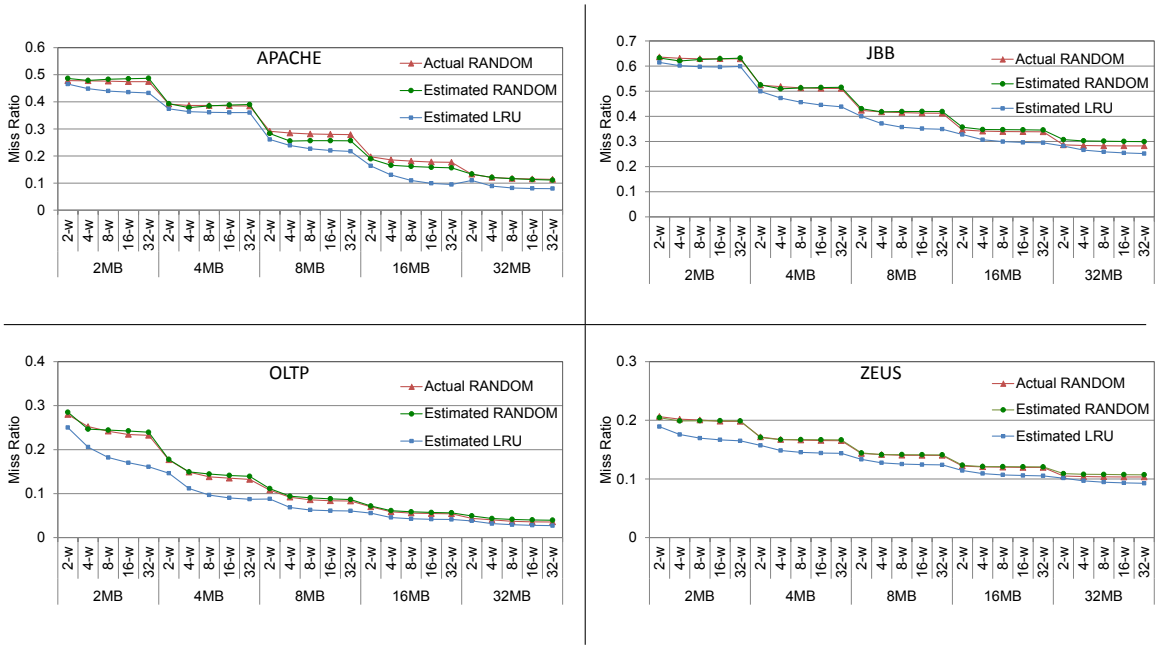


Figure 4.9: Actual vs estimated miss ratios with RANDOM replacement policy. LRU estimates are shown as reference. $\mathbf{r}(2^{10})$ is first computed from $\mathbf{r}(1)$ (Equation 4.2).

above condition must hold. This leads us to the following approximation.

$$\boldsymbol{\phi}_k \quad = \quad \boldsymbol{\phi}_{k-1} \cdot (1 - \boldsymbol{\phi}_1), \quad k \geqslant 2 \tag{4.14}$$

This approximation is possible since the model can exactly determine the set contents for URD $>= 2$. For higher associativities, exact determination of set contents is difficult.

Figure 4.9 shows actual vs estimated ($n = 512$) values of miss ratios for RANDOM with the estimates computed using Equations 4.3, 4.10, 4.14 and 4.4.

4.5.3 Estimating $\boldsymbol{\phi}$(NMRU)

The NMRU (or non-MRU) replacement algorithm differentiates the most recently accessed (MRU) line from other lines in the set [203]. On a miss, a line is chosen (uniformly)

randomly from among the $A' - 1$ non-MRU lines.

At $A' = 2$, $\boldsymbol{\phi}(\mathrm{NMRU}) = \boldsymbol{\phi}(\mathrm{LRU})$. For the rest of the cases, the framework is similar to that of RANDOM except that accesses at $\mathrm{URD} \leqslant 1$ are guaranteed to hit. Moreover, the replacement logic has $A' - 1$ possible choices for an eviction in case of a miss. This leads to a few simple modifications to the system of equations in 4.10. The modified system is shown below:

$$\begin{aligned} \boldsymbol{\phi}_1(\mathrm{NMRU}) &= 1 \\ h(\mathrm{NMRU}) &= \mathbf{r} \cdot \boldsymbol{\phi}(\mathrm{NMRU}) \\ \boldsymbol{\phi}_k(\mathrm{NMRU}) &= e^{\frac{-(\mathbf{d}_k - \mathbf{d}_1)\theta}{A'-1}} \\ \theta &= 1 - h(\mathrm{NMRU}) \end{aligned} \tag{4.15}$$

Figure 4.10 shows actual vs estimated ($n = 512$) values of miss ratios for NMRU with the estimates computed using Equations 4.3, 4.15 and 4.4.

4.5.4 Estimating $\boldsymbol{\phi}$(PLRU)

Partitioned LRU [203] (also popularly called pseudo-LRU) maintains a balanced binary tree that, at each level, differentiates between the two sub-trees based on access recency. Every internal node is represented by a single bit whose value decides which of the two subtrees was accessed more recently. The cache lines are represented by the leaves of the tree. Whenever a line is accessed, the nodes on the path from the root to the leaf flip their bit values, thus pointing to the other subtree at each level. On a miss, the subtree pointed to is chosen, recursively starting from the root. The line corresponding to the leaf reached in this way is chosen for eviction. The bit-values along this path are then flipped.

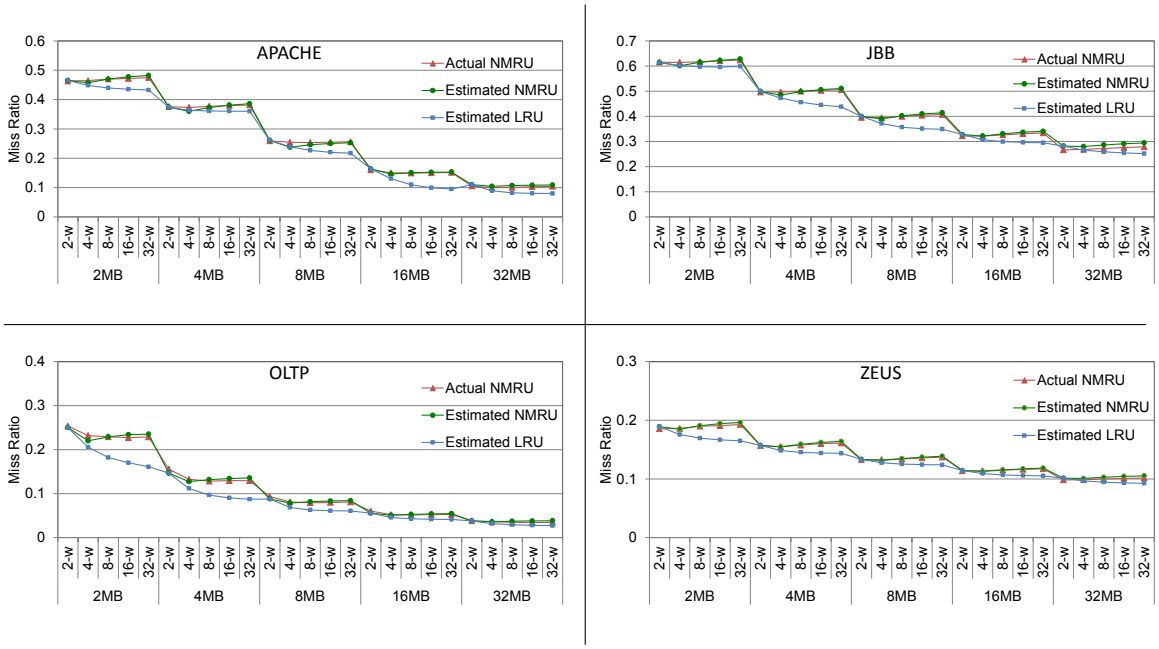


Figure 4.10: Actual vs estimated miss ratios with NMRU replacement policy. LRU estimates are shown as reference. $\mathbf{r}(2^{10})$ is first computed from $\mathbf{r}(1)$ (Equation 4.2).

In the PLRU scheme, the most recently accessed element is always known but the least recently accessed one is not. In contrast to the LRU scheme, that maintains a total access order between the lines, PLRU maintains only a partial order. Since there is no difference between partial and total orders involving 2 elements, PLRU is LRU when $A' = 2$. In contrast to LRU that guarantees exactly $A' - 1$ unique accesses before eviction, PLRU guarantees at least $\log_2(A')$ (=number of tree levels) unique accesses before the reference address is evicted.

Since the PLRU tree is symmetric, we can fix any way as reference without loss of generality. Let the immediate neighbor be denoted by Q_0, the next two neighbors be collectively denoted by Q_1 and so on with the most distant group of $A/2$ neighbors denoted by $Q_{\log_2(A)-1}$. To calculate the probability that the reference line will be evicted

on a particular miss we need to consider the immediate past sequence of accesses to that set. A necessary and sufficient condition for the reference line to be evicted is for the suffix of the trace to have accesses that match the particular regular expression described below.

$$
\begin{aligned}
A = 2 &: Q_0^+ \\
A = 4 &: Q_0 Q_1^+ \\
A = 8 &: Q_0 (Q_1 + Q_2)^* Q_1 Q_2^+ \\
A = 16 &: Q_0 (Q_1 + Q_2 + Q_3)^* Q_1 (Q_2 + Q_3)^* Q_2 Q_3^+ \\
A = 32 &: Q_0 (Q_1 + Q_2 + Q_3 + Q_4)^* Q_1 (Q_2 + Q_3 + Q_4)^* \\
&\quad Q_2 (Q_3 + Q_4)^* Q_3 Q_4^+
\end{aligned}
$$

On a miss, the reference line will be evicted if and only if the immediately preceding sequence of accesses follows a particular pattern. These patterns can be described using regular expressions. In contrast to RANDOM, not only the number of misses in the reuse interval, but also the pattern of accesses determines eviction probability. It is difficult to estimate $\boldsymbol{\phi}(\mathrm{PLRU})$ by computing probabilities of the regular expressions since the distance to misses within the reuse interval as well as the ways occupied by the intervening elements are not known. Instead, we use a different approach.

First, we compute $\boldsymbol{\phi}(A' = 4, \mathrm{PLRU})$ then compute $\boldsymbol{\phi}(A' = 8, \mathrm{PLRU})$ by dividing traffic using a binomial distribution and applying $\boldsymbol{\phi}(A' = 4, \mathrm{PLRU})$ on the divided traffic. We view an 8-way tree as a composition of two 4-way trees with the top-node dividing traffic between the two subtrees. Similar observations hold between 8-way and 16-way trees and so on. This helps us to estimate $\boldsymbol{\phi}(\mathrm{PLRU})$ for successively higher associativities. We assume that the top node divides traffic evenly between its two constituent sub-trees.

4.5.4.1 Base case: $A' = 4$

Since $\log_2(4) = 2$, ϕ_k is 1 when $k \leqslant 2$. Let x denote the reference element. We will now estimate the likelihood that the second occurrence of x in the access sequence $x\ e_1$ ... e_2 ... $e_k\ x$ will hit in the cache. The elements e_1 through e_k all map to the same cache set and are distinct so that the URD of the sequence is k. $\phi_k = \phi_{k-1} \cdot$ P(x not evicted by e_k).

First, consider the case when $k \geqslant 4$. To have $\phi_k = 1$, x must be present in the cache. Moreover, both e_{k-1} and e_k will also be in the cache as PLRU guarantees that the last two unique elements seen will remain in the cache. Since $A' = 4$, there is room for one more element other than x, e_{k-1}, and e_k in the cache set. This element must be e_{k-2} as it could not have been evicted by either e_{k-1} or e_k. Therefore, e_{k-3} must have been evicted by e_k. So if e_{k-3} were to reappear instead of the second occurrence of x in the above sequence, it would miss. That is, the access sequence e_{k-3} ... e_{k-2} ... e_{k-1} ... e_k ... e_{k-3} would have caused the second e_{k-3}, with URD 3, to miss. This probability is $(1 - \phi_3)$. Thus, P(x not evicted by e_k) = P(e_{k-3} evicted by e_k) = $(1 - \phi_3)$. So, $\phi_k = \phi_{k-1} \cdot (1 - \phi_3)$.

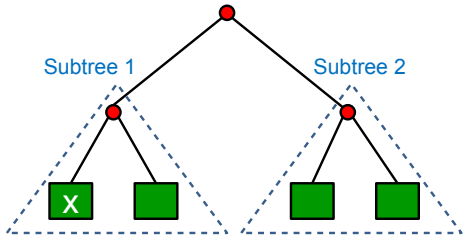


Figure 4.11: Schematic showing PLRU subtrees for $A' = 4$. Without loss of generality, we denote the subtree containing the reference element, x, as Subtree 1. Also, x is not necessarily the left-most child of Subtree 1.

The case that remains is when $k = 3$. For this case, we will refer to Figure 4.11 to

describe our estimation approach. The access sequence that we are considering is x e_1 ... e_2 ... e_3 x, with e_1, e_2, e_3 being distinct elements. The only scenario where x is evicted (by e_3) before its second occurrence occurs if all of the following conditions hold:

1. e_3 misses. e_3 has URD $\geqslant 3$. For an approximation, we just consider what would happen under LRU. It would miss for URD > 3. The probability for this happening is P(URD $> 3 \,|\, \text{URD} \geqslant 3$) = 1 - P(URD $= 3 \,|\, \text{URD} \geqslant 3$) = 1 - $\frac{\mathbf{r}_3}{1-\mathbf{r}_0-\mathbf{r}_1-\mathbf{r}_2}$.

2. The last access (to either e_2 or e_3) before the access to e_3 causes Subtree 2, not containing x, to be accessed. We assume this probability to be $\frac{1}{2}$.

3. e_1 and e_2 map to different subtrees. Since each subtree can have only 2 elements, Subtree 2 must get at least one of e_1 or e_2. Thus, the question is whether or not the other element (e_1 or e_2) maps to Subtree 1. We assume this probability to be $\frac{1}{2}$.

Thus, $\boldsymbol{\phi}_3 = 1 - \left(1 - \frac{\mathbf{r}_3}{1-\mathbf{r}_0-\mathbf{r}_1-\mathbf{r}_2}\right) \cdot \frac{1}{2} \cdot \frac{1}{2} = \frac{3}{4} + \frac{1}{4} \cdot \left(\frac{\mathbf{r}_3}{1-\mathbf{r}_0-\mathbf{r}_1-\mathbf{r}_2}\right)$.
Putting everything together,

$$\boldsymbol{\phi}_k = \begin{cases} 1 & \text{if } 0 \leqslant k \leqslant 2 \\ \frac{3}{4} + \frac{1}{4} \cdot \left(\frac{\mathbf{r}_3}{1-\mathbf{r}_0-\mathbf{r}_1-\mathbf{r}_2}\right) & \text{if } k = 3 \\ \boldsymbol{\phi}_{k-1} \cdot (1 - \boldsymbol{\phi}_3) & \text{if } k \geqslant 4 \end{cases} \tag{4.16}$$

4.5.4.2 Recurrence: $A' \geqslant 8$

Let $L = \log_2(A')$ and $\boldsymbol{\psi} = \boldsymbol{\phi}(A'/2)$. We will refer to Figure 4.12 to describe our estimation approach. For the first case, when $k \leqslant L$, $\boldsymbol{\phi}_k$ must be 1. For $k > L$, consider the element e_k in the access sequence x e_1 ... e_2 ... e_3 x. There are two subcases here:

1. It maps to Subtree 2. In this case, $\boldsymbol{\phi}_k = \boldsymbol{\phi}_{k-1}$.

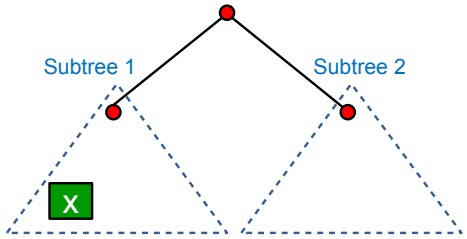


Figure 4.12: Schematic showing PLRU subtrees. Each subtree has $A'/2$ leaves. Without loss of generality, we denote the subtree containing the reference element, x, as Subtree 1.

2. It maps to Subtree 1. If $k \geqslant \frac{A'}{2} + 2$, there is at least one other element in Subtree 1 apart from x and e_k. This is because at most $\frac{A'}{2}$ elements can map to Subtree 2 before an element maps to Subtree 1. So, an element within the set $\{e_1 \ldots e_{\frac{A'}{2}+1}\}$ must map to Subtree 1. If $k \leqslant \frac{A'}{2} + 1$, the $\frac{A'}{2}$ elements other than $e_k$can all occupy Subtree 2.

   The remaining elements can map to Subtree 1 or Subtree 2. We use a Binomial distribution with success probability $\frac{1}{2}$ to estimate the likelihood of a certain number of them mapping to Subtree 1. This number plus 1 (for e_k) gives the URD for x considering only accesses to Subtree 1. We then get the hit probability for this URD from ψ.

Putting everything together,

$$\Phi_k = \begin{cases} 1 & \text{if } 0 \leqslant k \leqslant L \\ \dfrac{\Phi_{k-1}}{2} + \dfrac{1}{2}\displaystyle\sum_{i=0}^{k-3} {}^{k-3}C_i \left(\frac{1}{2}\right)^{(k-3)} \cdot \psi_{2+i} & \text{if } k \geqslant \dfrac{A'}{2} + 2 \\ \dfrac{\Phi_{k-1}}{2} + \dfrac{1}{2}\displaystyle\sum_{i=0}^{k-2} {}^{k-2}C_i \left(\frac{1}{2}\right)^{(k-2)} \cdot \psi_{1+i} & \text{otherwise} \end{cases} \tag{4.17}$$

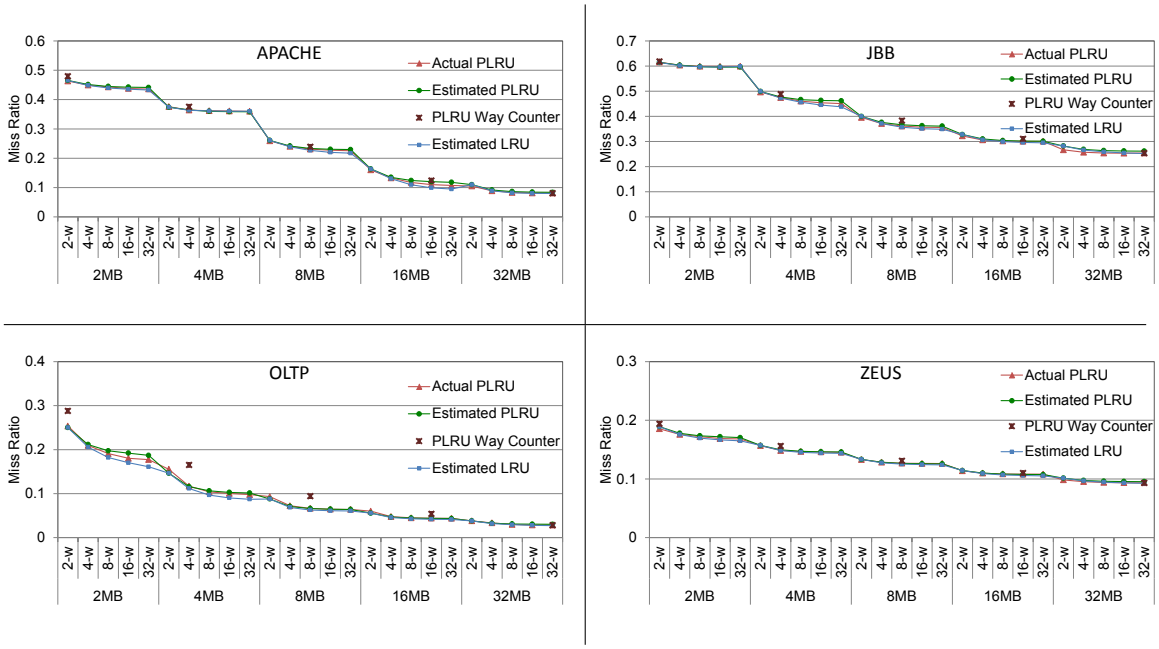


Figure 4.13: Actual vs estimated miss ratios with PLRU replacement policy. LRU estimates are shown as reference. $\mathbf{r}(2^{10})$ is first computed from $\mathbf{r}(1)$ (Equation 4.2). Section 4.6.2.1 describes PLRU Way-Counters.

In the above, we ignore the case when all of $\{e_1 \ldots e_k\}$ map to Subtree 1 along with x. This occurrence has a low probability since all of $\{e_1 \ldots e_k\}$ must have been hits (probability = P(hit)^k) and already have been present in Subtree 1 (probability = 2^{-k}).

Figure 4.13 shows actual vs estimated ($n = 512$) values of miss ratios for PLRU with the estimates computed using Equations 4.3, 4.16, 4.17 and 4.4.

4.5.5 Estimation Accuracy and Computation Time

Table 4.5 shows miss ratio prediction errors for different policies and workloads. LRU prediction is the most accurate, with relative errors < 2%, followed by PLRU with relative errors < 3%. Using the PLRU predictor instead of the LRU predictor when the actual cache uses PLRU improves prediction accuracy by ~2% for `oltp`. RANDOM and NMRU

Workload	LRU		RANDOM		NMRU		PLRU		LRU→PLRU	
	Abs.	*Rel.*	*Abs.*	*Rel.*	*Abs.*	*Rel.*	*Abs.*	*Rel.*	*Abs.*	*Rel.*
`apache`	1.23	0.81	5.12	2.67	4.68	2.27	3.41	2.31	3.40	2.23
`jbb`	3.40	1.12	7.80	2.27	6.58	1.90	5.24	1.59	4.44	1.29
`oltp`	1.59	1.85	4.04	4.90	3.77	4.05	2.88	2.97	4.88	5.18
`zeus`	0.69	0.57	2.68	1.96	1.78	1.36	1.21	0.96	1.55	1.19

Table 4.5: Average absolute values of prediction errors over all cache configurations. *Abs.* = (predicted - actual) miss ratio $\times 10^3$, *Rel.* = (predicted/actual - 1)$\times 10^2$ (to express as a percentage). LRU→PLRU shows what happens if the LRU predictor is used to predict for PLRU instead of using the PLRU predictor.

have relative errors < 5%.

A major contributor to hit ratio computation time is the determination of $\mathbf{r}$. Section 4.6.1 proposes low-cost hardware to approximate $\mathbf{r}(2^{10})$ (with $n = 512$) online. Assuming this is available, the hit ratio computation time per cache configuration on the Haswell machine (HS, at 3.9 GHz) were – LRU: $\leqslant$ 0.009 msec; PLRU: $\leqslant$ 0.011 msec; RANDOM: $\leqslant$ 0.012 msec; NMRU: $\leqslant$ 0.012 msec. On a Nehalem 2.26 GHz machine, the times were – LRU: $\leqslant$ 0.016 msec; PLRU: $\leqslant$ 0.018 msec; RANDOM: $\leqslant$ 0.023 msec; NMRU: $\leqslant$ 0.024 msec. This includes the time to compute $\mathbf{r}(S')$ from $\mathbf{r}(2^{10})$ (Equation 4.3), amortized over all configurations with the same S'.

4.6 Hardware Support

Section 4.4.2 discussed that to avoid expensive computation to determine $\mathbf{r}(1)$ or compute $\mathbf{r}(2^{10})$ from $\mathbf{r}(1)$, we need hardware support to directly estimate $\mathbf{r}(2^{10})$. Section 4.6.1 presents our proposed hardware technique to do this.

Section 4.6.2 discuss two traditional hardware mechanisms that help in cache miss ratio estimation—set-counters and way-counters.

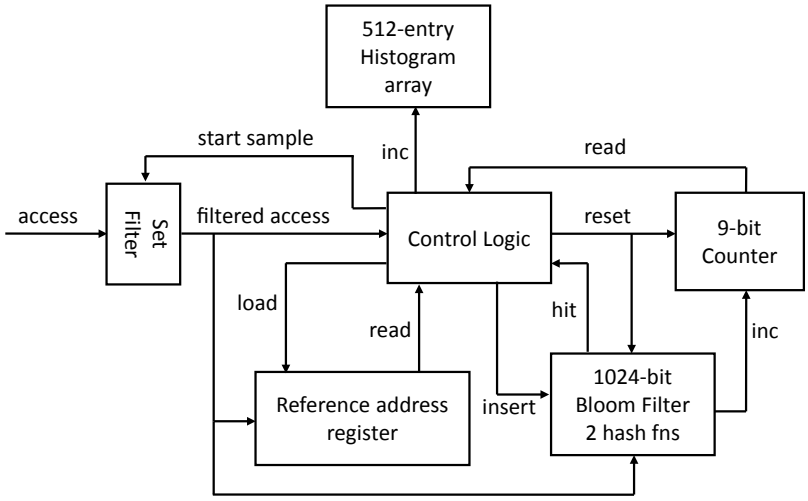


Figure 4.14: Schematic of new hardware support.

4.6.1 New hardware support to estimate reuse distributions ($\mathbf{r(2^{10}), n = 512}$)

The definition of unique reuse distance (URD) depends *only on the cardinality of the reuse interval (*RI*) and not on the contents of the set*. This suggests applicability of hardware signatures, such as Bloom filters [32], that can construct compact representations of sets. Whereas shadow tags store entire tag addresses, a Bloom filter uses only one bit per hash function to represent each address.

Our proposed hardware, shown in Figure 4.14, uses a Bloom filter (to summarize RI), a counter to determine |RI|, and set-sampling logic. We use a 1024-bit parallel Bloom filter [187] with two H_3 hash functions [42] and a 9-bit counter. The Bloom filter can be at most half-full (512 elements) before being reset. Larger Bloom filters can be used to reduce aliasing errors at the cost of more area/power overhead. The hardware uses a combination of set sampling and time sampling techniques [129, 132, 133, 172, 222]. It works as follows:

1. **Sample Initiation**: The Control Logic initializes the Set Filter to match a single set

of a cache with $S = 2^{10}$. It chooses this value by first time-sampling the incoming address stream (10% selectivity) and then choosing the set number of the chosen address as the value for the Set Filter. It also saves this address in the Reference address register.

2. **Sample Continuation**: The Control Logic inspects (see step 3) each address that passes through the Set Filter. Then it inserts the address into the Bloom Filter and increments the 9-bit Counter provided that the Bloom Filter does not return a match (already seen) for the address.

3. **Sample Termination**: This happens in one of two cases—(i) the reference address is seen again, or (ii) the maximum value (511) for the 9-bit counter is reached before inserting another new element. For case (i), the Control Logic increments the entry (whose position matches the 9-bit Counter value) in the Histogram array. For both cases, it transitions to Sample Initiation mode.

The above process is repeated. Each sequence of steps 1–3 estimates the reuse distance of a single address in the address stream. This value is between 0–511 (both inclusive) or considered as ∞ ($\geqslant 512$) otherwise. A separate counter (not shown) tracks the total number of measurements. This value, together with the histogram entries, is used to estimate $\mathbf{r}(2^{10})$.

The technique can be generalized to estimate $\mathbf{r}(2^{x})$ by sizing the Bloom filter, histogram, and set filter appropriately.

4.6.1.1 Bloom Filter Analysis

The Bloom Filter is used to estimate the number of unique addresses. Here we analyze its performance by comparing three kinds of filters:

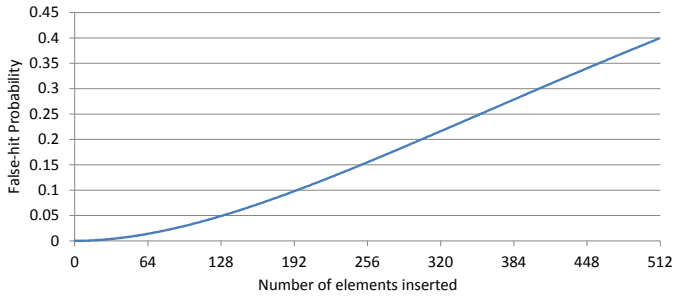


Figure 4.15: Probability of false hit in a 1024-bit Bloom filter with 2 hash functions.

1. **E** (Exact): This assumes that the addresses are fully tracked so that the estimation of the number of unique addresses is accurate.

2. **B** (Bloom): This uses traditional Bloom Filters. Addresses are not fully tracked, but represented by a few bits (2 bits in our study). Aliasing (same bits set for different addresses) may result in under-reporting of the number of unique addresses seen. This shortens the reuse distance measured.

3. **CB** (Bloom with Correction): This uses traditional Bloom Filters, but applies a correction term, based on the expected number of false aliases, to the measured reuse distance.

The aliasing probability for a traditional Bloom Filter (**B**) increases with the number of elements (addresses) inserted as more bits get set in the filter. For a Bloom Filter of size m bits and k hash functions, the probability of a false hit (alias) with n elements already inserted is given by

$$\text{P(false hit } \mid \text{ } n \text{ elements inserted)} \approx (1 - e^{-kn/m})^k$$

For our study, $m = 1024$ and $k = 2$. So the aliasing probability is

$$\text{P(false hit } \mid \text{ } n \text{ elements inserted)} \approx (1 - e^{-n/512})^2$$

Figure 4.15 plots this probability. We only plot till 511 elements since measurement is terminated beyond that and the reuse distance is considered as ∞.

To correct for this aliasing, the **CB** filter tracks the number of lookups for every state (number of elements already inserted) of the Bloom filter and computes an expectation of the total number of aliases. It then adds this count to the reuse distance measured. Note that the computation for the expected number of aliases may not match the number of *unique* aliases, so the correction is not exact.

For our analyses, the **E**, **B**, and **CB** filters use the same random numbers. However, the starting points of the samples can differ. This is because Sample Termination (followed by Sample Initiation) that happens when a long reuse distance ($\geqslant$ 512, equiv. ∞) is encountered, is affected by how accurately the reuse distance is calculated. So, individual samples across the three filters are not comparable. We compare the estimated miss ratios computed from sample results for the three filters.

Figures 4.16, 4.18, and 4.20 show the estimated miss ratios for LRU using **E**, **B**, and **CB** filters respectively along with the Actual LRU miss ratio. Figures 4.17, 4.19, and 4.21 show the corresponding errors—*absolute* = (estimated-actual), *relative* = (estimated/actual-1). The errors are also tabulated in Tables 4.6 and 4.7. For each analysis, we replicate the estimation hardware (except the Histogram array) to experiment with 2, 4, 8, 16, 32, and 64 Filters.

We find that **CB** filters reduce errors compared to **B** filters for some, but not all, cases. But it incurs additional complexity for applying the (approximate) corrections. **E** filters can have very low errors, e.g., for `apache` (#F=16) and `oltp` (#F=4), but are costly to implement. The **B** filters provide reasonable accuracy at low implementation cost. Surprisingly, increasing the number of filters (for any filter type) does not always increase accuracy for our experiments. We will discuss this issue shortly.

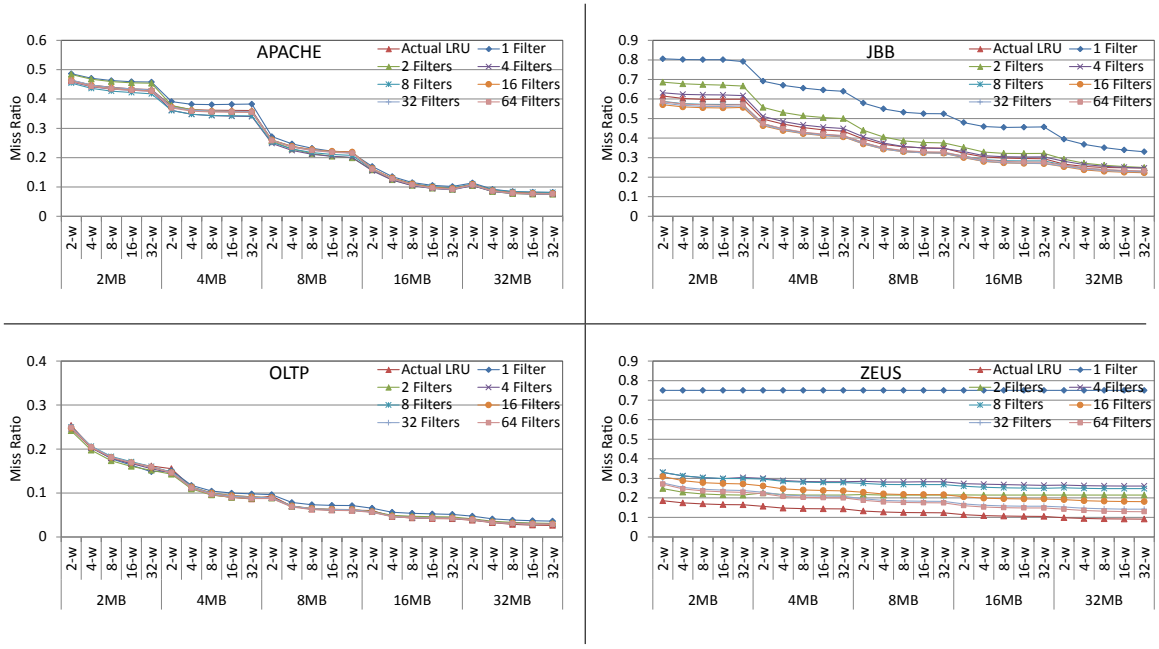


Figure 4.16: Online estimation of miss ratios using **E** filters.

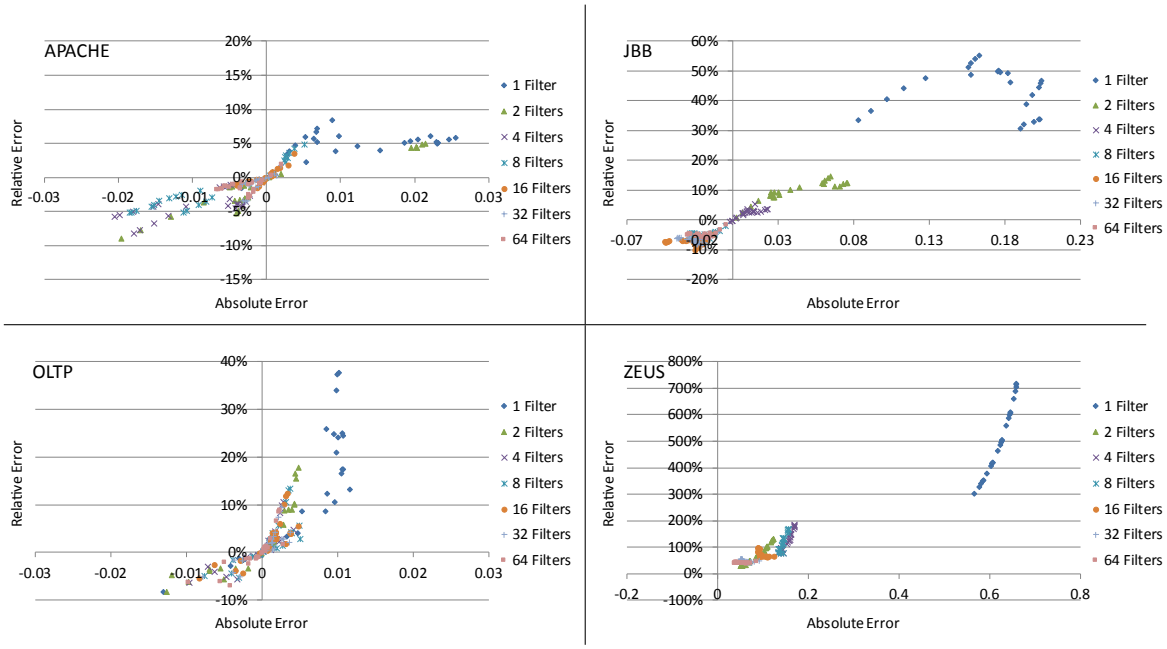


Figure 4.17: Online estimation errors with **E** filters.

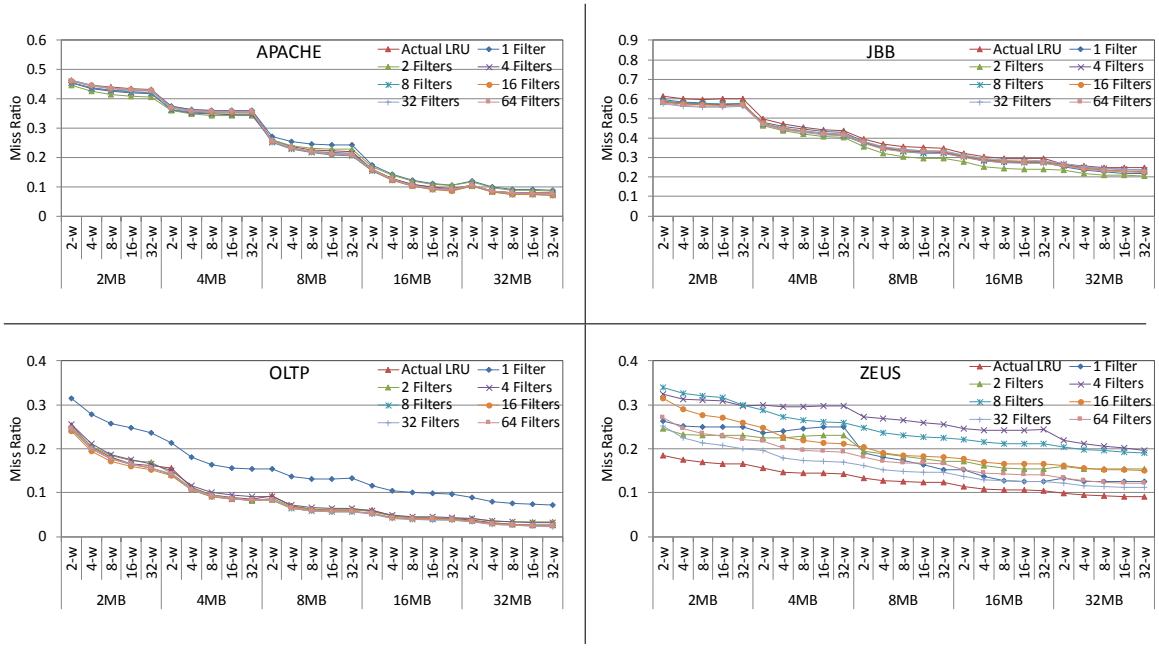


Figure 4.18: Online estimation of miss ratios using **B** filters.

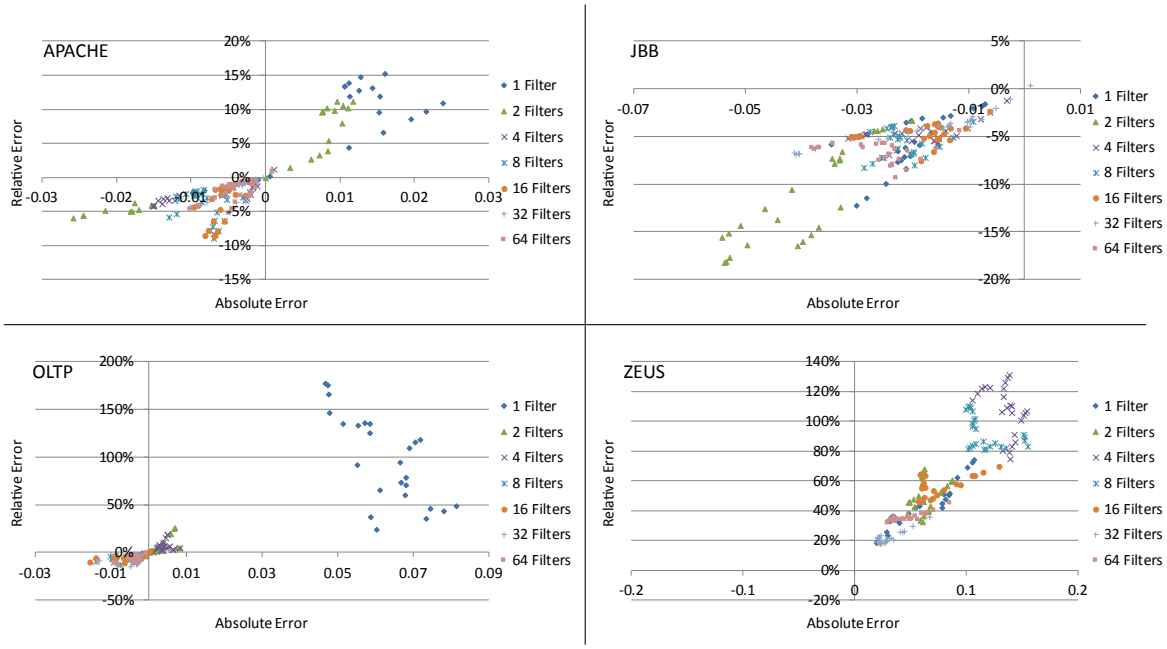


Figure 4.19: Online estimation errors with **B** filters.

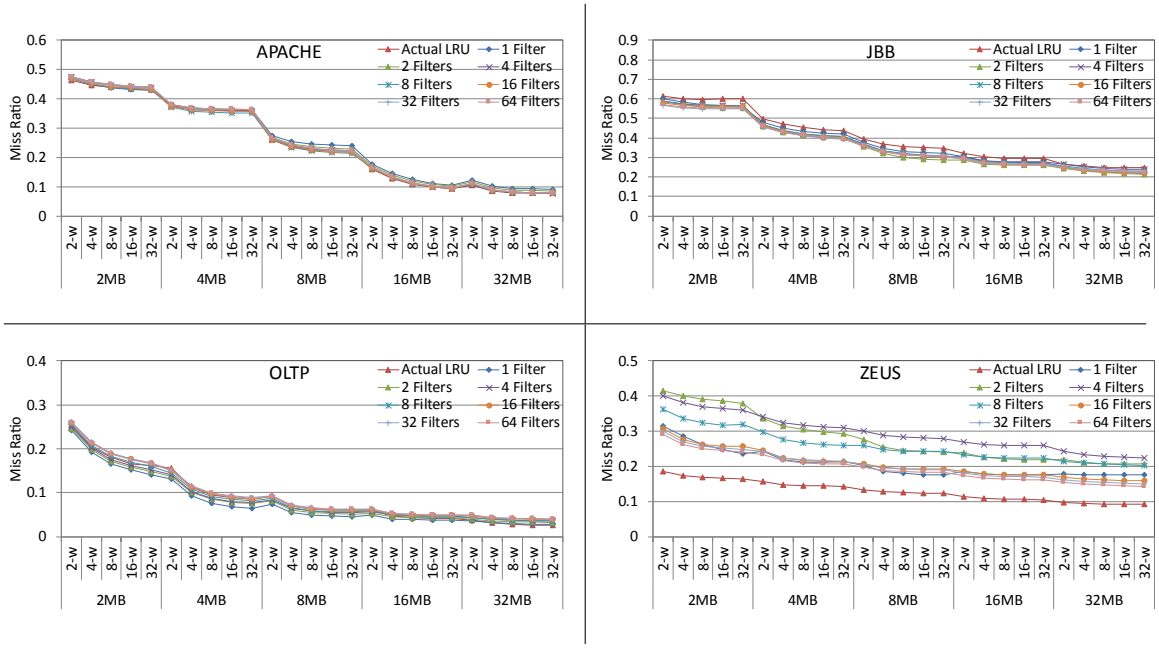


Figure 4.20: Online estimation of miss ratios using **CB** filters.

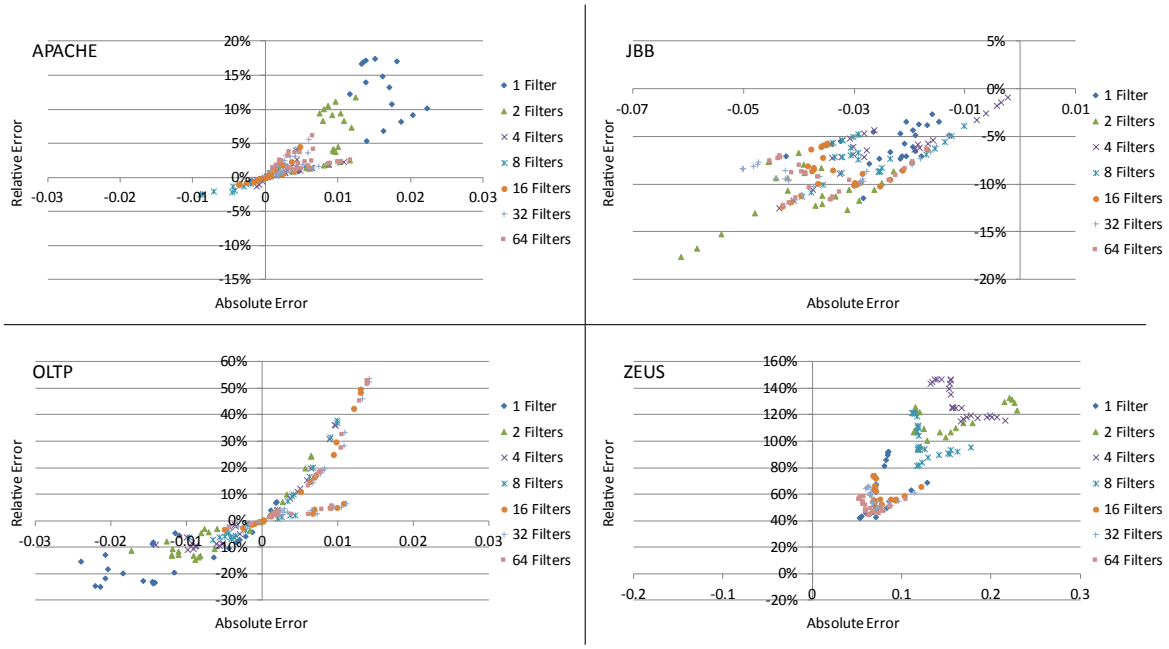


Figure 4.21: Online estimation errors with **CB** filters.

Filter	Workload	#F=1	#F=2	#F=4	#F=8	#F=16	#F=32	#F=64
E	apache	12.21	8.34	8.98	8.74	1.75	1.77	3.29
	jbb	167.64	41.22	10.46	19.57	28.84	25.60	19.68
	oltp	7.89	4.27	2.45	2.47	2.66	2.23	1.85
	zeus	620.31	87.51	152.39	143.10	96.35	61.90	52.98
B	apache	10.77	12.66	7.15	8.46	5.68	3.37	4.28
	jbb	19.87	39.38	18.83	21.51	18.53	18.26	25.84
	oltp	62.43	3.60	3.87	4.38	4.65	4.70	4.49
	zeus	56.45	62.20	134.82	117.29	73.88	29.01	46.41
CB	apache	10.80	8.38	3.14	3.13	2.00	3.88	5.21
	jbb	22.16	38.67	24.12	27.44	32.59	35.08	35.47
	oltp	11.81	7.45	6.59	4.94	5.54	6.61	5.88
	zeus	76.16	148.57	165.43	127.42	76.96	72.78	65.11

Table 4.6: $10^3\times$ Average of absolute error = abs(estimated - actual) miss ratio, over all cache configurations. Entries with values $\geqslant 50$, that is, average error $\geqslant 0.05$ are shaded.

Filter	Workload	#F=1	#F=2	#F=4	#F=8	#F=16	#F=32	#F=64
E	apache	4.94	3.44	3.89	3.42	0.86	1.16	1.44
	jbb	43.74	9.62	2.52	4.96	7.37	6.57	4.97
	oltp	15.33	6.21	3.37	3.78	4.15	3.27	2.60
	zeus	506.07	74.87	125.10	116.71	76.62	48.36	40.91
B	apache	7.25	6.22	2.62	4.44	3.67	1.95	2.16
	jbb	5.64	11.39	4.70	5.71	4.72	4.13	6.69
	oltp	97.88	6.01	6.30	5.53	4.59	7.14	6.49
	zeus	41.54	49.50	107.24	92.18	56.93	21.88	35.29
CB	apache	7.91	5.29	1.28	1.38	1.01	1.86	2.39
	jbb	6.04	10.39	5.95	7.12	8.61	8.83	9.23
	oltp	13.70	10.16	11.42	9.78	11.61	13.72	12.78
	zeus	60.83	113.93	129.96	100.55	60.52	56.96	50.81

Table 4.7: $10^2\times$ Average of relative error = abs(estimated/actual -1) miss ratio, over all cache configurations. Entries with values $\geqslant 10$, that is, average error $\geqslant 10\%$ are shaded.

Workload	apache			jbb			oltp			zeus		
#Filters & Type	**E**	**B**	**CB**	**E**	**B**	**CB**	**E**	**B**	**CB**	**E**	**B**	**CB**
1	332	235	509	48	62	88	236	89	782	4	16	17
2	725	487	1041	132	133	186	554	377	1412	28	26	29
4	1450	1060	2210	277	233	351	1154	812	2608	46	37	50
8	2748	2232	4454	596	471	720	2273	1816	5144	97	80	119
16	5581	4421	8940	1225	926	1468	4584	3544	9681	232	208	287
32	11385	8547	17627	2433	1818	2941	9397	7855	18894	542	557	603
64	22689	17042	35109	4755	3807	5936	19093	15513	38120	1142	987	1243

Table 4.8: Number of samples selected with different sampling configurations. Configurations that selected less than 385 samples are shaded (also see Section 4.6.1.3).

Table 4.8 shows the number of samples selected for reuse distance estimation in each experiment. The large prediction error for `zeus` using 1 **E** filter is due to selecting very few (4) samples for reuse estimation. We show more details for the 4 samples below. In the following, the notation $[n1, n2]$ indicates that the sample started at access number $n1$ (to the LLC) and ended at access number $n2$.

1. $[22, 54]$: Reuse distance of 0 was measured. That is, no intervening access happened that passed through the Set Filter.

2. $[84, 3299679]$: Reuse distance of ∞ was measured. That is, at least 512 intervening access happened that passed through the Set Filter. So, measurement for this sample was terminated.

3. $[3299697, 10292166]$: Reuse distance of ∞ was measured. That is, at least 512 intervening access happened that passed through the Set Filter. So, measurement for this sample was terminated.

4. $[10292169, 10488064]$: End of execution without seeing the reference address again. Reuse distance of ∞ was taken.

Workload	apache			jbb			oltp			zeus		
#Filters & Type	**E**	**B**	**CB**	**E**	**B**	**CB**	**E**	**B**	**CB**	**E**	**B**	**CB**
1	118	89	141	29	34	51	56	37	82	1	8	8
2	171	134	193	69	61	83	81	68	113	10	11	15
4	217	196	248	100	88	109	123	103	144	15	14	20
8	256	239	297	145	132	157	167	151	203	23	24	32
16	311	278	345	213	188	226	238	205	274	46	49	51
32	351	326	402	290	256	300	324	301	359	78	76	82
64	411	375	464	386	350	400	406	390	440	119	116	126

Table 4.9: Number of entries in the Histogram array (Figure 4.14) populated with different sampling configurations. The Histogram array has 512 entries (for reuse distances 0–511).

As can be seen above, measuring long reuse distances "uses up" a significant number of accesses in the trace resulting in a small number of samples for a given trace length. This is due to the fact that the absolute distance, $\mathbf{d}(T)$, increases rapidly with the reuse distance $\mathbf{r}(T)$ (see Section 4.3.3). This effect can be reduced by limiting the range of cache sizes that we want to predict for ($n = 512$ for our study due to the large range of cache sizes that we consider; see Section 4.4.2).

Table 4.9 shows how much of the Histogram array is touched by the different filter configurations. For the detailed example that we just discussed (`zeus` with 1 **E** filter), only 1 entry (for reuse distance 0) was touched. For every workload and filter type, the number of entries touched increases with the number of filters. There is thus a strictly monotonic reduction in the sparsity of the estimated reuse distribution. However, none of the filter configurations touch all 512 entries for our traces. Moreover, the reuse distances corresponding to the touched entries are not all contiguous. It is difficult to reason about accuracy of miss ratio predictions based on sparsely populated reuse distributions. Short traces tend to exacerbate the non-monotonicity in prediction error rates with the number of filters. We expect longer address traces to resolve this issue.

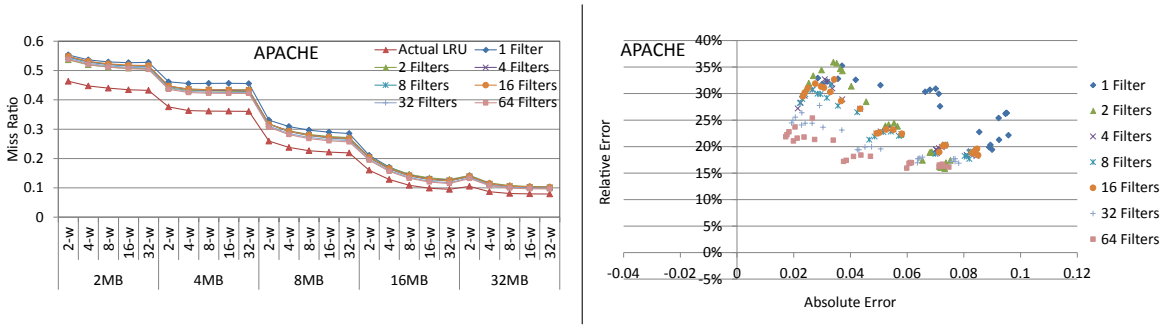


Figure 4.22: Online estimation of miss ratios using **E** filters and with a set sample randomly chosen at the start of every sample.

4.6.1.2 Time Sampling vs Set Sampling

In the Sample Initiation mode, the Control Logic time-samples the incoming address stream, then chooses the set address of the selected address for initializing the Set Filter. An alternative approach is to first choose a set address (select a sample over all possible set addresses) and initialize the Set Filter with that set address.

Figure 4.22 shows prediction errors for `apache` with this approach. The errors are larger than those when the Set Filter is initialized using time sampling (Figure 4.17). One reason is that LLC accesses are not equally distributed to all sets—some sets are accessed more often than others. Time sampling reflects these variations better than set sampling. More samples get chosen with time sampling than with set sampling, since with the latter, the Control Logic has to wait till an address accepted by the Set Filter arrives before using it as the reference address for the reuse measurement.

4.6.1.3 CLT criteria

The Central Limit Theorem (CLT) states that for large sample sizes, the distribution of the sample means approaches a Normal distribution. We will use this to develop a guideline for the minimum sample size that should be selected before the miss ratios are computed.

We will use the following notation:

- μ: Population mean.
- σ: Population standard deviation.
- n: Sample size.
- $\overline{X}$: Sample mean for a given sample.
- α: 1 - confidence level. For a confidence level of 95%, $\alpha = 1 - 0.05$.
- $\mathcal{Z}_{\alpha/2}$: The value ν such that the area under the standard Normal curve between 0–ν is $\alpha/2$.

Then, if the CLT theorem holds, it is well-known [33] that $\overline{X} - \mathcal{Z}_{\alpha/2}\left(\frac{\sigma}{\sqrt{n}}\right) < \mu < \overline{X} + \mathcal{Z}_{\alpha/2}\left(\frac{\sigma}{\sqrt{n}}\right)$. So,

$$\left|\overline{X} - \mu\right| < \mathcal{Z}_{\alpha/2}\left(\frac{\sigma}{\sqrt{n}}\right) \tag{4.18}$$

For a 95% confidence level, $\mathcal{Z}_{\alpha/2} = 1.96$ [33].

The metric that we are interested in this study is the average miss ratio, μ, which is the probability that an access to the cache will miss. Let M_t denote an indicator random variable such that

$$M_t = \begin{cases} 1 & \text{if the } t^{th} \text{access is a miss} \\ 0 & \text{otherwise} \end{cases} \tag{4.19}$$

We compute its expectation, $E(M_t)$, and variance, $Var(M_t)$, as follows:

$$\begin{aligned} E(M_t) &= 1 \cdot P(\text{access is a miss}) + 0 \cdot P(\text{access is a hit}) \\ &= \mu \end{aligned} \tag{4.20}$$

$$\begin{aligned} Var(M_t) &= E(M_t^2) - (E(M_t))^2 \\ &= \mu - \mu^2 \\ &= \mu(1-\mu) \end{aligned} \tag{4.21}$$

Let N denote the total number of accesses (population size). Then, the random variable $M = \frac{\sum_{t=1}^{N} M_t}{N}$ tracks the average number of misses (μ) over all accesses. Assuming that all M_t's are identically and independently distributed, we get

$$\begin{aligned} \sigma &= \sqrt{(Var(M))} \\ &= \sqrt{\left(Var\left(\frac{\sum_{t=1}^{N} M_t}{N}\right)\right)} \\ &= \sqrt{\left(\frac{\sum_{t=1}^{N} Var(M_t)}{N}\right)} \\ &= \sqrt{Var(M_1)} \\ &= \sqrt{\mu(1-\mu)} \end{aligned} \tag{4.22}$$

The expression $\mu(1-\mu)$ is maximized when $\mu = 1-\mu \implies \mu = 0.5$. Therefore, $\sigma \leqslant \sqrt{0.5 * 0.5} = 0.5$. Combining this with Equation 4.18, we can get $\left|\overline{X} - \mu\right| < 0.05$ for a

confidence level of 95%, by satisfying

$$\begin{aligned} |\overline{X} - \mu| &< 1.96 \left(\frac{\sigma}{\sqrt{n}}\right) \leqslant 1.96 \left(\frac{0.5}{\sqrt{n}}\right) < 0.05 \\ \text{that is, } 1.96 &\left(\frac{0.5}{\sqrt{n}}\right) < 0.05 \\ \implies n &\geqslant 385 \end{aligned} \tag{4.23}$$

So, for a sample size of at least 385, the absolute error of the average miss ratio of the sample from the average miss ratio of the entire trace should be less than 0.05. We call this restriction on the sample size as the CLT criteria.

Table 4.8 shows shaded entries for experiments that did not meet this criteria. Figures 4.23, 4.24, and 4.25 show estimation accuracy by the **E**, **B**, and **CB** filters only for configurations that satisfy the CLT criteria. Experiments for workloads other than `zeus` passed this criteria in the sense that the absolute errors in miss ratio were less than 0.05.

Our CLT criteria can fail to contain the maximum error within a specified bound because some assumptions may not hold in practice. For example, access are not necessarily independent, so iid assumptions may not hold. Moreover, we are sampling reuse distances whereas the derivation assumes sampling addresses to check whether they missed or not. These are not orthogonal aspects, since for LRU and the same number of sets, there is a one-to-one mapping between the reuse distance and whether or not the address missed in the cache. However, set locality predictions for a different number of sets can introduce errors.

We propose doing an additional test once the CLT criteria has been satisfied. This is to predict the miss ratio for the *current* cache configuration and compare with the actual value. If the difference is more than a threshold, additional sampling is needed before a cache configuration decision based on estimated miss ratios can be made.

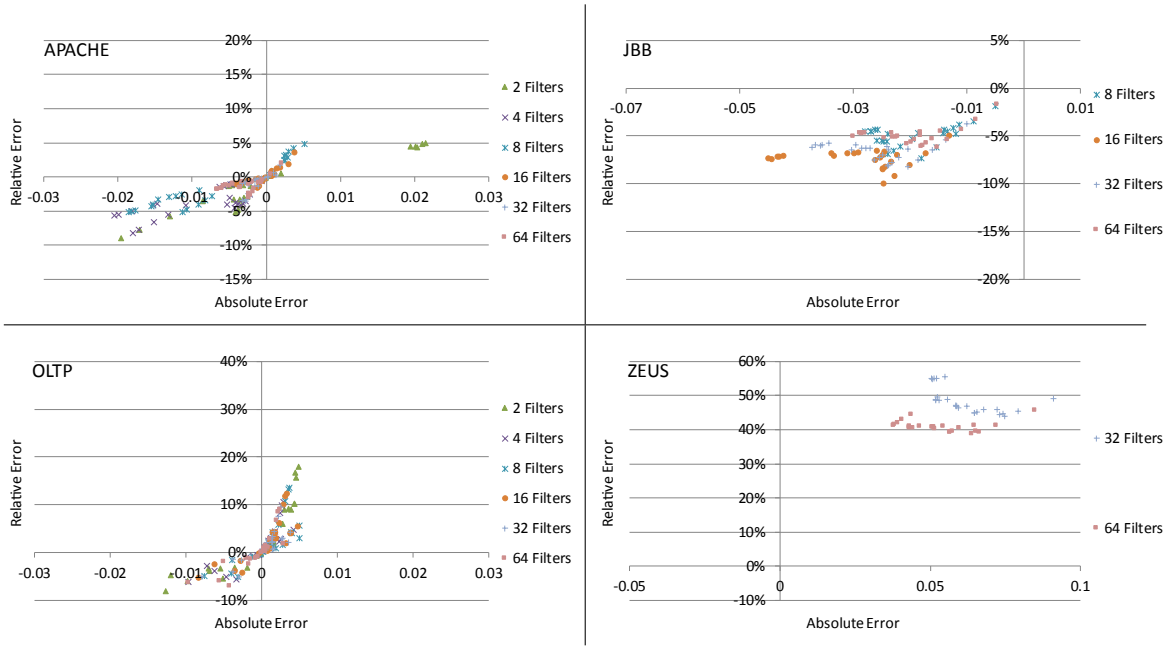


Figure 4.23: Online estimation errors using **E** filters and CLT criteria.

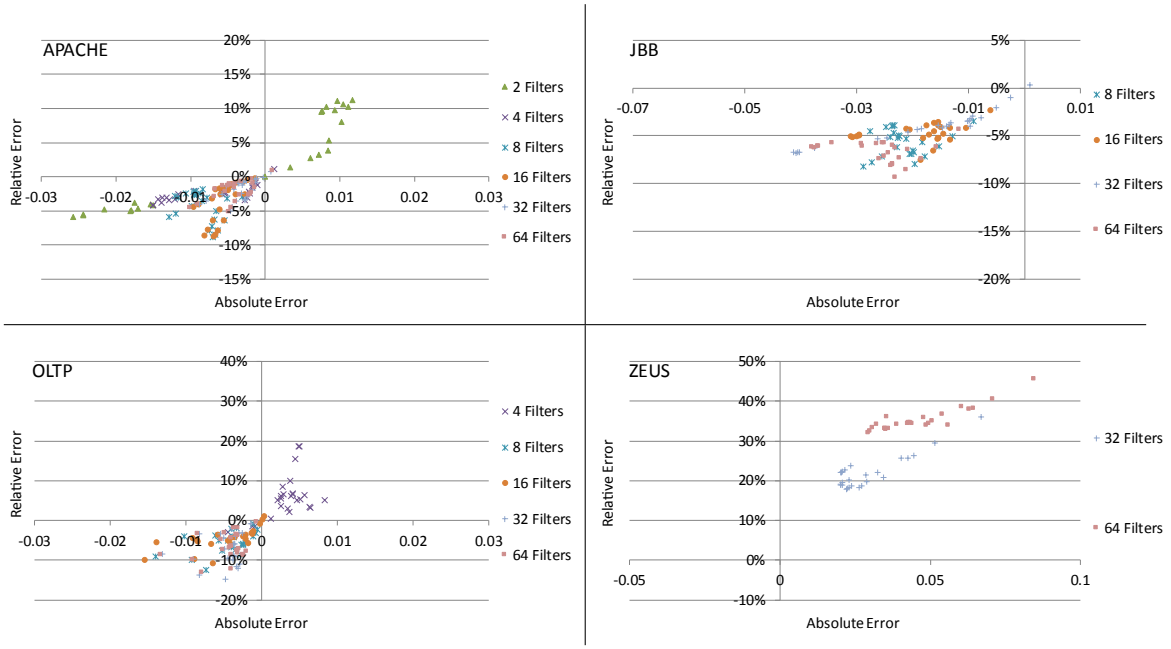


Figure 4.24: Online estimation errors using **B** filters and CLT criteria.

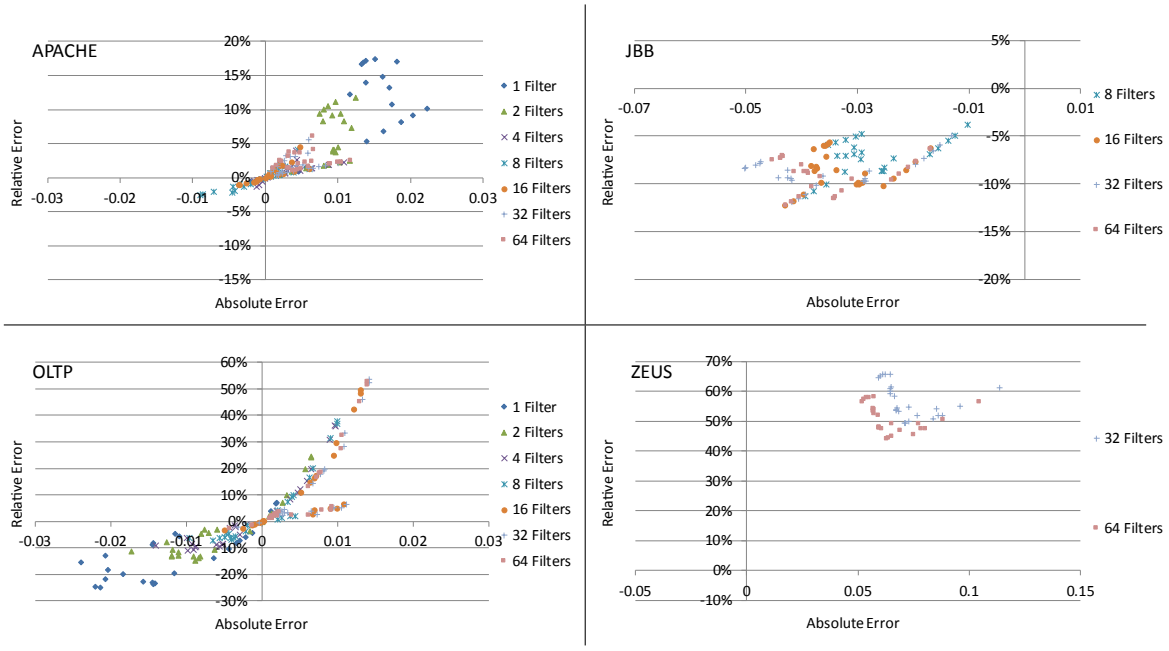


Figure 4.25: Online estimation errors using **CB** filters and CLT criteria.

4.6.2 Set-Counters, Way-Counters, and Shadow Tags

Set-counters [215] use counters that track the number of accesses per set or a group of sets. However, since they can only track changes in the number of accesses per set *but not changes in per-set locality*, they do not model the behavior shown in Figure 4.4.

Way-counters [215] increment a counter associated with each logical stack position (ordered by access recency) on every cache hit. The number of hits for associativity A' is the sum of the counter values from 0 to $A' - 1$.

The above assumes $A' \leqslant A$ where A is the associativity of the current/predicting cache (32MB 32-way in this study). In applications such as dynamic reconfiguration situations, this is problematic since the cache may need to be sized up, not only sized down. Shadow tags [172] (or auxiliary tag directories [175]) circumvent this difficulty by maintaining a copy of the tags that is not deactivated during reconfigurations. This

always maintains a stack depth to the maximum desired value and facilitates simulating the effect of hits and misses on a cache with associativity larger than that of the current cache. Qureshi et al. [174] used dynamic set sampling to reduce storage and power costs of the shadow copy.

Way-counter values, converted to probabilities, estimate $\mathbf{r}(S)$ up to length A. The estimation is exact for LRU caches. Their operation can be understood by deriving Equation 4.7 from Equation 4.3 instead of from Equation 4.2. We get

$$h(S') = \sum_{i=0}^{A'-1} \mathbf{r}_i(S) + \sum_{i=A'}^{n} \mathbf{r}_i(S) \cdot \sum_{k=0}^{A'-1} {}^{i}C_k \cdot \left(\frac{S}{S'}\right)^{k} \cdot \left(1 - \frac{S}{S'}\right)^{(i-k)}$$

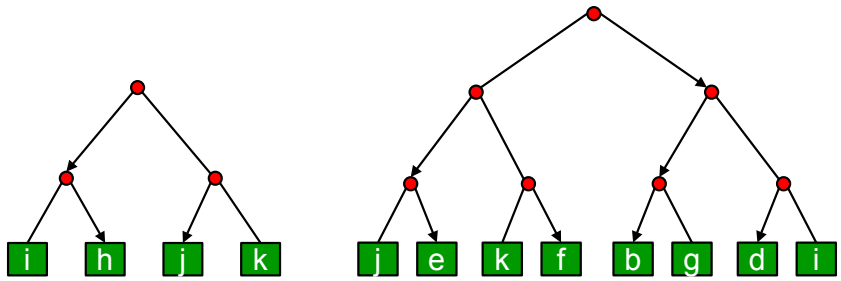
Under the assumption $S' = S$, $h(S') = \sum_{i=0}^{A'-1} \mathbf{r}_i(S)$ which is computationally extremely efficient.

4.6.2.1 Way-counters for PLRU

In PLRU, the MRU line is known with certainty but the rest of the logical ordering is not precisely known. Kedzierski et al. [127] proposed a heuristic for approximating logical stack positions for PLRU caches to enable way-counter based prediction . Let $waynum$ be the way number of the accessed line and $pathbits$ denote the bit-values of the tree nodes along the path from the root to the leaf with root bit in MSB position. Let the function $reverse(b)$ reverse bit positions in the binary representation of b. The following heuristic is used to approximate $URD(x, m)$:

$\hat{URD}(x, m) = A - 1 - (reverse(waynum) \oplus pathbits)$

This approach aims to compute $\mathbf{r} \cdot \boldsymbol{\phi}(LRU)$ with $\mathbf{r}$ approximately measured using the above mechanism. However, apart from the traditional limitations of way-counters (Section 4.6.2.2), it also ignores the fact that $\boldsymbol{\phi}(PLRU) \neq \boldsymbol{\phi}(LRU)$ for $A \neq 2$. Interestingly, it fails



(a) **4-way** (b) **8-way**

Figure 4.26: PLRU trees demonstrating non-inclusion. The 8-way tree does not include element h of the 4-way tree.

to accurately estimate the hit ratio even for a 2-way cache where $\phi(\mathrm{PLRU}) = \phi(\mathrm{LRU})$ *when the current configuration that does the estimation has* $A \neq 2$ (see, for example, Figure 4.13 where the current/predicting configuration has A=32). In contrast, our framework overcomes this by *decoupling hit ratio estimation from the organization of the current cache*.

4.6.2.2 Way-Counter Limitations

Way-counters (+shadow tags) have the following fundamental limitations:

Fixed number of sets: The relation ($S' = S$) that makes way-counters efficient also implies the restriction that the number of sets must be fixed. As can be observed from Table 4.1, miss ratios for only 4 of 24 configurations can be predicted at any time; other predictions must be preceded by (time-consuming) re-training for the changed S'.

However, our framework reveals that Equation 4.6.2 may be used to transform way-counter values when $S' \geqslant S$ (also see discussion for Case 1 in Section 4.4). With reference to Table 4.1, maintaining shadow tags corresponding to $S = 2^{10}$ allows conversion of values for any $S' \neq S$. But, Figure 4.6 shows that to use way-counter values for a cache with a larger number of sets, the shadow tags and counter values must be maintained for $n(> A)$ positions.

Replacement policies with stack inclusion: Way-counters exploit the stack inclusion property [150] of LRU to predict miss ratios $\forall A' \leqslant A$. For replacement policies that do not guarantee stack inclusion (PLRU/RANDOM/NMRU), this is no longer true.

For example, consider the access sequence: `a b c d e d f e g h f i j i k` simultaneously to an 8-way PLRU set and a 4-way PLRU set. Figure 4.26 shows the two sets and associated PLRU trees after the sequence. The arrows in the figures point to the less recently used subtree. Initially, both sets were empty and the eviction bits in each tree were pointing to the "left" subtrees. At the end of the sequence, the two sets together contain 9 distinct elements (`i h j k e f b g d`) whereas a policy satisfying inclusion would have exactly 8 elements. Thus, maintaining information for 8 ways is not sufficient to accurately predict miss ratios for both a 4-way and an 8-way cache *even if* $S' = S$.

Tight coupling with replacement policy implementation: Since way-counters are tightly coupled with the implementation of replacement policies that track stack positions (e.g. LRU), they are unusable with other policies such as RANDOM that can also be predicted well using reuse information. Way-counters depend on the replacement policy mimicking stack operation, so they run into trouble when the stack is absent (PLRU/RANDOM/NMRU) (see Section 4.6.2.1 for a discussion on PLRU) or reconfigured ($S' \neq S$).

Shadow Tag overhead: For very large caches, tag area and power are significant. Loh and Hill [142] propose novel tag management schemes for such caches. Maintaining additional shadow tags in those systems seem difficult.

4.7 Index Hashing

All the experiments in this chapter use an XOR-based hashed indexing scheme for the LLC. The hashing scheme is inspired by prior work [55, 56].

Let $x = \log_2(S)$. Given a 32-bit byte address b, a "plain" cache interprets the bits of b as follows:

- [0 : 5] :- block address. (Each cache line is 64 bytes.)
- [6 : 6+(x-3)-1] :- set address
- [6+(x-3) : 6+(x-1)] :- bank address. (Our LLC has 8 banks.)
- [6+x : 31] :- tag

Our "hashed" cache interprets the bits of b as follows:

- [0 : 5] :- block address. (Each cache line is 64 bytes.)
- Let k1 = bits 6 : (6+(x-3)-1) from b. (This is the set address for the "plain cache".) Let k2 = bits 20 : 31 from b. Compute $k3 = k1 \oplus k2$. Bits 0 : ((x-3)-1) from k3 form the set address for the "hashed cache".
- [6+(x-3) : 6+(x-1)] :- bank address. (Our LLC has 8 banks.)
- [6+x : 31] :- tag

Figure 4.27 shows the savings in miss ratio with hashed indexing compared to plain indexing, computed as 1 - (miss ratio with hashed indexing/miss ratio with plain indexing). While there is no guarantee that hashing will always reduce miss ratios, it reduces it in most cases. Depending on the bit patterns in the addresses, the savings can be non-trivial, e.g., up to ~27% (absolute difference in miss ratio of 0.036) for `apache`.

Hashed indexing aims to distribute the total number of *unique* addresses uniformly over all the cache sets. This is corroborated by Table 4.10 that shows the coefficient of variation of the number of unique addresses mapped to each cache set for a 32-way cache of the given size. The coefficient of variation is significantly lower with hashed

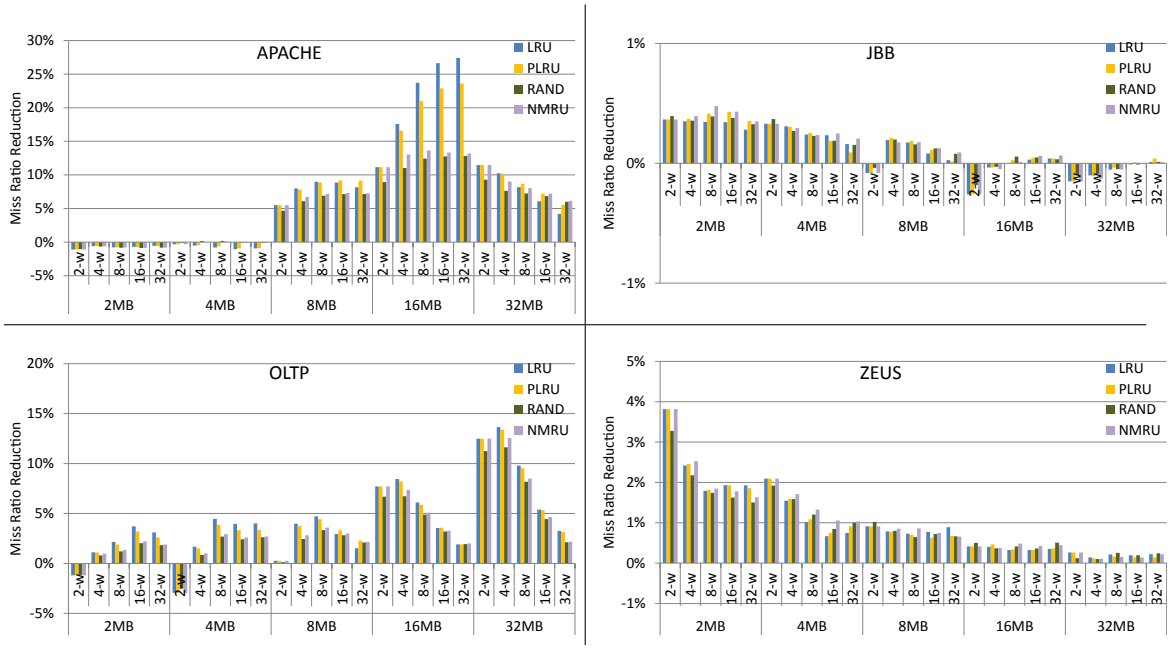


Figure 4.27: Miss ratio reduction with hashed indexing.

indexing compared to plain indexing. The Table also shows the minimum and maximum number of unique addresses mapping per set over all sets. As expected, the min–max range is higher with plain indexing than hashed indexing although the average is the same for both schemes. However, its impact on miss ratio reduction is less due to two reasons—(i) the associativity of 32 is much less than the average number of unique addresses mapping to each set, so conflict misses cannot be eliminated, and (ii) some addresses will compulsorily miss irrespective of how they map to cache sets.

Since the mapping of unique addresses to cache sets is more uniform with hashed indexing than with plain indexing, the former is also more suited for applying the models that we developed in this chapter. (Section 4.4.1 discusses the uniform mapping assumption made by our models.)

Workload	Size (MB)	Avg.	Plain			Hashed		
			Min.	Max.	Coeff.	Min.	Max.	Coeff.
apache	2	3739.28	2513	4900	0.115	3444	4033	0.028
	4	1869.64	1238	2481	0.116	1705	2027	0.025
	8	934.82	603	1283	0.117	839	1038	0.033
	16	467.41	284	658	0.120	406	530	0.042
	32	233.71	132	337	0.125	186	278	0.054
jbb	2	5694.68	5526	6259	0.013	5603	5798	0.006
	4	2847.34	2741	3143	0.015	2769	2936	0.008
	8	1423.67	1358	1578	0.017	1364	1485	0.012
	16	711.83	660	798	0.022	668	759	0.018
	32	355.92	320	400	0.029	324	392	0.025
oltp	2	2345.03	1375	8037	0.584	2213	2513	0.022
	4	1172.52	664	4094	0.585	1066	1303	0.030
	8	586.26	311	2048	0.586	504	658	0.041
	16	293.13	145	1085	0.589	246	348	0.053
	32	146.56	67	622	0.593	103	194	0.075
zeus	2	979.65	589	1311	0.107	901	1083	0.031
	4	489.82	289	681	0.110	422	559	0.043
	8	244.91	138	365	0.117	196	299	0.060
	16	122.46	66	192	0.128	89	155	0.080
	32	61.23	29	100	0.147	37	90	0.117

Table 4.10: #Unique. line addresses mapping to each set of a 32-way cache with plain and hashed indexing. Coeff. (Coefficient of Variation) = Standard Deviation/Average.

4.8 Limitations

Our models currently do not handle the following:

1. **Short-term Effects**: Our models predicts long-term averages for cache performance. Reconfiguration policies based on these models will not be able to react to high-frequency (short-term) variations in cache performance. The main reason for this is that the reuse sampling framework needs long traces, e.g., over several seconds of execution time, to get a reasonable number of samples that can be used for miss ratio

predictions. High-frequency reconfigurations for large caches may anyways not be feasible due to (i) latency and energy overheads in writing back a potentially large amount of dirty data to memory when downsizing the cache and (ii) significant cache warmup delays when upsizing the cache.

2. **Prefetching Effects**: Our models do not consider variability in the address stream that may be caused by prefetching. Currently, we assume that all demand and prefetch accesses to the LLC are included in the address stream presented to the reuse estimation hardware and that the characteristics of this address stream will remain unchanged with a reconfigured cache.

3. **Cache Hierarchy Effects**: Our models also do not consider variability in the address stream due to inclusion policies in the cache hierarchy. For example, in a strictly inclusive hierarchy, evictions at the LLC may cause evictions at L1 and/or L2. This in turn can cause additional misses at those cache levels, resulting in a different address stream arriving at the reconfigured LLC. This effect would be larger at smaller LLC sizes than at larger sizes.

4. **Other replacement policies**: We have not modeled other proposed replacement policies [117, 119] that improve upon LRU. One way to handle those could be to model their relative advantage over LRU and use that in conjunction with the LRU model described in this work to predict cache performance.

4.9 Conclusion

The central theme of this chapter is an online modeling framework, new analytical models, and efficient hardware support, to predict cache performance at runtime for a range of replacement policies and cache organizations. Our framework uses the concept of

reuse/stack distances and transformations of probability vectors with Binomial matrices. The framework unifies previous analytical models such as Smith's associativity model, Cypher's Poisson model, and hardware techniques such as way-counters. We discussed limitations of set and way-counters, gave a method to convert way-counter values for caches with a different number of sets and showed that this requires maintaining shadow tags for more than the maximum associativity. We also proposed a new predictor that is decoupled from the cache configuration, uses hardware signatures for compact representation of reuse intervals and can be used as an alternative to way-counters for miss ratio predictions.

These models will enable governors to also decide optimal cache configurations, without needing to profile numerous potential target cache configurations, in addition to configurations for other knobs. Chapter 5 demonstrates one such governor that uses these models and meets SLApower by simultaneously reconfiguring core frequency and size of the last-level cache.

5 CACHE POWER BUDGETING

> *The leakage power of a modern last-level cache is larger than the power of a simple core running full out.*
>
> — Mark Horowitz [105]

5.1 Overview

Caches improve performance by reducing the effective memory access latency, but consume significant static and dynamic power. Just as caches cannot be simultaneously large and fast, they also cannot be both large and low power. Smaller caches consume less static power, but can degrade performance and increase dynamic power due to more misses. There is thus a tradeoff between performance and power consumption, which depends on the currently executing set of workloads and data sets. Previous work has shown that workloads often have critical working sets [233], and by reducing the cache size to just hold the working set it is often possible to save power without a significant performance loss [10]. The recent Ivy Bridge microarchitecture powers down a subset of cache ways during periods of low activity [116].

In this chapter we explore power-performance management by dynamically configuring core frequency and resizing the last-level cache. We develop a new governor that targets SLApower, that is, maximizes performance for a power budget (see Figure 3.1 in Chapter 3 for an illustration of the actions that this governor must take). The power budget that we consider is the system power consumed by the baseline configuration that uses a large 32MB last-level cache and runs the cores at 2.132 GHz (see Section 5.2 for details about configurations) to execute a given workload.

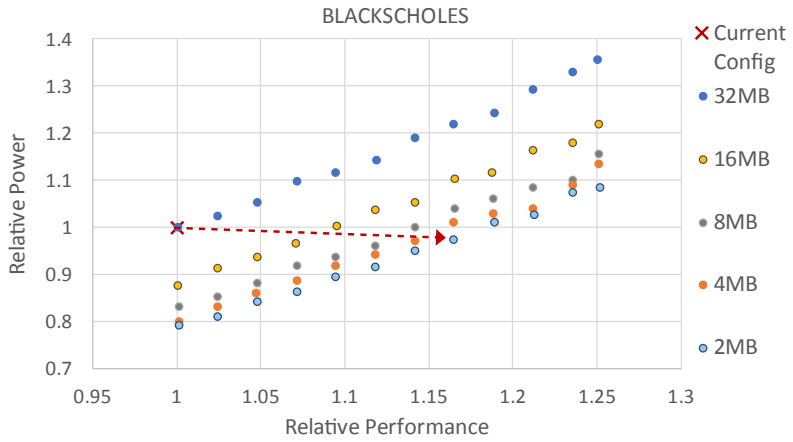


Figure 5.1: Power-performance for blackscholes with DVFS and cache resizing.

Figure 5.1 shows the power-performance state space for one of our workloads, blackscholes, with cache sizes ranging from 2MB to 32MB and core frequencies ranging from 2.132 GHz to 2.665 GHz. The point (1,1), marked with ×, corresponds to the baseline (current) configuration. For a fixed cache size, both performance and power increase with frequency. For this workload, when frequency is fixed and cache size is increased, performance barely changes but power consumption increases. The dashed arrow shows the action needed to be taken by the governor to maximize performance while not exceeding the power of the baseline (current) configuration. For this workload, the cache needs to be reconfigured to 2MB and the frequency needs to be set to 2.482 GHz to get a performance improvement of 16.5% while staying within the power budget. The arrow is slightly dipped instead of being perfectly horizontal due to the unavailability of a valid configuration at that point. Thus, for this workload, the entire power budget cannot be utilized and the desired configuration will save (under-utilize) 2.6% power in addition to improving performance by 16.5%.

Improved performance at the same power translates into energy savings by reducing

RUE and subsequently, CPUE. Depending on the system, cache reconfigurability may also reduce E_{min}. For example, we see in Figure 5.1 that having a 2MB configuration lowers the Pareto frontier compared to the 32MB configuration. This in turn can change EOP (decrease slope of line) and reduce E_{min}. Cache to core power-shifting benefits will be more pronounced on processors running at lower clock frequencies than at higher frequencies. This is because the cache power available to be redistributed is a larger fraction of the system power in such environments whereas core dynamic power dominates system power at higher frequencies.

To intelligently budget power between cores and caches, we investigate using hardware support to drive analytical models of system power and performance. Online estimation enables real-time feedback and adaptation to dynamic changes such as operating system interactions or changing workload mixes. We demonstrate an integrated framework that combines a cache reuse model, performance model, power model and DVFS model to identify optimal power-budgeted configurations.

We extend the state of the art in two ways:

1. We show that careful cache power budgeting—using DVFS and cache reconfiguration—driven by *low-overhead* predictors can improve performance, not just save power. Our new governor exploits this to make the system operate close to Dynamic EO and satisfy SLApower. For our target system, budgeting for system power improves performance by 0.5%–15.2% for 15 of 32 workloads and saves total energy (that includes wall power, not just on-chip and memory power) by 4.4% averaged over all 32 workloads.

2. We develop the first online analytic power and performance models for reconfigurable caches that work for practical replacement policies (i.e., PLRU not just true

LRU), do not use shadow tags, can predict for configurable-set caches, and can configure up to larger caches, not just down to smaller ones.

Earlier works, e.g., Meng et al. [155], have used way counters and shadow tags to determine appropriate cache configurations. However, way counters model fixed-set configurable-associative caches. We discuss in Section 5.5 that such caches can cause large performance degradation with small LLCs in an inclusive hierarchy that can be avoided by using an LLC with large associativity for the same cache size. Thus, we use configurable-set fixed-associative caches in this study. Our reuse distance based cache performance predictor, described in Chapter 4, can be used to model such caches.

The rest of this chapter is organized as follows. Section 5.2 describes the system model and the workloads that we use for our evaluation. Section 5.3 shows some of the opportunities for saving power and energy by reconfiguring caches and the need for intelligent power budgeting. Section 5.4 presents a big-picture view of our power-budgeting approach. Section 5.5 justifies our choice of using configurable-set fixed-associative caches and discusses errors in cache miss rate predictions. Section 5.6 presents the models that we use to predict performance and power. Section 5.7 combines these with DVFS models to predict optimal system configurations.

5.2 Infrastructure

Table 5.1 describes the 8-core CMP we use in this study. We assume that the core frequencies can be increased from the baseline frequency of 2132 MHz in steps of 50 MHz. Section 5.7 describes the scaling assumptions in more detail. Similar to our Haswell server, HS, all cores operate at the same frequency.

Core configuration		4-wide out-of-order, 128-entry window, 32-entry scheduler		
Number of cores		8	**On-chip frequency**	2132–2665 MHz
Technology Generation		32nm	**Temperature**	340K–348K
Functional Units		4 integer, 2 floating-point, 2 mem units		
Branch Prediction		YAGS 4K PHT 2K Exception Table, 2KB BTB, 16-entry RAS		
Disambiguation		NoSQ 1024-entry predictor, 1024-entry double-buffered SSBF		
Fetch		32-entry buffer, Min. 7 cycles fetch-dispatch time		
Inclusive	**L1I Cache**	private 32KB 4-way per core, 2 cycle hit latency, ITRS-HP		
	L1D Cache	private 32KB 4-way per core, 2 cycle hit latency, ITRS-HP		
	L2 Cache	private 256KB 8-way per core, 6 cycle access latency, PLRU, ITRS-LOP		
	L3 Cache	shared, configurable 2–32 MB 32 way, 8 banks, 18 cycle access latency, PLRU, ITRS-LOP, serial		
Coherence protocol		MESI (`Modified`, `Exclusive`, `Shared`, `Invalid`), directory		
On-Chip Interconnect		2D Mesh, 16B bidirectional links		
Main Memory		4GB DDR3-1066, 75ns zero-load off-chip latency, 2 memory controllers, closed page, pre-stdby		

Table 5.1: System configuration.

We assume an 8-banked L3 cache that is dynamically re-configurable, with capacities ranging from 2MB to 32MB and 32-way associativity, for a total of 5 cache configurations. We conservatively assume that the access latency in cycles is constant for all configurations. To evaluate power and performance, we perform full-system simulation using GEMS [149] augmented with a detailed timing and power model. We use CACTI 5.3 [197] to determine the static power and dynamic activation energy per component.

The conventional bit-selection cache index function may not map addresses uniformly across the sets. A number of systems that target commercial workloads use more sophisticated hashing functions [46, 198, 221] to reduce conflict misses. Complex hashed index functions are more common in L2 and L3 caches, where an XOR-based hash function adds a negligible delay to the access latency. We use a simple XOR-based hashing function, discussed in Chapter 4, to distribute lines more uniformly among the L3 cache sets [55, 56]. This usually improves performance over the conventional bit-

Multithreaded		Multiprogrammed	
Workload	**Abbrv.**	**Workload**	**Abbrv.**
blackscholes	blac	astar(4)+bwaves(4)	as-bw
bodytrack	body	astar(4)+gcc(4)	as-gc
fluidanimate	flui	astar_lakes(8)	as-la
freqmine	freq	bwaves(8)	bwaves
swaptions	swap	bzip2(8)	bzip2
ammp	ammp	cactusADM(2)+mcf(2)+milc(2)+bwaves(2)	c-m-m-b
equake	equa	gcc_166(8)	gcc
fma3d	fma3	gcc(1)+omnetpp(1)+mcf(1)+bwaves(1)+ lbm(1)+milc(1)+cactusADM(1)+bzip2(1)	g-o-m-...
gafort	gafo	lbm(8)	lbm
mgrid	mgri	libquantum(8)	libq
swim	swim	libquantum(4)+bzip2(4)	li-bz
wupwise	wupw	mcf(4)+bwaves(4)	mc-bw
apache	apac	mcf(4)+libquantum(4)	mc-li
jbb	jbb	omnetpp(8)	omnetpp
oltp	oltp	omnetpp(4)+lbm(4)	om-lb
zeus	zeus	soplex_pds_50(8)	soplex

Table 5.2: Workloads and their abbreviations. Numbers in parentheses for multiprogrammed workloads indicate the number of copies of the corresponding constituent workload. For example, astar(4)+bwaves(4) means 4 copies of astar and 4 copies of bwaves. Each constituent workload of multiprogrammed workloads is single threaded.

selection index function and also makes cache modeling easier by making the distribution more uniformly random. Due to the self-canceling property of XOR, no extra tag bits need to be stored to retrieve original addresses, if necessary, from the hashed values.

We use two types of workloads—multithreaded and multiprogrammed. Table 5.2 enumerates these workloads along with their abbreviations that we use in the rest of this chapter. Each workload uses a total of 8 threads.

Multithreaded workloads consist of 7 SPEComp [15] benchmarks (ammp, equake, fma3d, gafort, mgrid, swim, wupwise) with "ref" inputs, 5 PARSEC [30] benchmarks (blackscholes, bodytrack, fluidanimate, freqmine, swaptions) with "simlarge" inputs, and 4 Wisconsin

commercial workloads [4] (apache, jbb, oltp, zeus). Each workload runs for a fixed amount of work (e.g. #transactions or loop iterations [6]), corresponding to ~380M – ~570M instructions (average: ~500M) depending on the workload, that is logically partitioned into adjacent training and prediction intervals of roughly equal size. The interval sizes vary according to the available work units in each workload. (See Section 5.4 for a discussion of execution intervals.)

Multiprogrammed workloads consist of combinations of SPEC CPU2006 [101] benchmarks (astar, bwaves, bzip2, cactusADM, gcc, lbm, libquantum, mcf, milc, omnetpp, soplex). Each workload consists of 8 programs, equally divided among the benchmarks in the combination. The training and prediction intervals are ~250M instructions each.

Each simulation run starts from a mid-execution checkpoint that includes cache warmup. Simulating one or more seconds of target execution is infeasible due to high simulation overheads of detailed models.

5.3 Cache Resizing Opportunities

Figure 5.2 shows MPKI (Misses Per Kilo Instruction) with respect to cache size for our workloads in the Prediction Interval. (See Section 5.4 for a discussion of execution intervals.) Our workloads exhibit a range of miss rates from very small (<< 1 MPKI) to quite large (> 40 MPKI). We also observe the following distinctive characteristics among our workloads.

1. **Cache insensitive (low/medium locality)**: those that have a significant part of their working sets that never fit in the cache. They have a high miss rate that changes very little with cache size e.g., equa, fma3, flui, freq, gafo, wupw, libq, lbm.

2. **Cache insensitive (high locality)**: those whose working sets mostly fit in the cache.

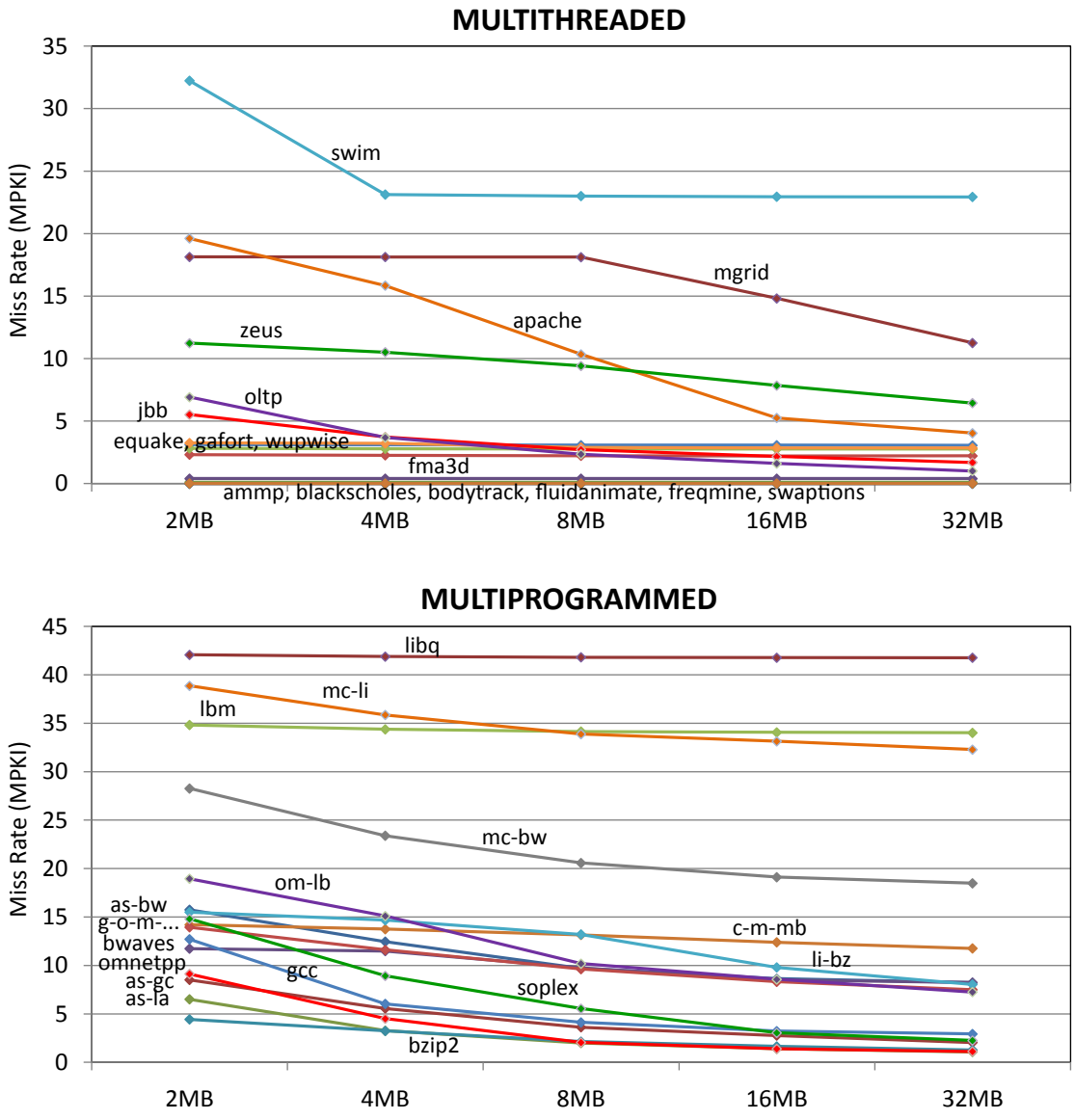


Figure 5.2: MPKI vs cache size (32-way) in the Prediction Interval. (See Section 5.4 for a discussion of execution intervals.)

They have a low miss rate that changes very little with cache size e.g., blac, body, swap, ammp.

3. **Cache sensitive**: those whose working sets fit significantly more in large caches than in smaller ones. Their miss rates change significantly with cache size, e.g., commercial workloads, mgrid, li-bz, mc-bw, mc-li, om-lb, soplex, etc.

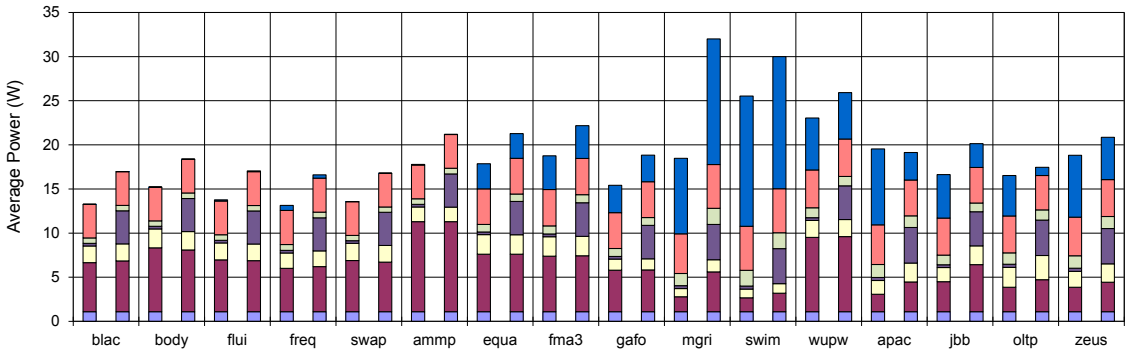
Cache sensitive workloads incur high performance losses with small caches and require larger caches for good performance. So there is less power-saving (and subsequently, power-budgeting) opportunity by using a smaller cache. In contrast, cache insensitive workloads are good candidates for power-budgeting, since their performance largely does not change for different configurations.

Figure 5.3 illustrates the opportunity. It shows the average power breakdowns for our workloads with the smallest LLC configuration (2MB 32-way), the largest LLC configuration (baseline 32MB 32-way), and the performance gain, power saved, and energy saved by executing the workload with the smallest cache instead of with the largest cache. The metrics are inter-related as follows.

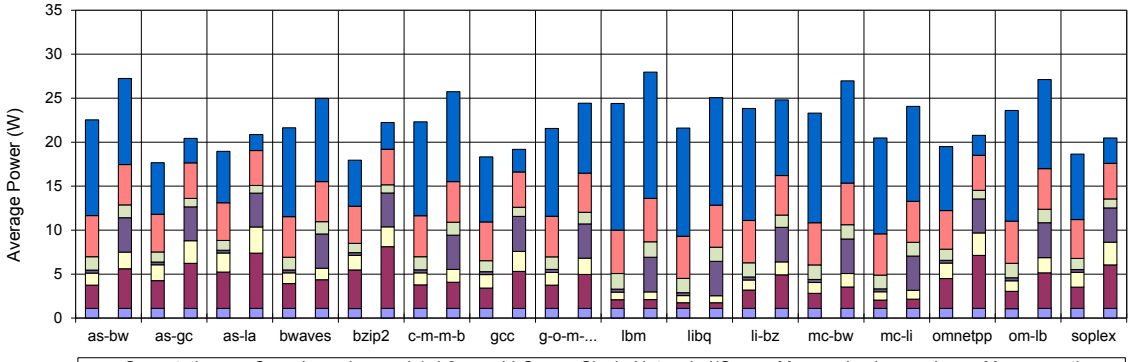
$$\begin{aligned} \text{PerfGain} &= \text{Speedup} - 1 \\ &= \frac{\text{time with 32MB LLC}}{\text{time with 2MB LLC}} - 1 \qquad (5.1) \\ \text{PwrSave} &= 1 - \frac{\text{power with 2MB LLC}}{\text{power with 32MB LLC}} \qquad (5.2) \\ \text{EnrSave} &= 1 - \frac{\text{energy with 2MB LLC}}{\text{energy with 32MB LLC}} \qquad (5.3) \end{aligned}$$

Since Energy = Time $\times$ Power, we have

$$(1 - \text{EnrSave}) = \frac{1 - \text{PwrSave}}{1 + \text{PerfGain}} \qquad (5.4)$$



Workload	blac	body	flui	freq	swap	ammp	equa	fma3	gafo	mgri	swim	wupw	apac	jbb	oltp	zeus
PerfGain	0.1%	-0.2%	-0.5%	-3.4%	-0.1%	0.6%	0.3%	-0.8%	-0.7%	-57.5%	-21.1%	-1.1%	-39.7%	-35.2%	-28.0%	-23.3%
PwrSave	21.8%	17.2%	19.3%	20.8%	19.2%	16.1%	16.0%	15.4%	18.1%	42.3%	14.9%	11.1%	-2.1%	17.5%	5.3%	9.8%
EnrSave	21.8%	17.0%	18.8%	18.1%	19.1%	16.6%	16.3%	14.7%	17.5%	-35.9%	-7.9%	10.1%	-69.4%	-27.4%	-31.4%	-17.6%
EnrSaveW	9.5%	7.6%	7.9%	5.7%	8.1%	8.7%	8.4%	7.2%	7.7%	-73.0%	-15.0%	5.4%	-67.6%	-41.2%	-35.5%	-24.0%



Workload	as-bw	as-gc	as-la	bwaves	bzip2	c-m-m-b	gcc	g-o-m-...	lbm	libq	li-bz	mc-bw	mc-li	omnetpp	om-lb	soplex
PerfGain	-38.5%	-36.8%	-33.1%	-12.6%	-36.1%	-9.4%	-42.0%	-30.6%	-1.3%	0.2%	-40.6%	-26.2%	-6.3%	-42.2%	-49.7%	-49.3%
PwrSave	17.3%	13.5%	9.2%	13.3%	19.2%	13.3%	4.5%	11.8%	12.7%	13.8%	3.9%	13.6%	14.9%	6.1%	13.0%	9.0%
EnrSave	-34.5%	-36.9%	-35.8%	0.9%	-26.4%	4.3%	-64.8%	-27.1%	11.6%	14.0%	-61.9%	-17.1%	9.2%	-62.3%	-72.8%	-79.6%
EnrSaveW	-46.0%	-47.7%	-42.6%	-5.9%	-40.9%	-2.0%	-68.9%	-34.7%	6.5%	7.9%	-64.8%	-24.7%	2.0%	-67.6%	-83.5%	-88.5%

Figure 5.3: Two power stacks are shown for each workload: the left stack for the lowest-power LLC (2MB 2-way) and the right stack for the highest-power LLC (baseline configuration, 32MB 32-way). Power consumed by the 2MB LLC is not conspicuous as it constitutes a small percentage of total power. **PerfGain** shows performance gain (= speedup - 1), **PwrSave** shows system power saved, and **EnrSave** shows system energy saved with the 2MB 32-way cache with respect to the baseline 32MB 32-way cache. **EnrSaveW** accounts for wall power in energy savings calculations. Negative improvements (= losses) are highlighted. Core power includes ROBs, bypass networks, functional units, register files, TLBs, branch predictors, fetch & decode logic. We assume aggressive clock-gating. See Section 5.2 for details on system model.

While PwrSave and EnrSave only account for socket power and memory power, EnrSaveW uses wall power estimates into calculating energy savings. We use a quadratic function, discussed in Section 3.8 of Chapter 3, to estimate wall power from system power reported by the simulator.

As discussed above, cache insensitive workloads show negligible performance impact but significant power (and energy) savings. Cache sensitive workloads, on the other hand, show significant performance and energy losses, e.g., apache suffers a 39.7% performance loss and a 67.6% energy loss (EnrSaveW), soplex suffers a 49.3% performance loss and a 88.5% energy loss (EnrSaveW). Further, using a smaller cache does not always save system power—apache burns 2.1% more power. Overall, 19 of the 32 workloads waste energy if the cache is configured to the smallest size. The geometric mean of energy waste is 8.3% for multithreaded workloads and 33.9% for multiprogrammed workloads.

Thus, power budgeting must be done carefully, as a poor choice of cache configuration can drastically degrade performance and waste energy. Selecting a cache configuration that optimizes power-efficient performance is challenging since the optimal point varies for different workloads, and even for different phases within a given workload. Exhaustive evaluation is impractical due to the large number of feasible system configurations.

In this work, we use analytical models to estimate the power and performance of different configurations, allowing rapid prediction of the optimal system configuration. This requires four predictors: cache miss rate predictor, performance impact predictor, power impact predictor, optimal DVFS scaling predictor. The miss rate predictions are combined with simple performance and power models (Section 5.6) and an on-chip DVFS model (Section 5.7) to identify power-budgeted configurations that will maximize performance.

5.4 Operations overview

This work focuses on *improving* performance by shifting power from the last-level cache to the cores. For cores, we use conventional on-chip DVFS techniques, similar to those used in Intel's Turbo Boost [111], to run cores at higher voltage and frequency. For caches, we use power-gating [162] to eliminate all power for disabled cache regions.

While power-gating can enable a higher power margin for utilization, it results in dirty data being written back to memory when down-sizing the cache and non-trivial warmup time after up-sizing the cache. Assuming a sustained memory bandwidth of 8.53 GBPS (half of maximum bandwidth in the baseline) a worst-case scenario with 32MB of dirty data needs ~3.8 msec writeback time. To keep performance and energy overheads small ($< 1\%$) the reconfiguration interval should be at least 380 msec. *This automatically precludes reacting to high frequency events such as context switches that may occur between 300 to 5000 times per second* [64]. In contrast, drowsy mode [77] can enable small reconfiguration intervals with low overhead, but its need to maintain a minimum data retention voltage (DRV) limits the available power margin.

Reconfiguration being *costly* and *infrequent*, trial-and-error search for the optimal configuration at runtime is not appealing. Stepwise adaptation [163] may progress through multiple intermediate states. In contrast, this work uses online power-performance predictors that enable *one-shot* reconfiguration. They have the following strengths:

- Ability to predict performance for caches larger or smaller than the currently active configuration so that the optimal configuration can be reached in one step.
- Ability to predict the effect of changes in the number of sets of the cache.
- Ability to work with implementable cache replacement policies, such as PLRU. Prior work [155] has assumed LRU replacement, which is impractical to implement

1. Concurrently,
   a) Specialized hardware (Section 4.6.1 of Chapter 4) tracks reuse distance distribution for L3 accesses.
   b) Simple hardware performance counters track activity factors of cores and caches.
   c) Simple hardware performance counters track correlation between miss rate and CPI.
2. Periodically (e.g., once per second), invoke a software routine (ISR) to determine optimal cache configuration:
   a) Predict miss rates (Section 5.5) using information from 1(a).
   b) Predict performance and power (Section 5.6) using information from 1(b), 1(c), 2(a).
   c) Predict DVFS scaling and possible gain (Section 5.7) using information from 1(c), 2(a), 2(b).
3. Reconfigure system with the best predicted configuration if predicted gain is $> 2\%$. Write back dirty cache blocks as needed.
4. System continues to monitor predicted and actual performance and power metrics to detect and adjust in case of incorrect predictions (e.g., due to phase change effects).

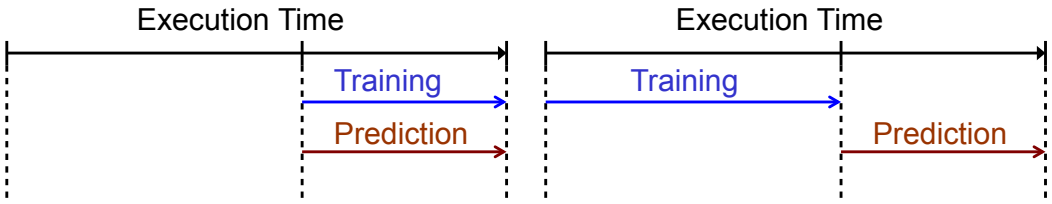
Figure 5.4: Operations Overview.

for highly-associative caches.

Figure 5.4 shows an operational overview of our proposed system. Execution time is logically partitioned into intervals. Like most predictors, our work uses past execution behavior to predict behavior in subsequent intervals. Simple hardware mechanisms are used to observe execution characteristics that are then used by prediction software (ISR) to determine the optimal system configuration. We refer to the interval used to train the predictors as the *Training Interval* and the interval where the results of prediction are applied as the *Prediction Interval*.

Intervals should be long enough so that predictor and reconfiguration overheads are small (< 1%). Predictor compute time depends on the number of target cache configurations evaluated. This has two components: miss-rate prediction (< 0.5 msec, using Section 4.5.5, Chapter 4, results for 25 configurations) and DVFS scaling computation (< 0.3 msec, see Section 5.7.4). Together with cache reconfiguration overhead (< 3.8 msec), the total time overhead < 3.8 + 0.5 + 0.3) = 4.6 msec. A small amount of additional time is needed to read various performance counters. So, we recommend an interval length > 500 msec. Longer intervals corresponding to 1 or more seconds of execution may be needed to get sufficient samples. Our work targets optimizing system operations for the average execution profile over long-term intervals.

There are two sources of error in the predicted values: *model error* and *phase error*. Model error results from simplifying assumptions (e.g., independence and identical distribution) that may not strictly hold in practice. Phase error also includes changes in workload behavior as it moves though different phases of execution [195]. This distinction helps identify which errors could be reduced by further refining the model and which errors are orthogonal to it. When phase changes occur the behavior from a previous interval is not a good predictor for the next interval. The predictors may be improved using previously proposed online phase detection mechanisms [161].



(a) **Model Error Prediction** (b) **Phase Error Prediction**

Figure 5.5: Training and Prediction intervals.

Model Error predictions use the same interval for training and prediction and hence

incur model error alone. Such predictions offer insight into model accuracy, but are useless for reconfiguration purposes. Phase Error predictions use a preceding training interval to predict the behavior of a subsequent prediction interval and includes both model and phase errors. These predictions are used to reconfigure the system. Figure 5.5 illustrates training and prediction intervals. In the rest of this chapter, we will use the term Model Error and Model Error prediction interchangeably and likewise with Phase Error and Phase Error prediction.

For both Model Error and Phase Error predictions, we study *absolute error* and *relative error*. The absolute error is (predicted value - actual value). Relative error is (absolute error/actual value). The actual and predicted values can be inferred using the calculation: actual value = (absolute error/relative error).

5.5 Cache miss rate prediction

Predicting the cache miss rate for all possible cache configurations is critical to our power-budgeting approach. We use our reuse distance based cache miss ratio predictor, described in Chapter 4, for this purpose. *This section elaborates step 2(a) of Figure 5.4.*

Since our simulation runs are not very long, we do not perform address sampling to estimate the reuse distributions. Instead, we assume that the full reuse distributions are available for use as inputs to our miss rate predictors. A practical deployment would need to use sampling, as described in Chapter 4, over longer runs. With a sufficient number of samples, the inferred reuse distributions should be close to the actual distributions.

Figure 5.6 shows reuse distance distributions for our workloads in the Prediction Interval. The dashed vertical lines with size annotations indicate fully-associative LRU cache sizes that would be necessary if all accesses with reuse distances less than or equal to that point must hit. Cache insensitive workloads, such as blac and equa, have “flat”

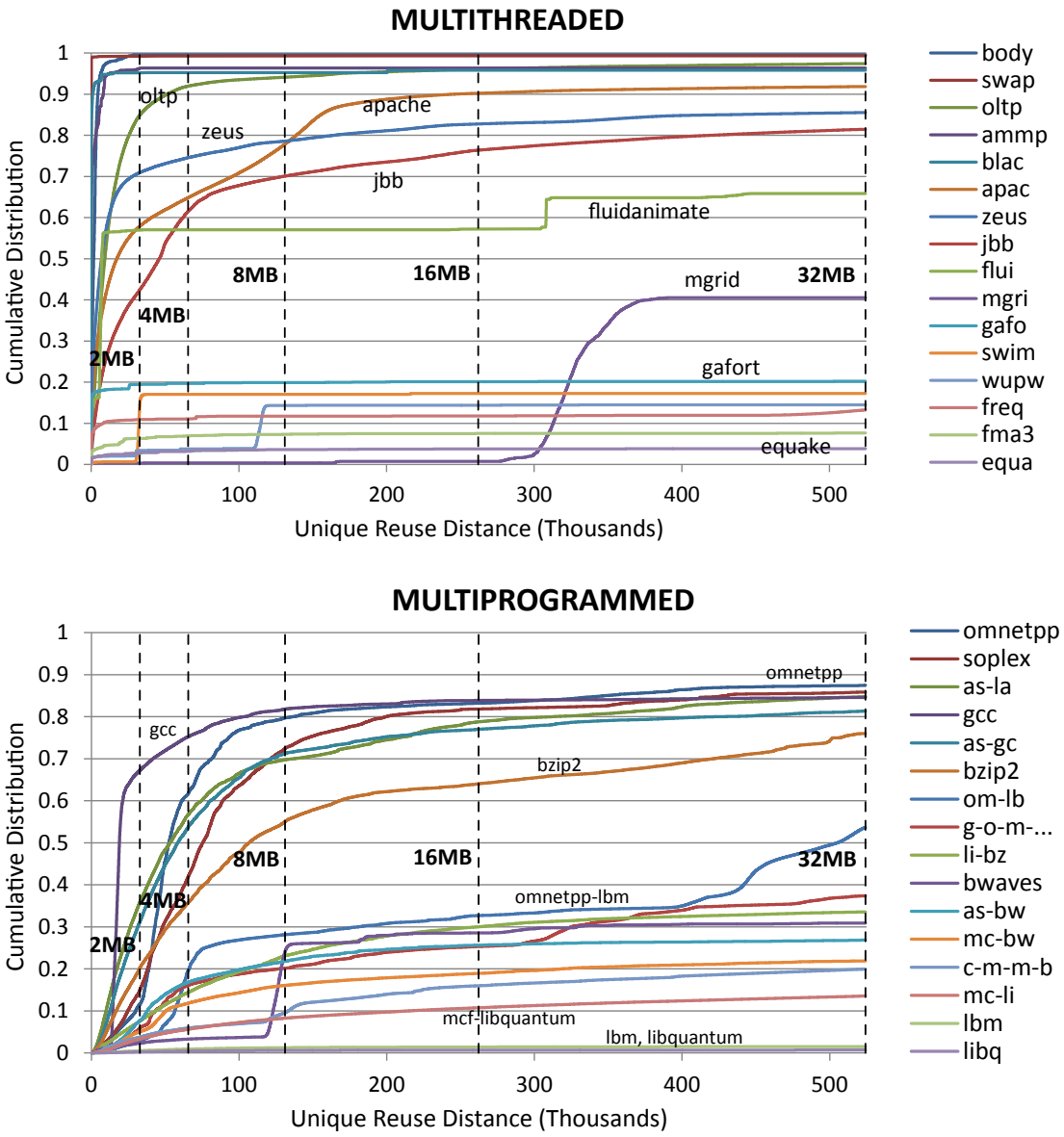


Figure 5.6: Reuse distributions (cumulative) in the Prediction Interval. Entries in the legend are ordered according to the intercept of the distributions on the right vertical axis. Additionally, a few distributions are labeled with the workload name to enhance readability. A 32MB cache has 512K lines.

reuse distributions over the range of cache sizes that we consider whereas cache sensitive workloads show a significant slope.

Good miss rate predictions should not have high absolute errors. That is, the difference between the predicted and actual MPKI should be small. Since cache miss rates significantly affect overall performance, large absolute errors in MPKI prediction will also translate into large errors in performance prediction that can lead to poor configuration selections. Note that relative errors can be large for workloads with small miss rates but have little effect on performance if the absolute errors are small.

For this study, we consider cache reconfiguration in the number of sets, but not in the number of ways. Our caches always have 32-way associativity for all cache sizes. This is because for small caches, smaller associativities usually cause more conflict misses than with larger associativities. Since our cache hierarchy is inclusive, evictions at the LLC can use evictions at L2 which can then result in increased demand accesses from L2 to L3 that subsequently miss in L3. The effect of this is two-fold:

1. 2MB 2-way LLCs can witness significantly more accesses and misses than 2MB 32-way caches while saving negligible power due to reduced associativity, and

2. miss rate predictions using the characteristics of the address stream with the 32MB 32-way LLC are no longer accurate since the nature of the access stream to the LLC is altered.

To elaborate on the above points, we momentarily assume that our LLC is configurable in both the number of sets and ways. Figure 5.7 shows the number of accesses (misses from L2) to a 2MB 2-way LLC and to a 2MB 32-way LLC, normalized to the number of accesses to the LLC in the baseline configuration (32MB 32-way). Figure 5.8 shows the corresponding MPKI values. Ideally, there should be no differences in the number

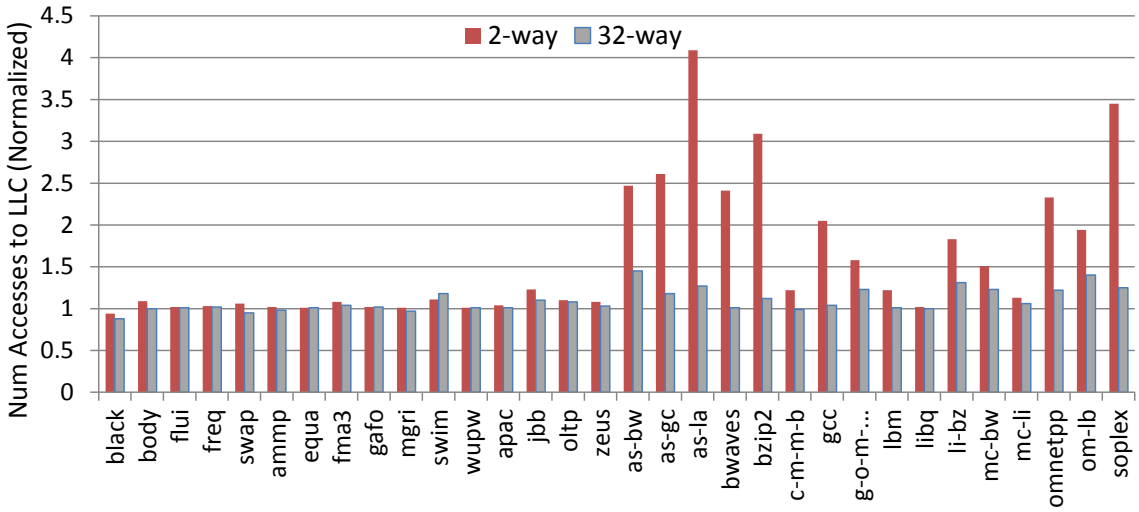


Figure 5.7: Number of accesses for a 2MB LLC with small and large associativities, normalized to the number of accesses to the baseline 32MB 32-way LLC.

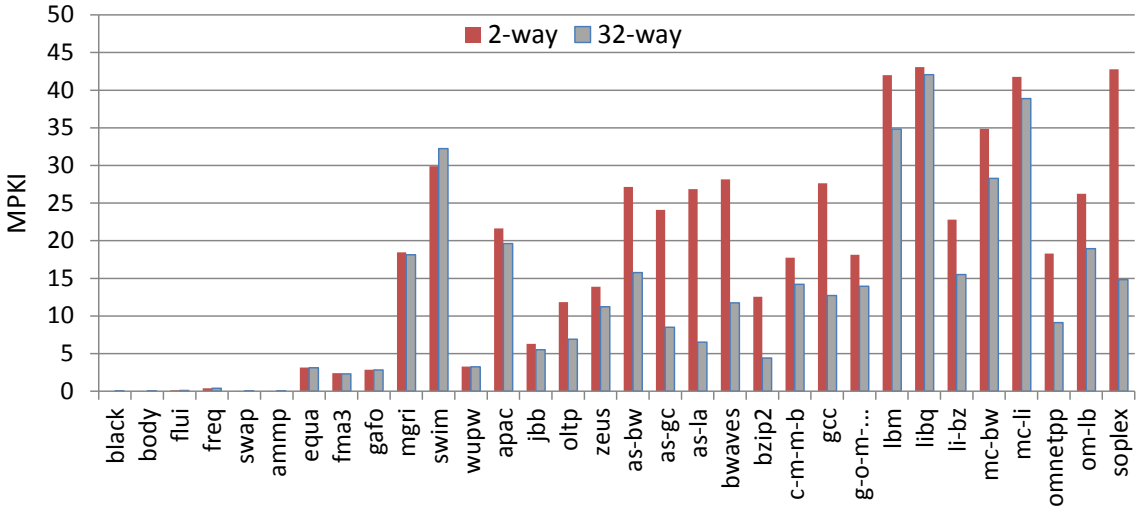


Figure 5.8: MPKI for a 2MB LLC with small and large associativities.

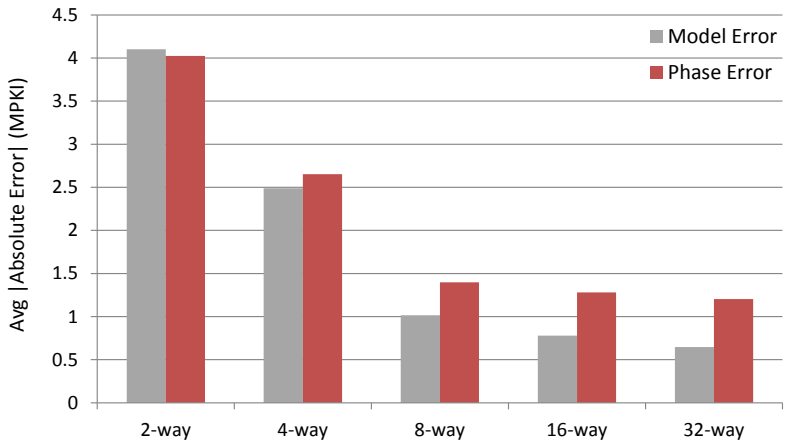


Figure 5.9: Averages of Absolute Error for miss ratio estimation.

of accesses to the LLC. However, due to a strictly inclusive cache hierarchy and more conflicts, the 2-way cache witnesses more accesses than the 32-way cache. The effect is more pronounced for the multiprogrammed workloads than for the multithreaded workloads. Large changes in the access stream contribute to large changes in MPKI values in addition to that due to changed cache organization.

Figure 5.9 shows the average of absolute errors in MPKI prediction for both Model Error and Phase Error over all workloads for different associativities and all cache sizes (2MB, 4MB, 8MB, 16MB, 32MB) for each associativity. We observe that for both Model Error and Phase Error, predictions for 32-way caches are significantly more accurate than for 2-way caches. For Model Error, the average error increases from 0.65 MPKI to 4.1 MPKI—a 6.3× change. For Phase Error, the average error increases from 1.2 MPKI to 4 MPKI—a 3.3× change.

Figures 5.10 and 5.11 show prediction error distributions for Model Error and Phase Error respectively for different associativities over all cache sizes that we consider. The predictions become more accurate at higher associativities. This is seen in the Absolute

Error vs. Relative Error scatter plots where the data points become more concentrated around the origin as associativity increases. It is also seen in the cumulative distribution plots where the worst case error decreases and the curves reach 100% more quickly as associativity increases.

For the rest of this chapter we only consider 32-way caches. Figures 5.10 and 5.11 show that the absolute error for Model Error is < 7 MPKI with 90% of errors within 2.4 MPKI. For Phase Error, the worst-case error is ~10 MPKI with 90% of errors within 3.74 MPKI. The Phase Error charts also show a few points with high relative error (swap) but this happens with at small MPKI values and result in small absolute errors, so the performance impacts of mispredictions are small.

5.6 Performance and Power prediction models

Predicting miss rates for target cache configurations is necessary, but not sufficient for making power-budgeting decisions; it is also necessary to predict performance and power impact. *This section elaborates step 2(b) of Figure 5.4.*

To predict performance given an L3 miss rate r for an instruction interval, let $\hat{CE}(r)$ denote the estimated number of cycles to complete that interval. $\hat{CE}(r) = \hat{CPI}(r) \times$ number of instructions (aggregate, over all cores). We approximate $\hat{CPI}(r)$ by measuring the actual L3 misses, cycles and instructions committed during the training interval using hardware performance counters and then using least-squares regression to fit this to a simple linear model so that $\hat{CPI}(r) = g + r \times h$ for estimated constants g and h.

For a set of n tuples of the form (x_i, y_i), $i = 1..n$, the constants can be calculated with

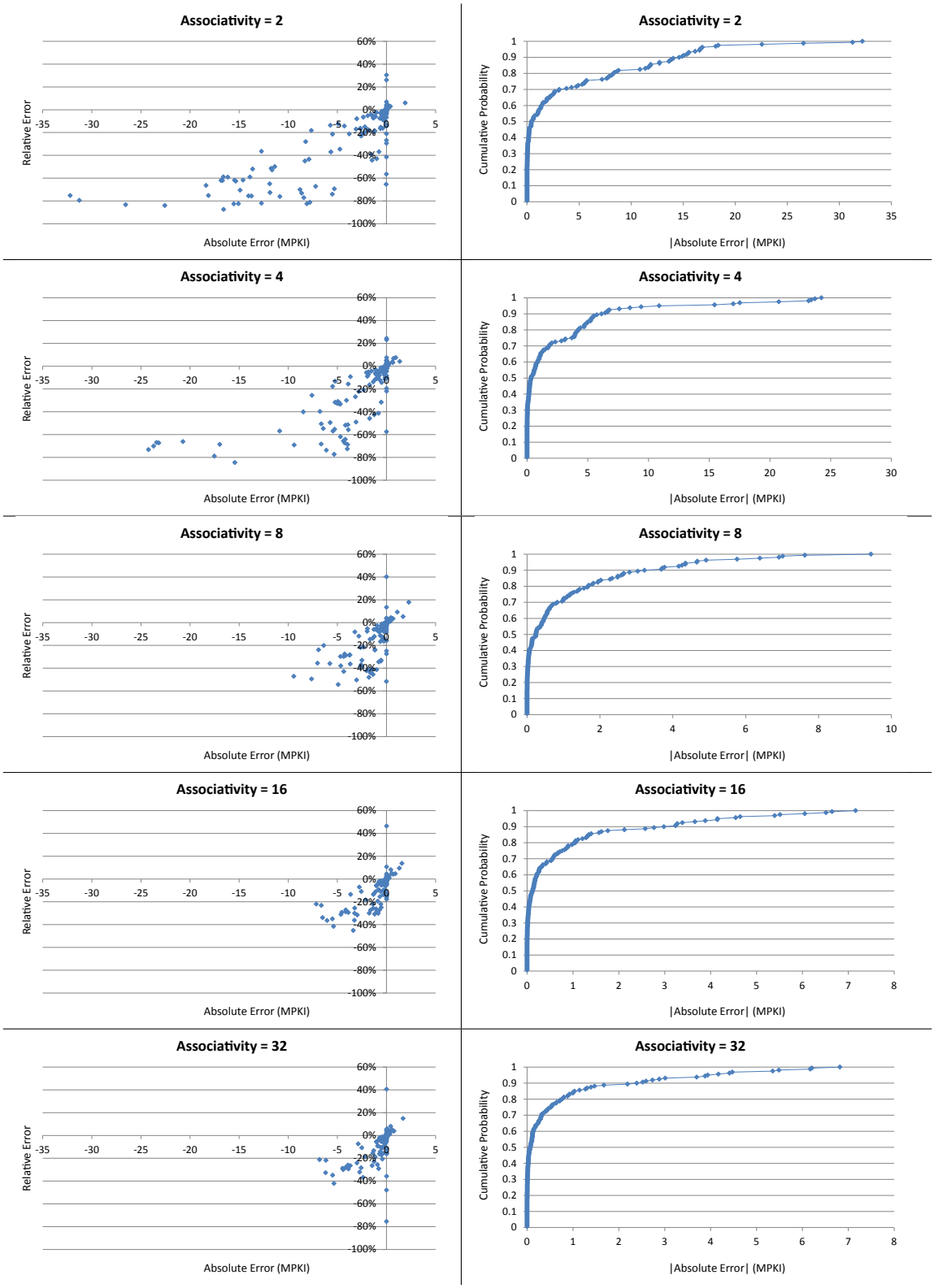


Figure 5.10: Model Error for miss ratio estimation.

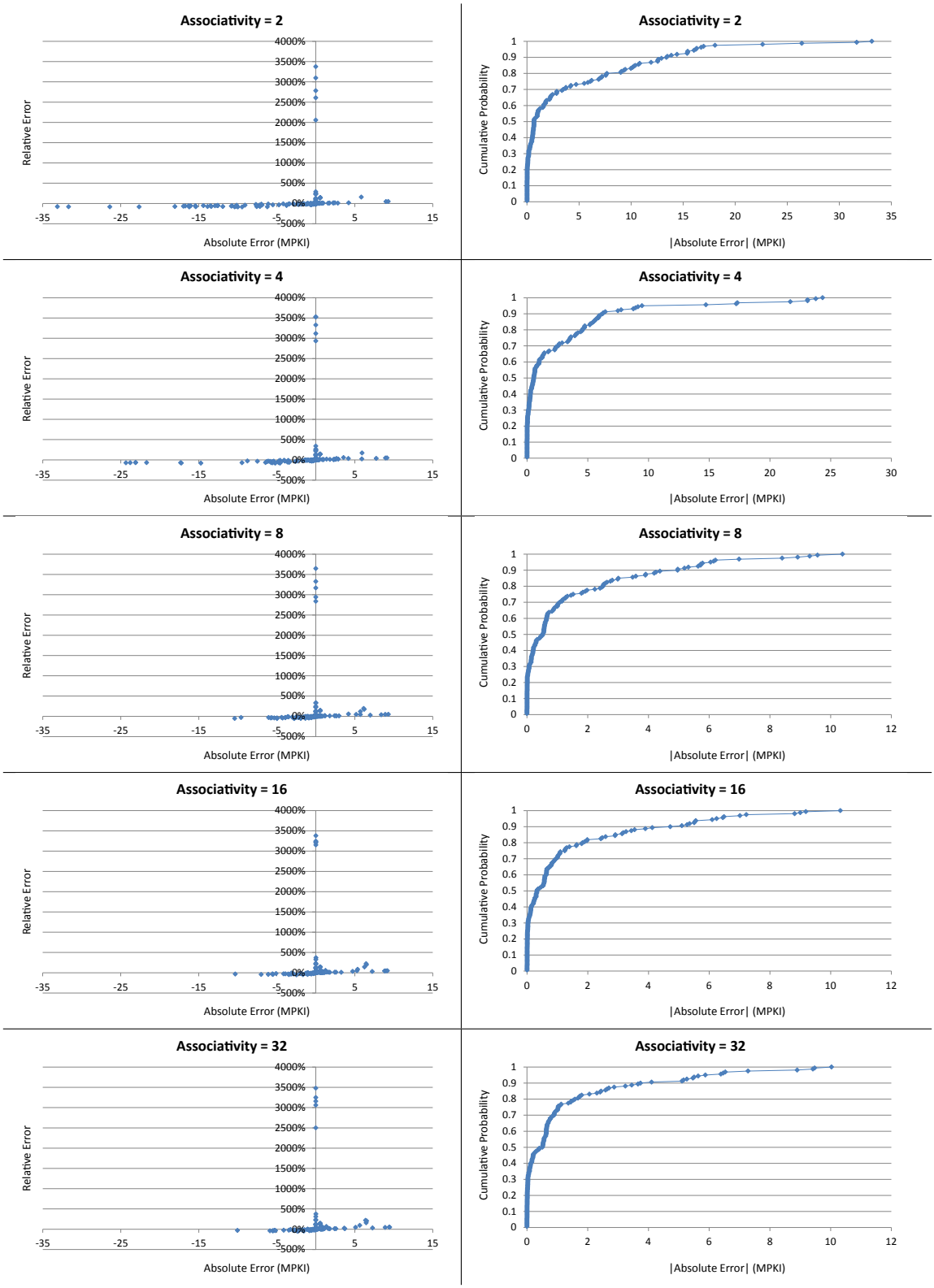


Figure 5.11: Phase Error for miss ratio estimation.

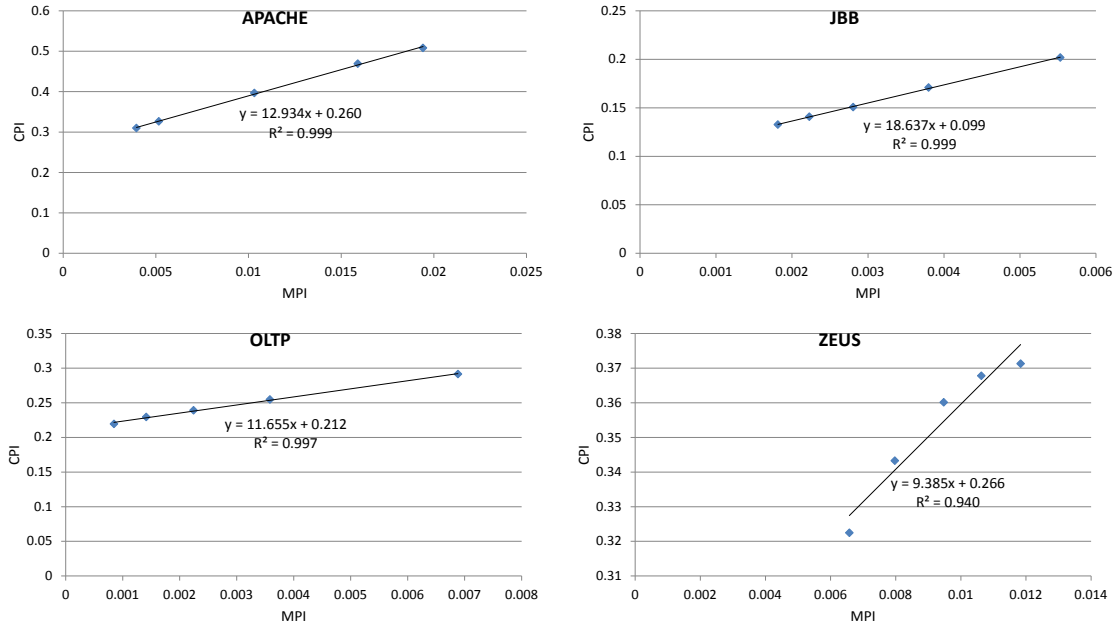


Figure 5.12: CPI regression for commercial workloads in the Training Interval

the following equations.

$$h = \frac{\frac{\sum_{i=1}^{n} x_i y_i}{n} - \left(\frac{\sum_{i=1}^{n} x_i}{n}\right)\left(\frac{\sum_{i=1}^{n} y_i}{n}\right)}{\frac{\sum_{i=1}^{n} x_i^2}{n} - \left(\frac{\sum_{i=1}^{n} x_i}{n}\right)^2} \tag{5.5}$$

$$g = \frac{\sum_{i=1}^{n} y_i}{n} - h\left(\frac{\sum_{i=1}^{n} x_i}{n}\right) \tag{5.6}$$

The above estimation is done online. At the end of every 20,000 cycles, two metrics are computed: (x_i = Δ LLC misses/Δ instructions committed) and (y_i = Δ cycles/Δ instructions committed). The individual (x_i, y_i) tuples are not stored. Instead, cumulative metrics (product, square, sum) with earlier values are computed. 4 registers and 1 counter (for tracking total number of tuples) are maintained. At the end of the training interval, the slope (h) and offset (g) of the best-fit line are computed.

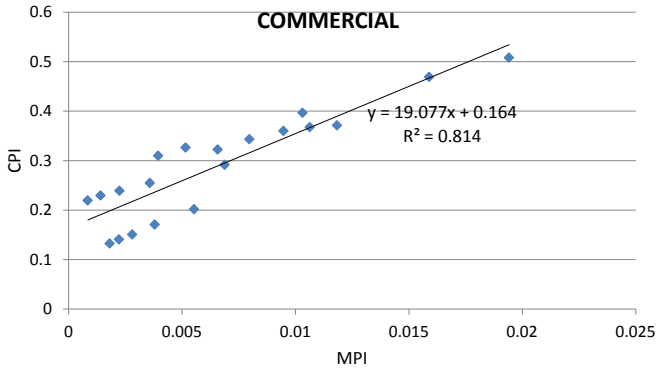


Figure 5.13: Combined CPI regression for commercial workloads in the Training Interval

The linear approximation does not always succeed. In some cases due to variations in program behavior, low MPKI, or lack of variability in the tuples, the slope may be computed as negative. This is unrealistic as we expect execution time to increase with MPKI due to the long off-chip miss latency. In such cases, the computed slope is discarded and instead a predetermined value is chosen. We chose this to be the value estimated for the commercial workloads over the Training Interval using oracle information. We get this oracle information by simulating the workloads for all possible cache sizes in the Training Interval and measuring the MPI and CPI values. Figure 5.12 shows the individually fitted lines as well as R^2 values for each commercial workload. The R^2 values are close to 1 indicating good fits. Figure 5.13 shows the fitted line over data points from all four commercial workloads. The fit is less good (lower R^2 value), but we use this as a representative for average program behavior. Also, we predict CPI to be the same if predicted MPKI is within 0.01 or 1% of the current configuration.

Figures 5.14 and 5.15 show Model Error and Phase Error in CPI estimation over all workloads using the online estimator. The average of absolute values of relative errors for Model Error is 4.7% whereas for Phase Error it is 9.9%. For Model Error, 90% of

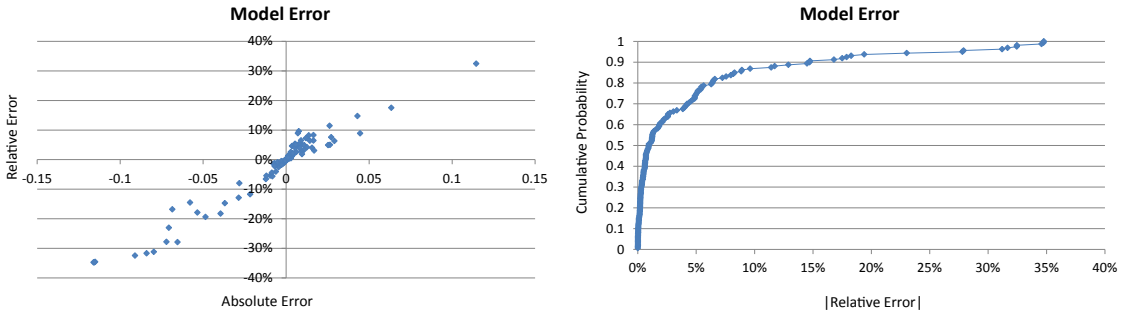


Figure 5.14: Model Error for CPI estimation.

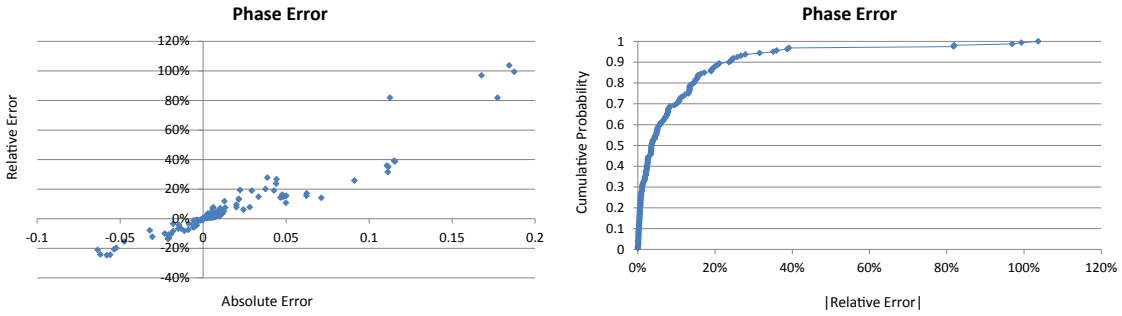


Figure 5.15: Phase Error for CPI estimation.

predictions have within 14.7% relative error whereas for Phase Error it is 23.6%.

For power predictions, we separately predict static power and dynamic power. We use CACTI to estimate static power and dynamic energy per activation per component. For the L3 cache, the number of tag and data activations for accesses, misses, replacements and coherence activities is tracked. To predict activations, the model makes some simplifying assumptions, e.g., L3 accesses, coherence activities, and the percentage of L3 misses that cause additional writebacks to memory is the same as that observed in the current configuration. These assumptions are not strictly true but work reasonably well.

Figures 5.16 and 5.17 show Model Error and Phase Error in system power estimation over all workloads using the online estimator. The average of absolute values of relative

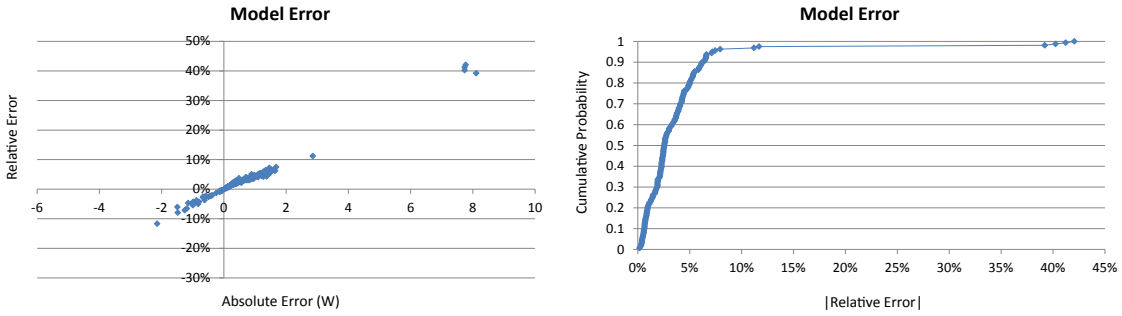


Figure 5.16: Model Error for system power estimation.

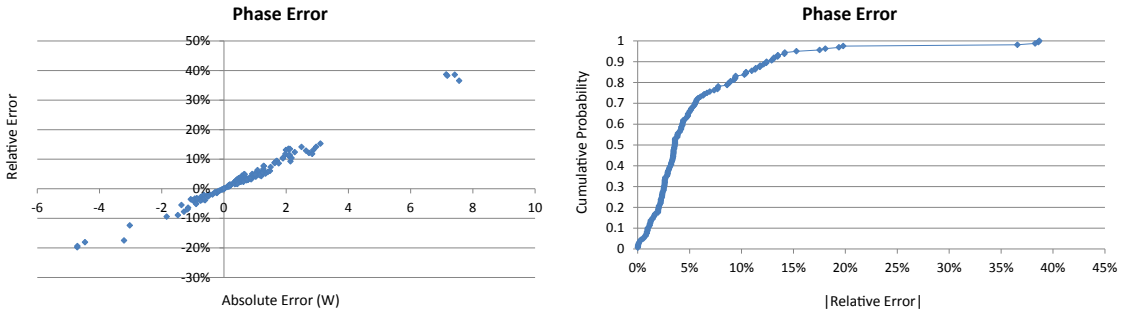


Figure 5.17: Phase Error for system power estimation.

errors for Model Error is 4% whereas for Phase Error it is 5.8%. For Model Error, 90% of predictions have within 6.2% relative error whereas for Phase Error it is 12.4%. Since static power is known, the errors are due to dynamic power estimations. This has two components: activation count estimation and performance estimation. Improving either will reduce errors.

5.7 Model-driven Power Budgeting

The analytical models of the preceding sections determine the power and performance of different cache configurations for a given workload. Here we describe how to estimate whether or not it is better to reconfigure the cache to a smaller configuration that uses less

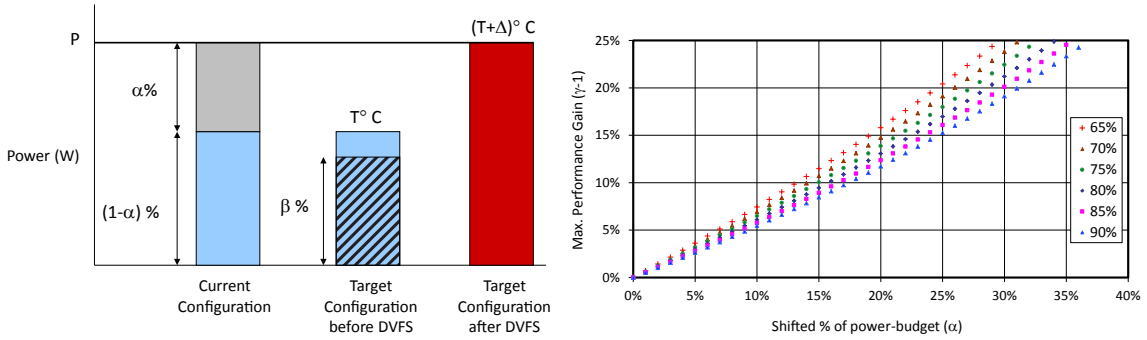


Figure 5.18: Max. performance gains with basic model (Section 5.7.1). Each series corresponds to a different value of β, from $\beta = 65\%$ to $\beta = 90\%$. $D_{old} = \beta(1-\alpha)P$.

power, leaving more power available to run the core at a higher voltage and frequency. *This section elaborates step 2(c) of Figure 5.4.*

We develop our power-budgeting model in three steps, each step adding some more complexity to the previous one. First, Subsection 5.7.1 presents a basic model for calculating maximum performance gains in a power-budgeting environment where power gating and on-chip DVFS are used for shifting power. Next, Subsection 5.7.2 applies it to power budgeting for on-chip resources and includes the effects of temperature rise with frequency scaling on static power estimations. Finally, Subsection 5.7.3 includes main memory power in the power budget. We use only on-chip DVFS. Memory voltage and frequency are not scaled.

5.7.1 Basic model

Let P be the power-budget (assumed to be fully utilized by the current configuration) and let α denote the fraction of this budget that can re-budgeted for better performance. The power margin, αP, can include both static *and* dynamic power of the current configuration. Ideally, DVFS transforms all of αP into additional dynamic power in

the target configuration. However, there are leakage losses due to temperature and voltage rise and performance losses due to non-scaling of memory latency. Figure 5.18 shows a schematic of power shifted from the old (current) configuration to the new (target) configuration. After DVFS, the dynamic power of the target configuration, $D_{new} \leqslant D_{old} + \alpha P$. Since dynamic power is proportional to $V^2 f$, we have

$$\frac{D_{new}}{D_{old}} = \left(\frac{V_{new}}{V_{old}}\right)^2 \left(\frac{f_{new}}{f_{old}}\right) \leqslant \left(1 + \frac{\alpha P}{D_{old}}\right) \tag{5.7}$$

Let $\gamma = \frac{f_{new}}{f_{old}}$. So, *Speedup* $\leqslant \gamma$. Using published data [99] for voltage-frequency pairs for the Pentium M, we assume that $(V_{new} - V_{old}) \propto (f_{new} - f_{old})$ and that every 200MHz change in frequency is accompanied by a 50mV change in voltage. Thus, $V_{new} = V_{old} + 50mV \left(\frac{f_{new} - f_{old}}{200MHz}\right)$. We also assume a base operating voltage of 0.9V and a base operating frequency of 2132MHz. Substituting values in Equation 5.7,

$$\left(\frac{V_{new}}{V_{old}}\right) = (1 + 0.5922(\gamma - 1)) \tag{5.8}$$

$$(1 + 0.5922(\gamma - 1))^2 \gamma \leqslant \left(1 + \frac{\alpha P}{D_{old}}\right) \tag{5.9}$$

Figure 5.18 shows maximum performance gains with $\gamma \leqslant 1.25$. The gains increase with both α and β.

5.7.2 On-chip power-budgeting model

We will now extend the basic model by including the effects of temperature rise with DVFS and will apply it to on-chip power budgeting. For on-chip power-budget P, current operating voltage V, frequency f, temperature T and target cache configuration $\bar{C} = (capacity, associativity)$, let $\hat{P}_{st}(\bar{C}, T)$ and $\hat{P}_{dyn}(\bar{C}, V, f)$ denote the estimated on-chip static and dynamic power respectively. The dynamic power before DVFS,

$D_{old} = \hat{P}_{dyn}(\bar{C}, V, f)$. The power margin, $\alpha P = P - \hat{P}_{st}(\bar{C}, T) - \hat{P}_{dyn}(\bar{C}, V, f)$ can be utilized by increasing the operating voltage and frequency to V_{new} and f_{new} respectively. However, scaling is accompanied by an increase in static power that reduces the available power margin due to temperature and voltage increase.

Since chip power-budget is fixed, ideally, overall chip temperature cannot rise (Stefan-Boltzmann law). However, since processor chips are not ideal black-bodies, the higher dynamic power of the cores can cause the operating temperature to rise locally. This in turn increases the static power dissipation by $\Delta\hat{P}_{st}(T) = \hat{P}_{st}(\bar{C}, T_{new}) - \hat{P}_{st}(\bar{C}, T)$. $\Delta\hat{P}_{st}(T)$ depends on γ. For our implementation we assumed a maximum temperature rise of 8 °C corresponding to a maximum scaling of 533MHz (2132MHz to 2665MHz) with 3 °C contributed by every 200MHz [99]. We assumed a continuous scaling domain with temperature rise proportional to scaling. Thus, $\Delta\hat{P}_{st}(T) = (\frac{\gamma-1}{1.25-1}) \times (\hat{P}_{st}(\bar{C}, T+8) - \hat{P}_{st}(\bar{C}, T))$. By default, CACTI allows modeling temperature effects in steps of 10K. We used linear interpolation to obtain static power at other points. So,

$$\Delta\hat{P}_{st}(T) = \left(\frac{\gamma-1}{0.25}\right) \times \frac{8}{10} \times (\hat{P}_{st}(\bar{C}, T+10) - \hat{P}_{st}(\bar{C}, T)) \tag{5.10}$$

The higher operating voltage increases static power dissipation. Assuming a linear model [38], the increase is $\Delta\hat{P}_{st}(V) = \left(\frac{V_{new}}{V_{old}} - 1\right) \times \hat{P}_{st}(\bar{C}, T))$. Combining with Equation 5.8, we have

$$\Delta\hat{P}_{st}(V) = 0.5922 \times (\gamma - 1) \times \hat{P}_{st}(\bar{C}, T) \tag{5.11}$$

Both $\Delta\hat{P}_{st}(T)$ and $\Delta\hat{P}_{st}(V)$ reduce αP. Plugging values into Equation 5.9, we get

$$(1 + 0.5922(\gamma - 1))^2\gamma \leqslant \left(\frac{P - \hat{P}_{st}(\bar{C}, T) - \Delta\hat{P}_{st}(T) - \Delta\hat{P}_{st}(V)}{\hat{P}_{dyn}(\bar{C}, V, f)}\right) \tag{5.12}$$

Since *memory voltage and frequency are not scaled*, the number of cycles to execute the same task in the scaled configuration is increased due to scaling of memory latency (in terms of processor cycles). This reduces the increase in on-chip dynamic power that Equation 5.12 assumes. We use this fact to improve Equation 5.12. With a linear performance predictor model, $\hat{CPI}(r) = g + r \times h$. After scaling, $\hat{CPI(r)}_{scaled} = g + \gamma \times r \times h$. So,

$$(1+0.5922(\gamma-1))^2\gamma \leqslant \left(\frac{g+\gamma\times r\times h}{g+r\times h}\right) \times \left(\frac{P - \hat{P}_{st}(\bar{C},T) - \Delta\hat{P}_{st}(T) - \Delta\hat{P}_{st}(V)}{\hat{P}_{dyn}(\bar{C},V,f)}\right) \quad (5.13)$$

5.7.3 System power-budgeting model

We will now include memory power in addition to on-chip power in the power budget. We start from Equation 5.13, noting that since memory voltage and frequency are not scaled, $\hat{P}_{dyn}(\bar{C},V,f)$ *still refers to on-chip dynamic power*. However, increase in memory power can reduce the available power budget. Let P', $\hat{P}_{mbp}(\bar{C})$ and $\hat{P}_{map}(\bar{C},\gamma f)$ denote the available system power-budget, estimated memory background power and estimated memory active power. Equation 5.13 can now be reformulated as

$$\begin{aligned}(1+0.5922(\gamma-1))^2\gamma \leqslant \left(\frac{g+\gamma\times r\times h}{g+r\times h}\right) \times \\ \left(\frac{P' - \hat{P}_{st}(\bar{C},T) - \Delta\hat{P}_{st}(T) - \Delta\hat{P}_{st}(V) - \hat{P}_{mbp}(\bar{C}) - \hat{P}_{map}(\bar{C},\gamma f)}{\hat{P}_{dyn}(\bar{C},V,f)}\right)\end{aligned} \quad (5.14)$$

For I instructions, the expected execution time with the target configuration is $I \times \left(\frac{g+\gamma\times r\times h}{\gamma}\right)$ with frequency scaling γ and $I \times (g + r \times h)$ without frequency scaling. So,

$$\hat{P}_{map}(\bar{C},\gamma f) = \gamma \times \left(\frac{g+r\times h}{g+\gamma\times r\times h}\right) \times \hat{P}_{map}(\bar{C},f) \quad (5.15)$$

Equation 5.15 is plugged into Equation 5.14 for the final equation.

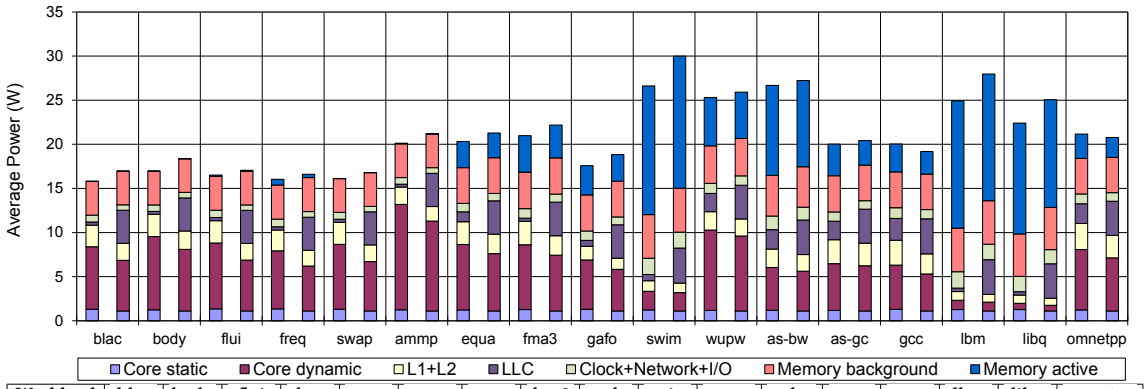
5.7.4 Results and Limitations

Equation 5.14 can be rearranged in the form $f(\gamma) \leqslant 0$ and solved in software using known techniques for solving cubic equations. To keep overheads low, we recommend using enumeration for $f(\gamma)$: for each allowed frequency step, evaluate $f(\gamma)$ and check the sign of the result. A change in sign between consecutive steps indicates a solution point. Each evaluation takes < 5 µsec on a Nehalem (2.26GHz) machine. We have 11 frequency steps over the frequency range that we consider. Thus, the DVFS computation time per cache configuration is < 0.06 msec and < 0.3 msec over all 5 cache configurations.

Figure 5.19 shows the performance gains and power-budget utilization of the best predicted configurations when optimizing performance subject to a system power budget. The baseline is the system with the highest-power LLC (32MB 32-way). The Table at the bottom of the figure includes the configurations selected.

Fifteen workloads show 0.5% to 15.2% performance improvement over the baseline. There is some under-utilization of the baseline power budget, leading to positive values of PwrSave numbers, due to conservative assumptions about the effect of DVFS on combinational logic and errors in expected performance and power of the target configurations. All 17 workloads for which a configuration different than the baseline was predicted saved energy (EnrSaveW) from 1.3% to 14.6%. The remaining workloads continued with the baseline configurations and did not save energy. The geometric mean of energy savings over all 32 workloads was 4.4%.

Unfortunately, the predicted configurations led to performance loss for swim and lbm and more power being used for gcc and omnetpp. Inaccurate predictions may lead to performance loss, under/over-utilization of the power budget and associated thermal overshoot. This situation is not unique to our system: any non-oracular predictive mechanism would need to detect and deal with occasional mispredictions.



Workload	blac	body	flui	freq	swap	ammp	equa	fma3	gafo	swim	wupw	as-bw	as-gc	gcc	lbm	libq	omnetpp
CacheSize	2MB	2MB	2MB	2MB	2MB	2MB	8MB	2MB	4MB	4MB	16MB	16MB	16MB	16MB	2MB	2MB	16MB
FreqScale	14.1%	9.4%	16.4%	16.4%	14.1%	9.4%	9.4%	11.7%	14.1%	9.4%	4.7%	7.0%	4.7%	14.1%	11.7%	11.7%	9.4%
PerfGain	14.1%	9.3%	15.2%	9.2%	14.0%	10.1%	7.4%	7.4%	8.1%	-3.3%	3.9%	2.3%	0.5%	7.6%	-1.1%	2.4%	6.5%
PwrSave	5.9%	6.8%	2.0%	2.4%	3.0%	4.1%	3.5%	4.2%	5.4%	10.8%	1.9%	1.6%	1.7%	-5.4%	10.4%	10.2%	-2.6%
EnrSave	17.5%	14.7%	14.9%	10.6%	14.9%	12.9%	10.2%	10.8%	12.5%	7.8%	5.6%	3.8%	2.2%	2.1%	9.4%	12.3%	3.6%
EnrSaveW	14.6%	11.4%	13.9%	9.3%	13.4%	11.1%	8.6%	9.0%	9.8%	3.6%	4.9%	3.2%	1.3%	4.7%	5.3%	7.9%	4.8%

Figure 5.19: System power budgeting (Section 5.7.3) with best predicted configuration (NI prediction). Two power stacks are shown for each workload: the left stack for the predicted configuration and the right stack for the baseline configuration (32MB 32-way LLC, 2132 MHz frequency). For each workload, the system power consumed by the baseline configuration is the power budget. **CacheSize** and **FreqScale** (= $\frac{\text{new frequency}}{\text{baseline frequency}}$ - 1) together define the predicted configuration. **PerfGain** shows performance gain (= speedup - 1), **PwrSave** shows system power saved (= remaining system power budget with respect to the baseline), and **EnrSave** shows system energy saved with the predicted configuration compared to the baseline. **EnrSaveW** accounts for wall power in energy savings calculations. Negative improvements (= losses) are highlighted.

We will now compare our model predictions to that by an oracle. The oracle runs all possible configurations (#configurations = #cache sizes $\times$ #frequencies) to determine the highest-performing configuration that satisfies the power budget. The oracle identifies configurations on the Pareto frontier. Figure 5.20 shows the performance gains with respect to the baseline configuration with the oracle and our model for all predictions where at least 2% gains were predicted, either by the oracle or the model, while being within the power budget. For example, the oracle predicts 16.5% gains for blac whereas

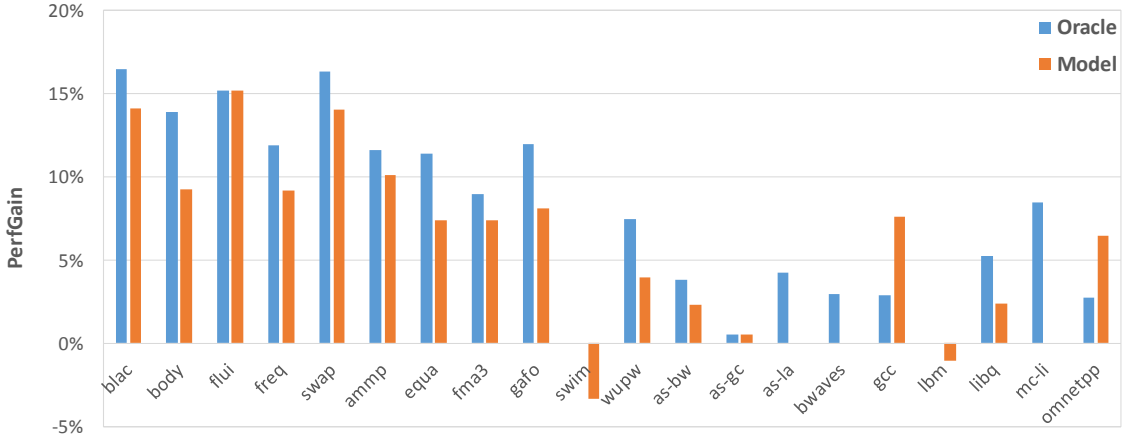


Figure 5.20: Comparison of performance gains between the oracle and our model.

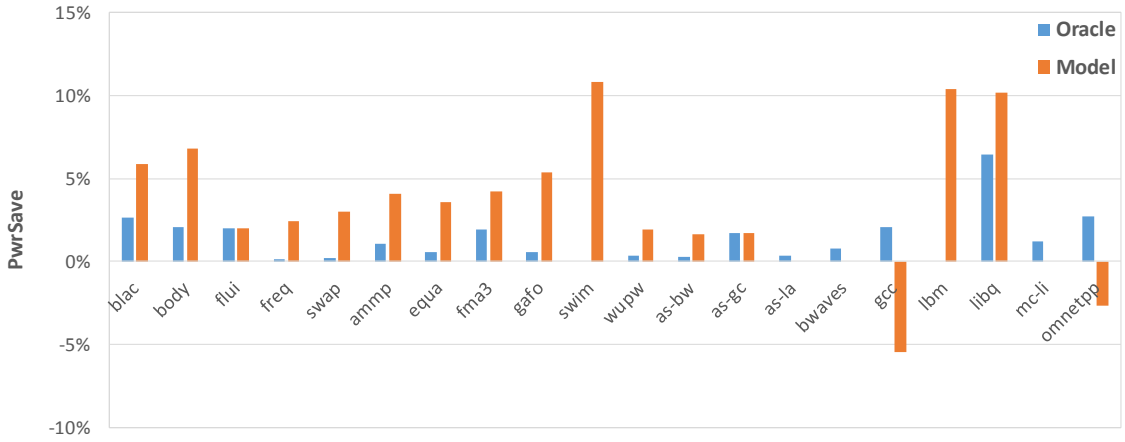


Figure 5.21: Comparison of power savings between the oracle and our model.

the configuration selected by our model achieved 14.1%. The oracle never selects a performance-losing configuration. Our model, on the other hand, selected configurations that caused performance losses on two applications—swim (3.3%) and lbm (1.1%). The model-selected configurations performed better than the oracle for two benchmarks—gcc

	Oracle PerfGain > Model PerfGain	**Oracle PerfGain < Model PerfGain**
Oracle PwrSave > Model PwrSave	as-la, bwaves, mcf-li	gcc, omnetpp
Oracle PwrSave < Model PwrSave	blac, body, freq, swap, ammp, equa, fma3, gafo, swim, wupw, as-bw, lbm, libq	

Figure 5.22: Comparison matrix between the oracle and our model.

(model: 7.6%, oracle: 2.9%) and omnetpp (model: 6.5%, oracle: 2.8%), because they overshot the power budget by 5.4% and 2.6% respectively. The power savings, shown in Figure 5.21, shows what percentage of the power budget remains unutilized. The savings are negative, indicate overshoot of the power budget, for gcc and omnetpp. For a number of workloads, the unutilized power budget is higher with the model-selected configurations than with the oracle-selected configurations. This under-utilization leads to lower performance gains than the oracle for the corresponding workloads in Figure 5.20.

Figure 5.22 shows a qualitative comparison between the oracle and the model, correlated across performance gains and power savings (power budget under-utilization). There cannot be any workload for which the oracle gains less performance and saves less power than the model, so the bin in the lower-right quadrant is empty. The bin in the upper-left quadrant shows workloads (as-la, bwaves, mcf-li)) where the oracle gained more performance as well as saved more power than the model. The upper-right quadrant shows workloads for which the model performed better than the oracle, but saved less power. This can only happen if the power budget is overshot. The lower-left shows workloads where the model resulted in conservative behavior—less performance than the oracle, but more underutilized power budget. The remaining workloads had identical gains with the oracle and with the model. This includes two workloads, flui and as-gc,

with performance gains (15.2% and 0.5% respectively) with power budgeting and the rest of the workloads with no gains or less than 2% gains. Note that since gains for as-gc are within the minimum 2% threshold, this configuration is not actually selected by the oracle, but we include it here to maintain the invariant that the oracle is always at least as good as the model.

Of the workloads show in Figures 5.20 and 5.21, 11 workloads (blac, body, flui, swap, ammp, fma3, as-bw, as-gc, gcc, libq, omnetpp) had the same cache size selected by both the model and the oracle. Of these, flui and as-gc also had the same frequency selections. For gcc and omnetpp, the frequency selected by the model was higher than that selected by the oracle resulting in overshoot of the power budget. For the remaining 7 workloads, the model-selected frequency was lower than the oracle-selected frequency resulting in under-utilization of the power budget. Since frequency reconfigurations can be done with significantly less overhead than cache reconfigurations, they can be subsequently adjusted to stay within or better utilize the power budget.

We propose using online monitoring schemes similar to Intel's Turbo Boost technology [111] so that the system can continuously monitor and compare predicted power-performance with actual values. In case performance is below expectations, the system will revert back to baseline thereby restoring long-term performance loss to 0%. In case of power/thermal overshoot with strict budgets, throttling mechanisms must be employed to scale down excess voltage and frequency immediately. For soft budgets, corrective action can be taken in the next interval. A guard mechanism may be used along with our predictors to make the system energy-secure [36].

Our DVFS model assumes that enough thermal headroom is available to accommodate the desired scaling. We claim that the maximum permissible scaling is limited by how efficiently heat can be moved away from each individual core to the heat sink, and is

not significantly lowered by our scheme. Thermal resistance is directly proportional to thickness/separation and inversely proportional to cross-sectional area [199]. Typical chip thicknesses are < 1 mm and thermal conductivity of copper is higher than silicon [199]. Thus, lateral thermal resistance between cores is expected to be much higher than vertical thermal resistance. As long as on-chip/system TDP is not exceeded, which is guaranteed when predictions are accurate, any system that supports overscaling of voltage/frequency of cores should be able to benefit from LLC power budgeting for performance. Although we assumed a maximum frequency scaling of 25%, the maximum actual scaling was 16.4%. A lower scaling limit would reduce speedups and is similar to having a lower power margin.

Limitations of the reuse-based cache performance estimation framework, discussed in Section 4.8 of Chapter 4, continue to be applicable to this study.

5.8 Conclusion

As technology scales, intelligently budgeting power between system components will become increasingly important to obtaining optimum power-performance. In this chapter, we focused on maximizing performance of a chip multiprocessor (CMP) system for a given power budget (SLApower), by developing techniques to budget power between processor cores and caches. While this governor only targeted SLApower, it can also be easily retargeted for SLAee or SLAperf.

Dynamic cache configuration can reduce cache capacity, thereby freeing up chip power, but may increase the miss rate (and potentially memory power). Dynamic voltage and frequency scaling (DVFS) can exploit the saved power to increase core performance, potentially increasing system performance. We demonstrated that favorable configurations can be selected using simple analytic models, driven by hardware performance counters

to estimate the cache reuse distribution (also see Chapter 4).

Unavailability of this reconfiguration knob in real systems made it necessary to evaluate this space using full-system simulation. Detailed simulation models show that carefully budgeting power between cores, memory, and caches can improve system performance 0.5%–15.2% for 15 of 32 workloads and save an average of 4.4% total energy (wall power) averaged over all 32 workloads.

6 RELATED WORK

6.1 Overview

In this chapter we briefly discuss related work in characterization and governance of reconfigurable computers. These include:

- Characterizations of energy efficiency (Section 6.2). We describe our work in this area in Chapter 2.
- Descriptors for system power-performance states (Section 6.3). We propose using Π-states and the Π-dashboard ([192], Chapter 2).
- Power-performance goals targeted by governors (Section 6.4). We discuss the SLAs that we target in Chapter 3.
- Analytical models for cache performance (Section 6.5). We develop new models ([191], Chapter 4) that are based on cache reuse distances.
- Examples of system reconfiguration studies and reconfigurable knobs (Section 6.6). We describe the knobs that we study and their governance mechanisms in Chapters 3 and 5.

Finally, in Section 6.6.1 we propose a new classification system for governance studies that is based on the behavioral semantics of the reconfiguration capabilities instead of on the components that are reconfigured. Semantics-based classification enjoys the advantages of being more compact and perhaps more insightful than the other classifications.

6.2 Energy Efficiency Characterization

Energy proportionality has been extensively studied [22, 106, 122, 141, 184, 213, 231, 232]. Barroso and Hölzle [22] introduced the concept and argued that energy proportionality should be one of the main design goals. They compute power efficiency as $\frac{\text{Utilization}}{\text{Power}}$. We use the same definition for efficiency, but relabel utilization as percentage of peak performance.

David Lo et al. [141] proposed a relaxed model of energy-proportionality, called Dynamic Energy Proportionality, that ignores idle power. This corresponds to the Dynamic EP line in Figure 2.1. This linear model has also been studied in other prior works [223, 232]. Daniel Wong and Murali Annavaram [232] name the region that we call Sub-Linear as Superlinear. We prefer to use "Sub-" in the sense that operating in this region lowers efficiency compared to that of Linear (Dynamic EP).

A number of metrics for characterizing energy efficiency exist. Efficiency ($\frac{\text{Performance}}{\text{Power}}$) can be computed at individual loads [22], or as $\frac{\text{Total Performance}}{\text{Total Power}}$ over all loads [205]. Metrics based on the dynamic power range compute the ratio between the idle and peak power consumptions [223]. Other metrics consider the deviation of the power curve from an ideal curve, e.g., maximum relative power difference with respect to Dynamic EP [223], area enclosed by the power curve relative to that by Dynamic EP [232] or EP [184, 232], power used in excess to that by EP [232], etc. These metrics continue to be useful with the new ideals, EOP and Dynamic EO, replacing the conventional ideals.

Wong and Annavaram [232] introduced the notion of the EP Wall that needs to be overcome. They proposed leveraging heterogeneity to improve energy efficiency at low utilization. Our work with prefetch control and cache power budgeting will further help to overcome the wall. Wong and Annavaram [231] also investigated techniques for quantifying energy proportionality of a cluster of servers.

We observe that both Pegasus [141] and Knightshift [232] report data appearing to show occasional incursions into super-proportional regions. Chung-Hsing Hsu and Stephen W. Poole [106] observed real machines doing better than the conventional "ideal" system that assumes linear proportionality. They proposed quadratic proportionality (Power(u) $\propto u^2$, where u is the load level) as the new ideal model. However, this makes ideal system efficiency load-dependent $\left(\frac{u}{\text{Power}(u)} = \frac{1}{u}\right)$, with higher efficiency at lower loads than at higher loads.

Our view is that the design ideal, EOP, will have maximum efficiency (η_{max}) independent of load and will consume power linearly proportional to load, as proposed in the original EP model, but the constant of proportionality is different: it is defined by the most efficient configuration instead of by the configuration achieving the maximum performance. The Pareto frontier (Dynamic EO) is the operational ideal for the system and its efficiency is load-dependent. The most efficient configurations lie at the intersection of the EOP and Dynamic EO curves.

Song et al. [204] proposed Iso-energy-efficiency (EE) as the energy ratio between sequential and parallel executions of a given application. Our CPUE, LUE and RUE metrics do not use specific execution modes (e.g., sequential/parallel, homogeneous/heterogeneous, speculative/non-speculative, cache-conscious/cache-oblivious, etc.) for reference, but compare system states to the Pareto frontier (Dynamic EO) or to EOP. The definitions of our metrics are oblivious to which configurations created the frontier.

The EE model focuses on maintaining equal efficiency as systems and applications scale up. In contrast, the EP and EOP models focus on maintaining equal efficiency under changing loads. So our metrics include load, along with the configuration, as a parameter for quantifying excess energy used. On the other hand, EE does not quantify its dependence on load.

Barroso and Hölzle [104] compute datacenter energy consumption as PUE $\times$ SPUE $\times$ energy to electronic components. While PUE [16] accounts for non-compute overheads in datacenter building infrastructure, SPUE (Server PUE) accounts for overheads, e.g., power supply losses, to computing energy. Our RUE and LUE metrics do not separate SPUE losses from computing energy but separate energy-wasting operating configurations and loads from optimal ones.

6.3 Power-Performance States

ACPI: The Advanced Configuration and Power Interface (ACPI) specification [102] is an open standard that allows devices (resources) to specify discrete operating states identified by alphanumeric names. For example, P0, P1, P2,... represent processor performance states. ACPI enumerations lack quantification of system-wide power-performance impacts by not accounting for inter-resource interactions or dynamic execution profiles. A static enumeration of possible states for individual knobs, as in the ACPI [102] approach, is insufficient because it does not quantify power-performance impacts at the system level or take into account correlated effects across different knobs (e.g., prefetching). So, it is difficult to answer questions such as: which system configuration performs the best for a given power budget? which system configuration has the minimum energy-delay (ED) or ED^2 product?

New P states: Eckert et al. [70] proposed new processor P-states and L2 cache P-states, but did not provide a framework for optimal system configuration selection. The new processor states save power by reconfiguring pipeline front-end structures and mechanisms, e.g., register and fetch buffer sizing, simplified speculation control, limited checkpoint state, etc. (Sharkey et al. [193] also explored different fetch throttling mechanisms that differed in local (per-core) vs global chip information and per-core vs

chip-wide settings.) New states for L2 include drowsy and power-gating states. Dirty data from power-gated L2 ways is written to the L3 cache.

Π-states: Sen and Wood [192] proposed Π-states that overcome the limitations of ACPI state enumerations. This is motivated by the observation that individually ordered lists of operating states for different resources, as in ACPI, do not identify ordering for combinations of states across resources, required for system-level coordinated management. ACPI enumerations lack quantification of system-wide power-performance impacts by not accounting for inter-resource interactions or dynamic execution profiles.

Each Π-state is a 4-tuple (slowdown, dynamic energy, static power, work) that describes the effect of using a configuration of a system component. A centralized coordinator stitches these descriptors together to determine system-wide impacts if multiple components are reconfigured. The system computes a Π-dashboard consisting of Pareto-optimal Π-states that are numbered in decreasing order of performance. The user or operating system selects a desired Π-state, that corresponds to the optimum value of a metric (e.g., minimum energy, minimum EDP, etc.), causing the system to transition to the corresponding configuration.

6.4 Optimization Goals

There exists a variety of flavors of the power/energy management problem. The *power-budgeting* problem seeks to partition a maximum power budget among resources to maximize performance [115]; the *energy-minimization* problem seeks to find configurations that minimize energy consumption (equivalently, maximizes performance/watt); the *min-EDP* problem seeks to minimize the energy-delay product (EDP) [82] so that configurations that reduce energy but cause unacceptable delays are not chosen. Snowdon et al. [202] generalized this metric to include non-integral exponents for power and delay. Our

two-level governors, described in Chapter 3, can be easily retargeted to optimize system operations for these metrics.

A number of works have studied power/energy management while meeting tail latency and response time deadlines [122, 141, 152]. Targeting this SLA may require that workload-specific semantic knowledge be available to the objective selector. Our current governors do not have high-level knowledge about the workloads and we do not target this SLA in this work.

6.5 Cache Models

The goal of these models is to predict cache performance (miss ratio) as a function of cache organization and workload properties. Here we briefly describe various approaches to solve this problem.

Power-Law models: Chow [50], Hartstein et al. [100], and others used power laws based on cache capacity to predict miss ratios. One instance of such a power law predicts that the miss ratio reduces by $\sqrt{2}$ if the cache capacity doubles, and is popularly known as the 2-to-$\sqrt{2}$ rule. These models have practically zero overhead but may have large errors since they do not account for working-set sizes and cache access patterns.

Unique and absolute reuse distance models: Mattson [150] introduced the concept of predicting miss ratios from (unique) stack distances for caches that use replacement policies having the inclusion property. This technique has been subsequently used in many works [29, 55, 56, 65, 103, 140, 191, 196, 200, 242]. Guo and Solihin [91] proposed circular sequence profiles that are similar to stack distances in reuse intervals.

Stack distance distributions can be determined offline or online. In offline algorithms, stack distances are computed offline from an address trace. Previous work has extensively studied cache miss rate prediction using offline estimation of LRU stack/reuse

distances [12, 29, 103, 196, 200], but have limited applicability for online use.

Computation time to determine the distributions can be reduced with efficient algorithms [13] or by approximate analysis [242]. Hill and Smith [103] introduced techniques for estimating miss ratios for many different cache organizations from a single pass over an address trace. Shi et al. [196] perform single-pass stack simulation to project cache performance and to study the impact of data replication for various L2 cache configurations. Online determination of the stack distance distribution cannot directly apply techniques from offline methods due to constraints on computational state and complexity.

Tam et al. [220] use hardware mechanisms for address sampling and post-processing software for computing stack distance distributions. Since distribution estimation and hit ratio computation is offline, it cannot react to workload changes in real time.

Online methods have severe restrictions on space and time complexity, but must achieve good accuracy. Way counters [127, 175, 215] exploit the LRU stack property to predict miss rates for configurations smaller than the current cache. Shadow tags [172] (or Auxiliary Tag Directories [175]) extend way counters to predict configurations with higher associativity than the active cache configuration. Dynamic set-sampling can reduce overheads [174]. Way counters have been extended to work with PLRU replacement [127] using a heuristic that estimates the LRU stack depth using the PLRU tree bits. Suh et al. [214], Qureshi et al. [175] proposed mechanisms for partitioning of shared caches (L2) among competing processes using way-counters. Suh et al. [215] also proposed set-counters in LRU order, with each counter tracking accesses to a group of sets. We discuss a limitation of set counters in Chapter 4, Section 4.6.2. Gordon-Ross et al. [84] used a hardware TCAM to track stack distances for LRU miss-ratio predictions. However, area overheads probably limit this approach to small caches. In contrast, our methods

([191], Chapter 4) have very low area and power overheads, so they can used for large caches. We predict miss ratios for caches that are configurable in both the number of sets as well as ways.

StatCache [28] and StatStack [71] used sampling on the cache access stream to estimate the absolute distance distributions, then use that to estimate miss ratios for fully-associative LRU or RANDOM caches. Estimating absolute distances is easier than estimating unique distances. We also use absolute distances for RANDOM cache estimations, but estimate average absolute distances from unique distances. We use unique distances for LRU caches. Pan and Jonsson [167] use absolute distance distributions and Markov models to estimate performance of set-associative caches. Inter-Reference Gaps in IRG models [169, 219] are basically absolute reuse distances.

Binomial and Poisson models: Smith [200] and Hill [103] introduced the technique of using Binomial distributions along with stack distances to model set-associative LRU caches. We use the same approach in our work ([191], Chapter 4) but can handle some implementable, non-LRU, policies as well. Agarwal et al. [3] also use Binomial models, in conjunction with other models, e.g., Markov models, for cache performance estimations. Their models also account for block sizes, degree of multiprogramming, and task switching intervals. Stone and Thiebaut [211], Falsafi and Wood [75] use binomial probability models to model cache reload transients due to context switches based on the footprints of the competing programs and cache size. The Binomial model assumes independent mapping of addresses to cache sets. Pan and Jonsson [167] improved prediction accuracy by considering the actual mapping achieved.

Cypher [55, 56] proposed using Poisson distributions to estimate cache miss ratios from estimated stack distances. We use a similar approximation in Chapter 4 to reduce the computational costs associated with using the Binomial model. Cypher also used

filter fraction metrics that reduce the effective distance to be tracked. Computing filter fractions can be expensive.

Markov models: These models consider the evolution of cache states with transition probabilities between states [90, 91, 167] but can be computationally expensive. Guo and Solihin [91] proposed Replacement Probability Functions (RPFs) that describe the probability that a line at a certain stack position will be replaced given that a miss happens to that cache set. (Our cache hit function, ϕ, describes the probability that a line at a certain stack position will hit given that an access happens to that line.) Guo and Solihin's Markov model tracks the evolution of states described by reuse intervals and stack positions. Grund and Reineke [90] proposed replacement policy tables that improve upon Guo and Solihin's approach by decoupling policy characterization from workload properties. Agarwal et al. [3] used Markov models to characterize spatial locality. Others have used Markov models to analyze the behavior of context switch misses [138].

Closed-form models: These models express cache performance in terms of easily-computable, non-recursive expressions having a small number of terms. The power-law models are examples of closed-form models (parametrized with an initial measurement). Cache Calculus [24] developed a system of differential equations, solving which produces closed-form expressions of (fully-associative) cache performance in terms of cache size and high-level data structures, e.g., array sizes in array access workloads. One challenge is to be able to formulate the cache access stream in terms of high-level data structures.

Worst-case models: Reineke and Grund [180] prove relations on best and worst-case bounds of cache performance for several replacement policies. Our work, in contrast, studies average case behavior.

A number of prior works [14, 89, 179] have explored applying static analysis techniques, such as abstract interpretation, to determine bounds on cache performance for real-time

systems. Lv et al. [144] provide a survey of the area.

Xiang et al. [235] developed a higher-order theory of locality (HOTL) that interrelates a number of locality metrics—reuse distance, fill time, footprint, miss ratio, and time between cache misses.

6.6 Reconfiguration Knobs

In this section we briefly discuss some microarchitectural and runtime knobs for power-performance management where a dynamic choice can be made on whether or not to execute using a certain system/component configuration or on the extent/degree of such reconfiguration. Our goal is to present different kinds of reconfigurable architectures/capabilities, not necessarily to catalog all prior studies within each class.

Core DVFS: Dynamic voltage and frequency scaling (DVFS) and dynamic frequency scaling (DFS) for cores are well-known techniques [86, 110, 111, 229].

Isci et al. [115] introduced the concept of maximizing performance for a given power budget, using a global power manager and per-core monitors to set per-core DVFS modes. Their MaxBIPS policy predicts the power and performance for each possible configuration. This can limit scalability to larger number of cores. A global manager selects the configuration predicted to have the highest throughput within the power budget.

Ma et al. [145] explored the problem of selecting different per-core DVFS levels for systems that have the capability. Their mechanism first uses feedback control to determines an aggregate frequency that does not exceed the specified power budget. Next, this is partitioned among groups of cores where all cores within the same group run threads of the same application. Finally, cores within the group are allocated their frequency settings based on thread criticality.

Koala [202] developed a power-performance management framework in the OS. It uses DVFS to manage tradeoffs between performance and energy consumption. Koala uses performance counters to characterize an offline model to predict energy for all frequency settings. It then uses this model online to select the best setting. Koala proposed a new metric called generalized energy-delay ($P^{1-\alpha}T^{1+\alpha}$, $\alpha \in [-1, 1]$) that extends the ED metric by allowing non-integral exponents for both power (P) and delay (T).

Spiliopoulos et al. [207] proposed green governors that improve energy efficiency by controlling core frequency and voltage settings. The governors use performance counters (to obtain IPC and stall counts) and measurements of idle static power to predict the performance and energy consumption for different frequency settings. The governors choose the best frequency that optimizes a given metric, e.g., minimum EDP, minimum ED^2P, minimum EDP with a performance constraints. Our frequency governors, described in Chapter 3, use a profiling-based approach that interpolates power and performance from a few measurements. Our prefetching governor profiles prefetching modes and selects the best-performing mode. Our governor for SPECpower does not predict power/energy for any configuration. Its goal is to reduce the number of idle cycles while maintaining the current performance. It only profiles for prefetch modes but not frequencies. A profiling-based approach removes the need for building accurate performance and power models from performance counters, but is only possible for knobs that allow fast reconfiguration, e.g., processor frequency or prefetching. Our governor for cache sizing, described in Chapter 5, builds a performance and power model from performance counters.

Rubik [125] uses per-core DVFS to reduce power while still meeting latency constraints of work requests. It maintains two tables that describe distributions of per-request core and memory completion times in cycles. These tables are updated periodically (every

100ms). Every time a request completes or a new request is admitted, the tables are consulted and a frequency is calculated that meets completion time constraints for all current requests. Rubik uses a power model based on performance counters to guide its decisions. A proportional-integral (PI) controller runs over a longer interval (1 sec) to compare predicted with measured latencies and calculate adjustments. Rubik also studies consolidating batch and latency-critical applications together to reduce total idle power of servers.

Prekas at al. [171] propose mechanisms that control per-core DVFS and the number of logical cores (physical cores and hyperthreads) allocated to applications running on the IX [26] operating system. Their controller keeps track of network queueing delays to help in resource management and can migrate applications and network processing between cores. They propose two schemes, one for reducing computing energy and the other for reducing idle power through consolidation. In both schemes, latency constraints are maintained.

Rangan et al. [177] proposed maintaining different homogeneous cores at different voltage/frequency levels. Depending on its runtime characteristics, a workload can be rapidly migrated from one core to another so that it can be executed at a different frequency. This scheme aims to reduce DVFS transition times from microseconds, with voltage regulators, to nanoseconds by inter-core migration.

A number of prior works [141, 182, 213] have explored the use of RAPL [113] in specifying power caps. Rountree et al. [182] used RAPL to reduce power for HPC (High Performance Computing) applications and observed that power limits can transform power variations across machines into performance variations. Lo et al. [141] used RAPL to improve energy efficiency of OLDI (online data intensive) workloads. As we have discussed in Section 3.9, current RAPL implementations do not deal with reconfiguration

knobs such as cache prefetching.

Memory DFS/DVFS: Deng et al. (MemScale) [62] and David et al. [59] proposed using DFS/DVFS for main memory.

Core DVFS and memory DVFS: Subramaniam and Feng [213] explored using RAPL for both cores and DRAM to improve energy efficiency of enterprise workloads and found that limiting power for the core domain provides the most benefits compared to other domains.

CoScale [61] demonstrated coordinated management of core DVFS and memory DVFS. It uses a gradient-descent heuristic that is related to the greedy solution strategy for the integer knapsack problem. The approach in CoScale is to arrange operating states in decreasing order of marginal utility ($\Delta power/\Delta performance$) and greedily pick states till a maximum slack is not violated. Although theoretically the greedy strategy can have up to a 2-approximation factor [58], CoScale demonstrates a tighter approximation in practice. Their default epoch length is 5ms.

Number of cores/threads and core DVFS: Li and Martinez [136] explored simultaneous management of the number of cores and DVFS levels for the chip (all cores at the same level). Their algorithm performs a binary search on the number of processors, starting from the middle number. For each number, it tries lowering the DVFS level till the performance target is just missed. This process is repeated on the upper or lower halves of the search space on the number of processors till no improvement is found. Heuristics are used to speed up the search for the appropriate DVFS level for each setting of the number of cores. Overall, the complexity is $O(\alpha log(N))$ for N cores and $\alpha << L$ for L DVFS levels.

Vega et al. [224] explored coordinated management of core DVFS and the number of enabled cores. This is driven by heuristics based on core utilizations. The utilization is

averaged over a number of intervals, with the best choice of history length being 3. They consider a time interval of 1 second for reconfiguration decisions.

Curtis-Maury et al. [54] studied varying the number of threads, mapping of threads to cores, and core DVFS levels. This involves sampling the execution with a few configurations before a reconfiguration decision is made. The applications are instrumented around OpenMP parallel regions. Varuna [208] also changes parallelism dynamically, but does not need to make changes to application source code.

Cache size: Albonesi [10] introduced mechanisms for disabling cache ways to save energy. Dropsho et al. [67] proposed the accounting cache that uses way counters to evaluate configurations with different associativities. Yang et al. proposed cache reconfiguration in the number of sets [238] and in both sets and ways [237]. Kaxiras et al. [126] introduced cache decay that exploits generational behavior of cache line usage to reduce cache leakage with (area-expensive) power-gating control per line and a (small) performance hit. Instead of fully disabling cache lines and losing state, Flautner at al. [77] introduced the drowsy cache where cache lines can be put into a low-power state-saving mode for saving energy. Our work (Chapters 4 and 5, [190]) studies reconfiguring the cache for both associativity and the number of sets, but not enabling/disabling at the granularity of cache lines, and considers power gating the disabled regions for maximum power savings.

Intel processors/microarchitectures such as the Core Duo [163], and Ivy Bridge [181] dynamically size caches based on activity. In contrast, reuse-based predictors track locality and hence select a smaller cache for highly cache-active but cache-insensitive workloads whereas activity-based models would select a large cache. Some recent Intel processors allow core-wise monitoring and partitioning of the LLC capacity [53].

Yang et al. [237, 238] compare the number of misses to a bound/threshold to drive

reconfiguration decisions. Keramidas et al. [128] compared the number of misses and conflict misses to bounds/thresholds to decide reconfigurability in the number of sets and ways of LRU caches. Sundararajan et al. [216] considers both the LRU stack distance and number of dead sets to determine the configuration for the number of sets and associativity.

Gordon-Ross et al. [84] proposed a one-shot cache reconfiguration scheme. It uses a hardware TCAM to determine stack distances of addresses and tracks stack hit counts for various cache sizes, assuming stack inclusion. It (time-)samples the address stream to reduce the average processing time. The hardware TCAM results in a 12% area overhead for the ARM920T that has 16KB instruction and data caches. As we have discussed in Chapter 4, tracking stack distances for large multi-megabyte caches is prohibitively expensive.

Albericio et al. [8] proposed the reuse cache, an LLC organization where the data array is sized depending on the amount of data that is reused. The tag array is decoupled from the data array. On a miss in the tag array, data is fetched from memory but not placed in the LLC data array. On a hit to the tag array for a line that is not present in the data array (indicates reuse), that line is fetched from memory and placed in the data array. This scheme requires modifications to the coherence protocol, forward pointers in the tag array, reverse pointers in the data array, and uses a different replacement policy for the tag and data arrays. Traditional cache resizing, in the form that we have studied, does not need these changes but incurs overheads during resizing due to subsequent warmup.

Cache compression: Compression helps to store more data in a cache of a given size compared to storing in uncompressed form. Compression may reduce misses but makes hits more costly due to the decompression latency. Alameldeen and Wood [5] proposed

an adaptive compression scheme that uses stack distance information to decide whether to allocate cache lines in compressed or uncompressed form. Further benefits can be derived by combining compression with adaptive prefetching [7].

Cache size and compression: Hajimiri et al. [97] studied cache size reconfiguration along with code compression.

Cache partitions: The fraction of shared cache space available to each core or application can also be dynamically controlled. Cache partitioning [49, 178] can be application-utility-aware [175, 214], LLC-bank-aware [124], MLP-aware [159], spatial-locality-aware [93], cooperative [217] etc. The partitions may be coarse-grained [49, 178, 217] or fine-grained [146, 186].

Cache hierarchy: Albonesi [9] studied simultaneous reconfiguration of L1 and L2 cache sizes in an exclusive hierarchy. Balasubramonian et al. [20] studied reconfigurable L1, L2, and TLBs for energy-efficient performance. A single large cache organization serves as a configurable 2-level non-inclusive hierarchy. The optimal cache configuration is selected by exploration—successive cache sizes are chosen till the miss rate is sufficiently small. The reconfiguration intervals for the cache and TLB are 100K cycles and 1M cycles respectively.

Cache content replication: Chang and Sohi [45], Beckmann et al. [23] explored replicating cache blocks from a remote LLC bank in the private or local LLC bank. The advantage is that hits to local banks are faster than those to remote banks. However, replicating blocks reduces effective capacity of the cache that in turn can potentially increase the miss rate. The degree of replication can be varied based on cost-benefit tradeoffs.

Cache insertion/promotion/eviction: Caches are of finite size and hence must eventually evict some existing data to make room for new data when needed. A number

of works have proposed new management policies that decide whether or not to insert new data [79, 94, 165], where (in terms of recency) to insert new data [79, 118, 119, 120, 123, 173, 234], and how to promote [119, 120, 131, 135, 236] or protect [69] or evict [169, 176, 185, 219] data.

Access granularity:Veidenbaum et al. [225], Yoon et al. [239] proposed dynamically reconfiguring the width/granularity for cache and memory accesses.

Cache size and core DVFS: Meng et al. [155] explore cache resizing (in the number of ways) and changing core DVFS levels. They use way counters and analytic power-performance models for power management. In contrast to MaxBIPS, they use a greedy search strategy to decide the final configuration.

Their work has several limitations as follows. First, they assume true LRU replacement, which is impractical to implement for highly associative last-level caches. This is critical, because practical implementations such as PLRU do not have the stack property and thus a single tag array cannot both provide replacement decisions and miss-rate predictions for larger sized caches. While Meng et al.'s work could probably be extended to use shadow tags and dynamic set sampling [172, 175], the extra area and power would far exceed that of our reuse sampling approach. Second, their study evaluates 600µsecs observation intervals, which are far to short to amortize the reconfiguration overhead of large, e.g., 32MB LLCs. Finally, their approach only handles limited cache reconfigurability (number of ways, not sets), does not consider thermal effects on leakage power, and optimizes for the lowest-power configuration, not the highest-performance configuration.

Cache size, memory bandwidth, and core DVFS: Bitirgen et al. [31] used a machine learning approach for resource management. They construct a per-application artificial neural network (ANN) that takes as input the power budget, bandwidth, cache size, and various performance counter values to predict performance as output. These ANNs are

queried by a global resource manager that uses a stochastic hill-climbing algorithm to select a configuration predicted to achieve the highest performance.

Chip-wide DVFS and thread packing: Cochran et al. [51] explored simultaneous management of DVFS levels and packing of threads on to a subset of cores. They use an offline characterization process to analyze performance counters and power caps to create a lookup table. This is queried at run time for all potential configurations to determine the optimal setting. The online lookup uses performance counter values including information from thermal sensors, but does not require power information. The control activation period is in the order of seconds.

Core organization: Reconfiguring various components in both the frontend and backend of cores has been well studied. Albonesi [9] proposed Complexity-Adaptive Processors (CAP) with reconfigurable resources (instruction queue size, L1 and L2 cache sizes). Manne et al. [147] proposed mechanisms that reduce energy consumption by adapting control speculation to prevent potentially wrong-path instructions from being dispatched. Ghiasi et al. [80] proposed switching between in-order and out-of-order executions. Bahar and Manne [18] explored changing the issue width and the number of enabled functional units. Buyuktosunoglu et al. [39, 40] studied issue queue reconfiguration and fetch gating. Forwardflow [81] dynamically tracks dataflow dependencies in the instruction stream and uses a dataflow queue whose capacity can be dynamically reconfigured, thereby changing the instruction window size. The dataflow queue is organized into banks that can be independently activated/deactivated by system software.

Core throttling and memory throttling: Felter et al. [76] demonstrated performance benefits through power-shifting between the core and memory within a power budget. They achieve this by throttling core and memory operations depending on workload

characteristics. Core throttling is done at the instruction dispatch unit. Memory throttling is done by limiting the number of memory request per unit time (allowed bandwidth). Their throttling mechanism affects shifting of active power. They studied interval sizes in the range of 5μ sec to 1ms.

Core organization and number of cores: Ipek et al. proposed core fusion [114] where resources of independent cores are dynamically grouped/fused together to form a larger core resulting in asymmetric CMPs. WiDGET [228] dynamically changes the number of instruction engines and the number of execution units allocated to each engine. Instructions are steered from the instruction engines to the execution units dynamically allocated to it.

Cache size and core organization: Albonesi et al. [11] explored adaption of the L1 cache sizes and core resources. Dropsho et al. [68] also studied dynamic reconfiguration of these resources in GALS (Globally Asynchronous, Locally Synchronous) microprocessors. The L1I, L1D, and L2 caches have reconfigurable associativities. The branch predictor has a configurable history length. The L1D and L2 caches are sized together. Similarly, the L1I cache and branch predictor are sized together. The sizes of the integer and floating point issue queues are configured depending on the ILP that is available.

Core organization and DVFS: Sasanka et al. [189] explored changing the DVFS level along with core instruction window size, issue width, and the number of functional units. This study aims to reduce energy consumption for multimedia applications while still meeting deadlines. The DVFS decision seeks to eliminate idle time between processing different frames whereas reconfiguring the other resources aims to reduce processing energy within each frame without affecting execution time.

Core organization and memory idle states: Li et al. [137] studied reconfiguration of both core resources (instruction window size, issue width, number of functional units)

and memory idle states (standby, nap, powerdown). Each application phase is profiled, with the number of profiling intervals per phase being equal to the number of processor configurations. This overhead is amortized over multiple occurrences of each phase for long running applications. The decision algorithm aims to optimally distribute the target slack between the core and memory.

Dynamic task reassignment: Chakraborty et al. [43, 44] proposed computation spreading in over-provisioned multicore systems (OPMS). This uses a light-weight virtual machine monitor (VMM) to assign similar computation fragments (parts of software threads) to cores. For example, OS code is executed on a different core than user code. The idea is to improve energy efficiency through dynamic specialization (e.g., predictive structures such as branch predictors, etc., can work better if similar code is executed on the same core) as well as manage peak power consumption by limiting the number of simultaneously active cores in OPMS.

Prefetching: Data prefetching is a well-known speculative technique [17, 41, 74, 209] for improving performance. However, inaccurate prefetching will use more energy either due to unneeded or inadequate fetches or due to lack of timeliness. Both the number of blocks prefetched [57] as well as the lookahead distance [85] may be dynamically configured. Gornish and Veidenbaum [85] combined hardware prefetching with software (compiler-directed) prefetching. Guo et al. [92] developed a power-aware prefetch engine that uses analysis information from the compiler to decide on the prefetch mechanism.

Prefetching, DVFS: In Chapter 3, we demonstrate governors that improve energy efficiency by simultaneously controlling DVFS levels and enabling/disabling of L2 cache prefetching.

Prefetching, DVFS, and number of cores/threads: Kamruzzaman et al. [121] proposed using helper threads to prefetch data needed by the main computation. A number

of helper threads can be used, each of which can prefetch a different chunk of data. Once the data is fetched, the compute thread is migrated to that core whereas the helper thread is shifted elsewhere. The helper threads can run at lower frequencies whereas the compute thread can run at higher frequencies.

Frequency control and sleep states: SleepScale [139] investigated coordinated management of frequency levels and sleep state selection. Running at higher frequencies consumes more power, but finishes tasks earlier leaving more time to enter and remain in deep sleep states. The best policy is determined by factors such as job size and desired average response time. They logically partition the execution into epochs of 5mins. Once every epoch, the best policy is determined based on the estimated interarrival times, response times, and utilization. Considering each policy requires ~6ms for a total of < 1s to consider all policies.

Networks: Abts et al. [2] proposed making datacenter networks consume energy in proportion to their traffic by using a flattened butterfly topology and dynamically changing the frequency of the communication links. Changing the link frequency changes both its data rate and its power consumption. RAFT [157] dynamically varies the frequencies and number of virtual channels in routers. PowerNetS [241] explores joint optimization of workload consolidation and network traffic routing to minimize total power.

Kim et al. [130] use spatial speculation to reduce the number of flits/bit-flips transmitted thereby reducing interconnect energy. On a miss, L1 caches fetch block-sized data from the next level. However, not all words within the block may be used in future and hence need not be fetched. It uses a predictor to determine which parts of the requested block is likely to be reduced. A misprediction causing a required word not to be fetched would cause it to be fetched when the word is requested resulting in latency overheads.

Accelerators: Being specialized, accelerators (including GPGPUs) are more energy-efficient than general purpose processing elements. The DySER [87] accelerator uses specialized circuit switching between computation units to eliminate instruction execution overheads. Venkatesh et al. [226] proposed specialized processors called conservation cores (c-cores) to reduce energy and energy-delay of hot code paths. The CPU and c-cores communicate through scan chains. KnightShift [232] uses a heterogeneous server architecture that adds a low power compute node along with the primary server. DreamWeaver [154] proposed using a co-processor called the Dream processor, that monitors and suspends incoming work requests along with a Weave scheduler that aligns and increases idle times.

6.6.1 Classification

We will now present a new classification system for power-performance management studies by the semantics of reconfiguration capabilities. Our classification system has five main semantic types, each of which have several subtypes with a total of twelve subtypes.

1. **Computation**:

    - *Organization*: Number and organization of structures, such as functional units, instruction windows, issue queues, pipelines, etc. that make up or control how processing elements (CPU cores, network routers, etc.) do computations. Examples: [9] [18] [40] [39] [81] [114] [228] [11] [68] [189] [137] [157].
    - *Speed*: Time to process and transform information. Core DVFS and RAPL studies are included in this class. We also include network router DVFS in this class, but place memory controller DVFS in the Storage class (see below).

Examples: [229] [86] [115] [145] [202] [207] [125] [171] [177] [182] [213] [141] [61] [192] [136] [224] [54] [190] §5 [155] [31] [51] [189] §3 [121] [139] [157].

- *Concurrency*: Number of processing elements or threads to do a computation. This includes core parking, hyperthreading, task creation/stealing, and workload consolidation studies.

  Examples: [171] [136] [224] [54] [208] [51] [114] [228] [44] [43] [241].

2. **Communication**:

   - *Latency*: Time to transmit a bit of information over an interconnect. This includes frequency scaling of memory and network links.

     Examples: [213] [62] [59] [61] [2].

   - *Bandwidth*: Number of bits of information that are moved per unit time. This includes fetch bandwidth and memory bandwidth throttling.

     Examples: [213] [62] [59] [61] [192] [31] [76] [2].

3. **Storage**:

   - *Size*: Number of storage cells available to applications. Not all cells may be available—they can be shut down or allocated to other applications. This includes cache power-gating and partitioning studies.

     Examples: [192] [10] [67] [238] [237] [126] [77] [190] §5 [128] [216] [84] [8] [97] [49] [178] [214] [175] [124] [159] [93] [217] [186] [146] [9] [20] [155] [31] [11] [68].

   - *Content*: Information stored in allocated storage cells. This includes cache replication, replacement, bypass, and compression studies.

     Examples: [5] [97] [45] [23] [79] [94] [165] [123] [173] [118] [119] [120] [131] [236] [69] [176] [185] [135] [234] [169] [219] [7].

- *Latency*: Time to read or modify storage cells. This include memory DVFS/RAPL studies.

  Examples: [213] [62] [59] [61] [192].

4. **Scheduling**:

   - *Spatial*: Particular processing elements on which to do the computation. This includes scheduling work on specific cores or accelerators.

     Examples: [177] [54] [80] [44] [43] [121] [87] [226] [232].

   - *Temporal*: When to do the computation. It may be scheduled later for coordinated management with sleep states.

     Examples: [137] [154] [139].

5. **Speculation**:

   - *Control*: This includes adapting structures that affect branch prediction and dealing with the predicted results.

     Examples: [147] [68].

   - *Data*: Amount of extra/less data to access than requested. This includes prefetching and access granularity studies.

     Examples: [225] [239] [17] [57] [85] [92] §3 [121] [130] [7].

Tables 6.1–6.3 summarize how the studies can be classified according to this system. The Count column in the tables show tuples of the form (n_1, n_2) where n_1 is the number of types and n_2 is the number of subtypes considered by the corresponding study. In these examples, at most three of the five possible types and at most four of twelve possible subtypes have been considered in any single study. This suggests the existence of potentially unexplored combinations. For example, simultaneous adaptivity

of communication and scheduling (such as, how can Varuna [208] coordinate with interconnect frequency scaling), communication and speculation (such as, how can MemScale[62] coordinate with adaptive prefetching), or storage and scheduling (such as, how can ICP [121] coordinate with cache resizing) does not seem to have been well explored.

Refs.	Computation			Communication		Storage			Scheduling		Speculation		Count
	Org.	*Speed*	*Conc.*	*Latency*	*BW*	*Size*	*Content*	*Latency*	*Spat.*	*Temp.*	*Control*	*Data*	
[18]	✓												1,1
[39]	✓												1,1
[40]	✓												1,1
[81]	✓												1,1
[141]		✓											1,1
[115]		✓											1,1
[86]		✓											1,1
[229]		✓											1,1
[125]		✓											1,1
[145]		✓											1,1
[182]		✓											1,1
[202]		✓											1,1
[207]		✓											1,1
[189]	✓	✓											1,2
[157]	✓	✓											1,2
[241]			✓										1,1
[208]			✓										1,1
[228]	✓		✓										1,2
[114]	✓		✓										1,2
[171]		✓	✓										1,2
[136]		✓	✓										1,2
[224]		✓	✓										1,2
[51]		✓	✓										1,2
[76]					✓								1,1
[2]				✓	✓								1,2
[67]						✓							1,1
[10]						✓							1,1
[8]						✓							1,1
[178]						✓							1,1
[216]						✓							1,1
[159]						✓							1,1
[217]						✓							1,1
[20]						✓							1,1
[84]						✓							1,1
[77]						✓							1,1

Table 6.1: Classification, by semantic types, of system reconfiguration capabilities.

Refs.	Computation			Communication		Storage			Scheduling		Speculation		Count
	Org.	*Speed*	*Conc.*	*Latency*	*BW*	*Size*	*Content*	*Latency*	*Spat.*	*Temp.*	*Control*	*Data*	
[128]						✓							1,1
[93]						✓							1,1
[126]						✓							1,1
[238]						✓							1,1
[186]						✓							1,1
[237]						✓							1,1
[214]						✓							1,1
[124]						✓							1,1
[49]						✓							1,1
[175]						✓							1,1
[146]						✓							1,1
[9]	✓					✓							2,2
[11]	✓					✓							2,2
[190]		✓				✓							2,2
§5		✓				✓							2,2
[155]		✓				✓							2,2
[31]		✓			✓	✓							3,3
[5]							✓						1,1
[169]							✓						1,1
[120]							✓						1,1
[236]							✓						1,1
[135]							✓						1,1
[79]							✓						1,1
[165]							✓						1,1
[219]							✓						1,1
[69]							✓						1,1
[173]							✓						1,1
[185]							✓						1,1
[131]							✓						1,1
[234]							✓						1,1
[119]							✓						1,1
[176]							✓						1,1
[94]							✓						1,1
[118]							✓						1,1
[45]							✓						1,1

Table 6.2: Classification (cont.), by semantic types, of system reconfiguration capabilities.

Refs.	Computation			Communication		Storage			Scheduling		Speculation		Count
	Org.	*Speed*	*Conc.*	*Latency*	*BW*	*Size*	*Content*	*Latency*	*Spat.*	*Temp.*	*Control*	*Data*	
[123]							✓						1,1
[23]							✓						1,1
[97]						✓	✓						1,2
[62]				✓	✓			✓					2,3
[59]				✓	✓			✓					2,3
[213]		✓		✓	✓			✓					3,4
[61]		✓		✓	✓			✓					3,4
[192]		✓			✓	✓		✓					3,4
[80]									✓				1,1
[232]									✓				1,1
[87]									✓				1,1
[226]									✓				1,1
[177]		✓							✓				2,2
[43]			✓						✓				2,2
[44]			✓						✓				2,2
[54]		✓	✓						✓				2,3
[154]										✓			1,1
[137]	✓									✓			2,2
[139]		✓								✓			2,2
[147]											✓		1,1
[68]	✓					✓					✓		3,3
[85]												✓	1,1
[225]												✓	1,1
[239]												✓	1,1
[57]												✓	1,1
[92]												✓	1,1
[130]												✓	1,1
[17]												✓	1,1
§3		✓										✓	2,2
[7]							✓					✓	2,2
[121]		✓							✓			✓	3,3

Table 6.3: Classification (cont.), by semantic types, of system reconfiguration capabilities.

7 CONCLUSION

Power and energy consumption are first-class constraints today for computer system design and operations. One of the reasons for power (and energy) waste in computers is that they continue to consume power (and energy) even when they are idle. Barroso and Hölzle [22] proposed Energy Proportionality (EP) as a model for ideal system behavior. In this model, the ideal system should use power in proportion to utilization (performance or load served). The EP model has been very influential in encouraging system designers to improve energy efficiency of their systems.

The EP model for ideal behavior holds true for systems with fixed resources. However, modern computers have reconfigurable resources. Such reconfigurable systems may also exhibit super-proportional behavior. The advent of such super-proportional systems is a game changer. In these systems, one can get more performance for the power consumed (or equivalently, more work done for the energy consumed) at intermediate loads than what one could get in an "ideal" EP system. EP no longer characterizes the maximum energy efficiency that super-proportional systems can attain. In fact, aiming for EP behavior can be significantly suboptimal except at very low loads. In Chapter 2 we proposed a new ideal model, EOP (Energy Optimal Proportional), that characterizes maximum energy efficiency for all systems, both super-proportional and otherwise.

EOP is an ideal model for system designers. It describes a lower bound on the energy consumption (upper bound on the energy efficiency) that the given system can achieve. Any shortfall from this maximum efficiency at any operating load indicates potential for further improvement. System designers can use this characterization to both improve the upper bound as well as reduce shortfall, that is, make the Pareto frontier (Dynamic EO) closer to EOP, throughout the operating range.

For a given machine, system operators should aim to constrain the system to operate at or close to the Pareto frontier (Dynamic EO). They should use operating policies, or governors, that manage the system to satisfy different SLAs—to maximize energy efficiency (SLAee), to maximize performance within a given power budget (SLApower), to maximize power savings while meeting a given performance constraint (SLAperf), etc. We develop new governors that configure processor frequency and cache prefetching (Chapter 3) or cache capacity (Chapter 5).

We use a profiling-based approach to decide prefetch settings (enable/disable). To predict cache performance, we first sample the cache access stream and use a Bloom filter to determine the reuse distance distribution. The, our analytical models, described in Chapter 4, read the estimated reuse distribution to predict cache performance for other cache configurations that differ in the number of sets or ways. Our method is decoupled from the current cache configuration. The hardware requirement of our approach is less than that of shadow tags+way counters for large caches and is also more flexible in that it can also estimate performance for target cache configurations having different numbers of sets in addition to those with different associativities.

Being able to dynamically reconfigure (last-level) cache capacity opens up the possibility of "lowering" the Pareto frontier resulting in more energy-efficient system operation. In Chapter 5 we developed a governor that simultaneously configures processor frequency and cache capacity. Our experiments in that chapter only considered SLApower, but it can be easily extended to target the other SLAs as well. We showed that our analytical models that drive reconfiguration decisions work quite well compared to an idealistic/oracular approach.

In datacenters, energy is wasted by both the cooling and the IT (compute, networking, etc.) infrastructures. The PUE (Power Usage Effectiveness) metric [16] tracks excess

energy used by the cooling infrastructure. Through intense focus on the problem of reducing energy waste in the cooling infrastructures, modern datacenters have succeeded in having very low PUE values. This in turn makes energy waste in the IT infrastructure, particular servers, a major contributor to overall energy waste in modern datacenters. In Chapter 2 we proposed a new metric, Computational PUE (CPUE), that tracks excess energy used by the servers. Our new Iron Law of Energy decomposes CPUE into three factors—LUE (Load Usage Effectiveness), RUE (Resource Usage Effectiveness), and E_{min}. LUE is affected by inter-server load management decisions and demand-driven load fluctuations. RUE is affected by intra-server resource configurations. E_{min} is affected by system design choices. Making systems more energy efficient will require focused effort on improving each of these aspects. We hope that the Iron Law of Energy will help to drive and focus this effort.

Our work has the following limitations.

- **Single-server systems**: Our experiments in this work used single-server systems. We believe that the concepts and metrics that we introduced, such as EOP, CPUE, LUE, RUE, will continue to hold for multi-server systems as well. However, currently we do not have experimental data to quantitatively compare against the traditional EP model for such systems or how close to Dynamic EO the governors make the systems operate.

- **Long-term adaptations**: Our governors target relatively long-term adaptations of resources. The DVFS and prefetching governors (Chapters 2 and 3) work over hundreds of milliseconds while our cache governors (Chapters 4 and 5) work over seconds of execution time. Cache adaptations over short time intervals may not be attractive due to significant overheads in reconfiguration and subsequent warmup. A short time interval for our DVFS governor is suboptimal (**R(1)**, see Chapter 3).

- **Simulation study**: We use a real system (HS) for our DVFS and prefetching studies (Chapters 2 and 3). Real systems today support dynamic reconfiguration for only a small set of resources. On HS, dynamic reconfiguration of cache sizes is not supported. Some other recent systems [109] support cache space allocation to cores/applications, but are limited in the modes of reconfiguration (it is not clear if both the number of sets and ways can be changed). Moreover, hardware support for inspecting or sampling addresses of cache accesses is missing. Instrumentation-based frameworks [143] may not fully address the needs because usually they do not track OS kernel accesses to the cache. Thus, we use full-system simulation to study cache reconfigurations (Chapters 4 and 5). We believe that simulation is an indispensable tool for fully exploring this space. Simulators are flexible and provide insight but may be inaccurate with respect to real system implementations.

- **Prefetching models**: Our governors in Chapter 3 account for the effects of hardware prefetching and can handle prefetching reconfigurability in terms of enabling or disabling it. Further reconfigurability for prefetching, e.g., dynamically changing prefetch depths and strides, may need analytical modeling for prefetching impact as a function of configuration. It is unfortunate that real systems today do not support such reconfigurability. Our cache models in Chapter 4 and governors in Chapter 5 do not account for potential changes in the address stream due to prefetching reconfigurations. This limitation, however, is not unique to our cache models but affects almost all, if not all, current analytical models for cache performance.

- **Throughput metrics**: Our governors have focused on throughput-based metrics for performance, e.g., BIPS, transactions per second, etc. that may be required to also satisfy some constraints, e.g., in the case of SPECpower. Our governors currently do not focus on latency-based metrics, e.g., response time or tail latency

distributions that may be important for some applications, e.g., interactive online applications, high-frequency trading applications, etc.

Future research will focus on removing these limitations. Recent work [125, 171] has proposed techniques for fast adaptation, but doing so in a workload-agnostic manner and without changes to the underlying OS is hard. Cache reconfigurations will likely be at long-term intervals. Integrating cache performance models with prefetching models [48] seems to be an interesting direction for further exploration.

Current studies have not fully explored simultaneous reconfiguration of multiple different types of knobs within the same server. Our classification scheme in Chapter 6 highlights the potential for future work in the area. One of the challenges in doing such work is the lack of availability of many dynamically reconfigurable knobs in real systems. This makes full-system simulation a necessity for exploring this space. A number of modern systems support reconfiguring the number of logical cores and socket-level DVFS, but fewer systems support per-core DVFS. Only a handful of Xeon models support cache capacity allocation. Hardware support for inspecting cache access streams is absent. Hardware prefetching control is extremely limited (only enable/disable). Some other knobs such as QPI (Quick Path Interconnect) speed and memory speed can be configured only at system boot time. Making these and other knobs dynamically reconfigurable will greatly help OS schedulers to operate systems more efficiently.

A SPECPOWER POWER-PERFORMANCE

Figure A.1 shows the power-performance state space for SPECpower on HS for various configurations chosen statically and fixed throughout the entire run. Each subfigure shows particular combinations of number of cores (4 cores or 1 core) and memory frequency (1600 MHz or 1067 MHz) and all possible DVFS levels for each combination. We keep the default setting of prefetching enabled for all runs.

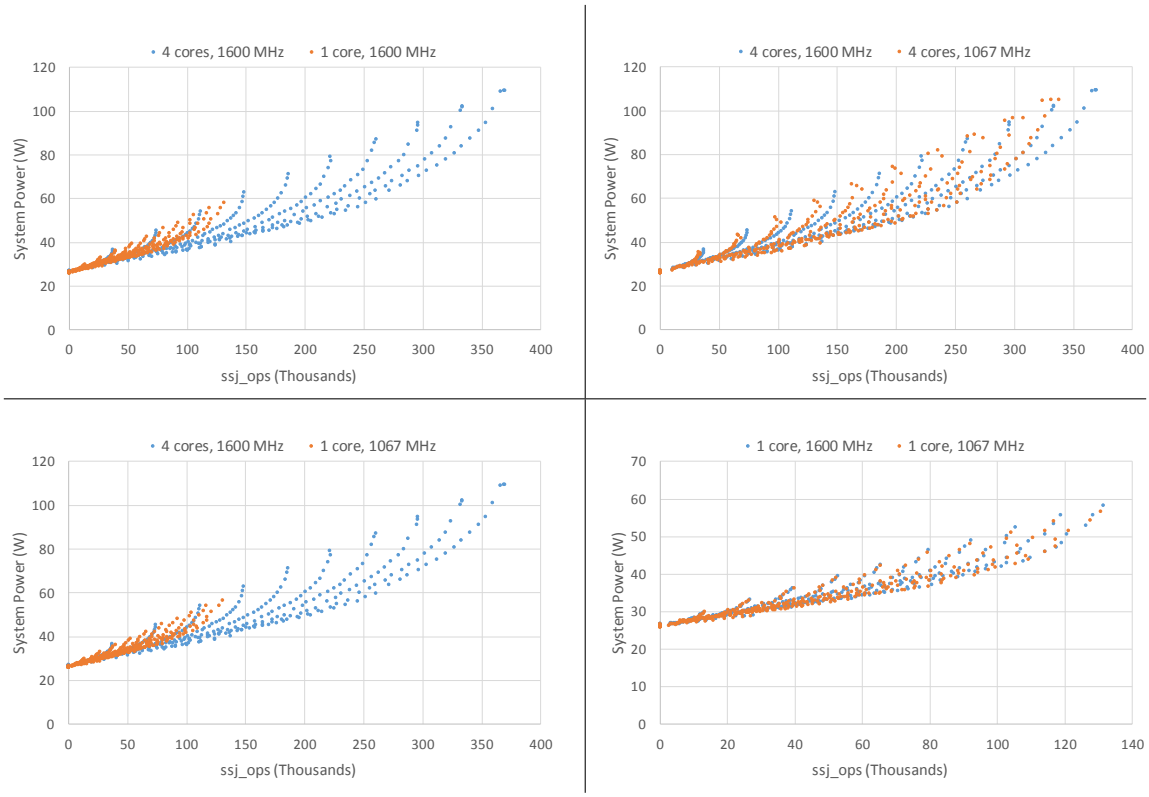


Figure A.1: SPECpower power-performance with different configurations.

Reducing memory frequency affects bandwidth, but has noticeable impact only when there are more active cores to generate memory traffic. Reducing the number of cores or memory frequency reduces the maximum load that can be served, but does not significantly lower the Pareto frontier at low loads.

Bibliography

[1] 3M. Two-phase immersion cooling: A revolution in data center efficiency. http://multimedia.3m.com/mws/media/1127920O/2-phase-immersion-coolinga-revolution-in-data-center-efficiency.pdf.

[2] Dennis Abts, Michael R. Marty, Philip M. Wells, Peter Klausler, and Hong Liu. Energy proportional datacenter networks. In *Proceedings of the 37th Annual International Symposium on Computer Architecture*, ISCA '10, pages 338–347, New York, NY, USA, 2010. ACM.

[3] A. Agarwal, J. Hennessy, and M. Horowitz. An analytical cache model. *ACM Transactions on Computer Systems*, 7(2):184–215, May 1989.

[4] Alaa R. Alameldeen, Milo M. K. Martin, Carl J. Mauer, Kevin E. Moore, Min Xu, Mark D. Hill, David A. Wood, and Daniel J. Sorin. Simulating a $2M commercial server on a $2K PC. *Computer*, 36(2):50–57, February 2003.

[5] Alaa R. Alameldeen and David A. Wood. Adaptive cache compression for high-performance processors. In *Proceedings of the 31st Annual International Symposium on Computer Architecture*, ISCA '04, pages 212–223, Washington, DC, USA, 2004. IEEE Computer Society.

[6] Alaa R. Alameldeen and David A. Wood. IPC considered harmful for multiprocessor workloads. *IEEE Micro*, 26(4):8–17, July 2006.

[7] A.R. Alameldeen and D.A. Wood. Interactions between compression and prefetching in chip multiprocessors. In *IEEE 13th International Symposium on High Performance Computer Architecture*, pages 228–239, Feb 2007.

[8] Jorge Albericio, Pablo Ibáñez, Víctor Viñals, and José M. Llabería. The reuse cache: Downsizing the shared last-level cache. In *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO-46, pages 310–321, New York, NY, USA, 2013. ACM.

[9] David H. Albonesi. Dynamic IPC/clock rate optimization. In *Proceedings of the 25th Annual International Symposium on Computer Architecture*, ISCA '98, pages 282–292, Washington, DC, USA, 1998. IEEE Computer Society.

[10] David H. Albonesi. Selective cache ways: On-demand cache resource allocation. In *Proceedings of the 32Nd Annual ACM/IEEE International Symposium on Microarchitecture*, MICRO 32, pages 248–259, Washington, DC, USA, 1999. IEEE Computer Society.

[11] David H. Albonesi, Rajeev Balasubramonian, Steven G. Dropsho, Sandhya Dwarkadas, Eby G. Friedman, Michael C. Huang, Volkan Kursun, Grigorios Magklis, Michael L. Scott, Greg Semeraro, Pradip Bose, Alper Buyuktosunoglu, Peter W. Cook, and Stanley E. Schuster. Dynamically tuning processor resources with adaptive processing. *Computer*, 36(12):49–58, December 2003.

[12] George Almási, Călin Caşcaval, and David A. Padua. Calculating stack distances efficiently. *SIGPLAN Notices*, 38(2 supplement):37–43, June 2002.

[13] George Almási, Călin Caşcaval, and David A. Padua. Calculating stack distances efficiently. In *Proceedings of the 2002 Workshop on Memory System Performance*, MSP '02, pages 37–43, New York, NY, USA, 2002. ACM.

[14] Martin Alt, Christian Ferdinand, Florian Martin, and Reinhard Wilhelm. Cache behavior prediction by abstract interpretation. In *Proceedings of the Third International Symposium on Static Analysis*, SAS '96, pages 52–66, London, UK, UK, 1996. Springer-Verlag.

[15] Vishal Aslot, Max J. Domeika, Rudolf Eigenmann, Greg Gaertner, Wesley B. Jones, and Bodo Parady. SPEComp: A new benchmark suite for measuring parallel computer performance. In *Proceedings of the International Workshop on OpenMP Applications and Tools: OpenMP Shared Memory Parallel Programming*, WOMPAT '01, pages 1–10, London, UK, UK, 2001. Springer-Verlag.

[16] Victor Avelar, Dan Azevedo, and Alan French, editors. *PUE: A Comprehensive Examination of the Metric*. The Green Grid, 2012.

[17] Jean-Loup Baer and Tien-Fu Chen. Effective hardware-based data prefetching for high-performance processors. *IEEE Transactions on Computers*, 44(5):609–623, May 1995.

[18] R. Iris Bahar and Srilatha Manne. Power and energy reduction via pipeline balancing. In *Proceedings of the 28th Annual International Symposium on Computer Architecture*, ISCA '01, pages 218–229, New York, NY, USA, 2001. ACM.

[19] Peter E. Bailey, Aniruddha Marathe, David K. Lowenthal, Barry Rountree, and Martin Schulz. Finding the limits of power-constrained application performance. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '15, pages 79:1–79:12, New York, NY, USA, 2015. ACM.

[20] Rajeev Balasubramonian, David Albonesi, Alper Buyuktosunoglu, and Sandhya Dwarkadas. Memory hierarchy reconfiguration for energy and performance in general-purpose processor architectures. In *Proceedings of the 33rd Annual ACM/IEEE International Symposium on Microarchitecture*, MICRO 33, pages 245–257, New York, NY, USA, 2000. ACM.

[21] Luiz André Barroso, Jimmy Clidaras, and Urs Hölzle. *The Datacenter As a Computer: An Introduction to the Design of Warehouse-Scale Machines*. Morgan & Claypool Publishers, second edition, 2013.

[22] Luiz André Barroso and Urs Hölzle. The case for energy-proportional computing. *Computer*, 40(12):33–37, December 2007.

[23] Bradford M. Beckmann, Michael R. Marty, and David A. Wood. ASR: Adaptive selective replication for CMP caches. In *Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO 39, pages 443–454, Washington, DC, USA, 2006. IEEE Computer Society.

[24] Nathan Beckmann and Daniel Sanchez. Cache calculus: Modeling caches through differential equations. *Computer Architecture Letters (CAL)*, 15(1), 2016.

[25] Laszlo A. Belady. A study of replacement algorithms for a virtual-storage computer. *IBM Systems Journal*, 5(2):78–101, June 1966.

[26] Adam Belay, George Prekas, Ana Klimovic, Samuel Grossman, Christos Kozyrakis, and Edouard Bugnion. IX: A protected dataplane operating system for high throughput and low latency. In *Proceedings of the 11th USENIX Conference on Operating Systems Design and Implementation*, OSDI'14, pages 49–65, Berkeley, CA, USA, 2014. USENIX Association.

[27] Luca Benini, Alessandro Bogliolo, Giuseppe A. Paleologo, and Giovanni De Micheli. Policy optimization for dynamic power management. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 18(6):813–833, Jun 1999.

[28] Eklov Berg and Erik Hagersten. StatCache: A probabilistic approach to efficient and accurate data locality analysis. In *Proceedings of the 2004 IEEE International Symposium on Performance Analysis of Systems and Software*, ISPASS '04, pages 20–27, Washington, DC, USA, 2004. IEEE Computer Society.

[29] Kristof Beyls and Erik D'Hollander. Reuse distance as a metric for cache behavior. In *Proceedings of the IASTED International Conference on Parallel and Distributed Computing and Systems (PDCS)*, pages 617–622, 2001.

[30] Christian Bienia. *Benchmarking Modern Multiprocessors*. PhD thesis, Princeton University, January 2011.

[31] Ramazan Bitirgen, Engin Ipek, and Jose F. Martinez. Coordinated management of multiple interacting resources in chip multiprocessors: A machine learning approach. In *Proceedings of the 41st Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO 41, pages 318–329, Washington, DC, USA, 2008. IEEE Computer Society.

[32] Burton H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, July 1970.

[33] Allan G. Bluman. *Elementary Statistics: A Step by Step Approach*. McGraw-Hill, seventh edition, 2008.

[34] Arnon Boneh and Micha Hofri. The coupon-collector problem revisited — a survey of engineering problems and computational methods. *Communications in Statistics. Stochastic Models*, 13(1):39–66, 1997.

[35] Shekhar Borkar and Andrew A. Chien. The future of microprocessors. *Communications of the ACM*, 54(5):67–77, May 2011.

[36] Pradip Bose, Alper Buyuktosunoglu, John A. Darringer, Meeta S. Gupta, Michael B. Healy, Hans Jacobson, Indira Nair, Jude A. Rivers, Jeonghee Shin, Augusto Vega, and Alan J. Weger. Power management of multi-core chips: Challenges and pitfalls. In *Proceedings of the Conference on Design, Automation and Test in Europe*, DATE '12, pages 977–982, San Jose, CA, USA, 2012. EDA Consortium.

[37] Dominik Brodowski and Nico Golde. CPU frequency and voltage scaling code in the Linux(TM) kernel. Linux CPUFreq. CPUFreq governors. `https://www.kernel.org/doc/Documentation/cpu-freq/governors.txt`.

[38] J. Adam Butts and Gurindar S. Sohi. A static power model for architects. In *Proceedings of the 33rd Annual ACM/IEEE International Symposium on Microarchitecture*, MICRO 33, pages 191–201, New York, NY, USA, 2000. ACM.

[39] Alper Buyuktosunoglu, Tejas Karkhanis, David H. Albonesi, and Pradip Bose. Energy efficient co-adaptive instruction fetch and issue. In *Proceedings of the 30th Annual International Symposium on Computer Architecture*, ISCA '03, pages 147–156, New York, NY, USA, 2003. ACM.

[40] Alper Buyuktosunoglu, Stanley Schuster, David Brooks, Pradip Bose, Peter W. Cook, and David H. Albonesi. An adaptive issue queue for reduced power at high performance. In *Proceedings of the First International Workshop on Power-Aware Computer Systems-Revised Papers*, PACS '00, pages 25–39, London, UK, UK, 2001. Springer-Verlag.

[41] Surendra Byna, Yong Chen, and Xian-He Sun. Taxonomy of data prefetching for multicore processors. *Journal of Computer Science and Technology*, 24(3):405–417, 2009.

[42] J. Lawrence Carter and Mark N. Wegman. Universal classes of hash functions (extended abstract). In *Proceedings of the Ninth Annual ACM Symposium on Theory of Computing*, STOC '77, pages 106–112, New York, NY, USA, 1977. ACM.

[43] Koushik Chakraborty. *Over-provisioned Multicore Systems*. PhD thesis, University of Wisconsin-Madison, 2008.

[44] Koushik Chakraborty, Philip M. Wells, and Gurindar S. Sohi. Computation spreading: Employing hardware migration to specialize CMP cores on-the-fly. In *Proceedings of the 12th International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS XII, pages 283–292, New York, NY, USA, 2006. ACM.

[45] Jichuan Chang and Gurindar S. Sohi. Cooperative caching for chip multiprocessors. In *Proceedings of the 33rd Annual International Symposium on Computer Architecture*, ISCA '06, pages 264–276, Washington, DC, USA, 2006. IEEE Computer Society.

[46] Jonathan Chang, Szu-Liang Chen, Wei Chen, Siufu Chiu, Robert Faber, Raghuraman Ganesan, Marijana Grgek, Venkata Lukka, Wei Wing Mar, James Vash, Stefan Rusu, and Kevin Zhang. A 45nm 24MB on-die L3 cache for the 8-core multi-threaded Xeon processor. In *2009 Symposium on VLSI Circuits*, pages 152–153, June 2009.

[47] Jeffrey S. Chase, Darrell C. Anderson, Prachi N. Thakar, Amin M. Vahdat, and Ronald P. Doyle. Managing energy and server resources in hosting centers. In *Proceedings of the Eighteenth ACM Symposium on Operating Systems Principles*, SOSP '01, pages 103–116, New York, NY, USA, 2001. ACM.

[48] Xi E. Chen and Tor M. Aamodt. Hybrid analytical modeling of pending cache hits, data prefetching, and MSHRs. *ACM Transactions on Architecture and Code Optimization (TACO)*, 8(3):10:1–10:28, October 2011.

[49] Derek Chiou, Srinivas Devadas, Larry Rudolph, Boon S. Ang, Derek Chiouy, Derek Chiouy, Larry Rudolphy, Larry Rudolphy, Srinivas Devadasy, Srinivas Devadasy, Boon S. Angz, and Boon S. Angz. Dynamic cache partitioning via columnization. In *In Proceedings of Design Automation Conference*, 2000.

[50] C. K. Chow. Determination of cache's capacity and its matching storage hierarchy. *IEEE Transactions on Computers*, 25(2):157–164, February 1976.

[51] Ryan Cochran, Can Hankendi, Ayse K. Coskun, and Sherief Reda. Pack & cap: Adaptive DVFS and thread packing under power caps. In *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO-44, pages 175–185, New York, NY, USA, 2011. ACM.

[52] Intel Corporation. Intel Performance Counter Monitor - a better way to measure CPU utilization. `https://software.intel.com/en-us/articles/intel-performance-counter-monitor`, 2012.

[53] Intel Corporation. Improving real-time performance by utilizing cache allocation technology. *White Paper, Document Number: 331843-001US*, April 2015.

[54] Matthew Curtis-Maury, Ankur Shah, Filip Blagojevic, Dimitrios S. Nikolopoulos, Bronis R. de Supinski, and Martin Schulz. Prediction models for multi-dimensional power-performance optimization on many cores. In *Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques*, PACT '08, pages 250–259, New York, NY, USA, 2008. ACM.

[55] Robert Cypher. Apparatus and method for determining stack distance including spatial locality of running software for estimating cache miss rates based upon contents of a hash table, 2008. US Patent 7366871.

[56] Robert Cypher. Apparatus and method for determining stack distance of running software for estimating cache miss rates based upon contents of a hash table, 2008. US Patent 7373480.

[57] F. Dahlgren, M. Dubois, and P. Stenstrom. Fixed and adaptive sequential prefetching in shared memory multiprocessors. In *International Conference on Parallel Processing (ICPP)*, volume 1, pages 56–63, Aug 1993.

[58] George B. Dantzig. Discrete-variable extremum problems. *Operations Research*, 5(2):266–288, 1957.

[59] Howard David, Chris Fallin, Eugene Gorbatov, Ulf R. Hanebutte, and Onur Mutlu. Memory power management via dynamic voltage/frequency scaling. In *Proceedings of the 8th ACM International Conference on Autonomic Computing*, ICAC '11, pages 31–40, New York, NY, USA, 2011. ACM.

[60] Christina Delimitrou and Christos Kozyrakis. Quasar: Resource-efficient and qos-aware cluster management. In *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '14, pages 127–144, New York, NY, USA, 2014. ACM.

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

[62] Qingyuan Deng, David Meisner, Luiz Ramos, Thomas F. Wenisch, and Ricardo Bianchini. MemScale: Active low-power modes for main memory. In *Proceedings of the Sixteenth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS XVI, pages 225–238, New York, NY, USA, 2011. ACM.

[63] R. H. Dennard, F. H. Gaensslen, V. L. Rideout, E. Bassous, and A. R. LeBlanc. Design of ion-implanted MOSFET's with very small physical dimensions. *IEEE Journal of Solid-State Circuits*, 9(5):256–268, Oct 1974.

[64] Ashutosh S. Dhodapkar and James E. Smith. Saving and restoring implementation contexts with co-designed virtual machines. In *In Workshop on Complexity-Effective Design*, June 2001.

[65] Chen Ding and Yutao Zhong. Reuse distance analysis. Technical Report UR-CS-TR-741, University of Rochester, 2001.

[66] Jack Dongarra and Michael A. Heroux. Toward a new metric for ranking high performance computing systems. Technical Report SAND2013-4744, Sandia National Laboratories, June 2013.

[67] Steve Dropsho, Alper Buyuktosunoglu, Rajeev Balasubramonian, David H. Albonesi, Sandhya Dwarkadas, Greg Semeraro, Grigorios Magklis, and Michael L. Scott. Integrating adaptive on-chip storage structures for reduced dynamic power. In *Proceedings of the 2002 International Conference on Parallel Architectures and Compilation Techniques*, PACT '02, pages 141–152, Washington, DC, USA, 2002. IEEE Computer Society.

[68] Steven Dropsho, Greg Semeraro, David H. Albonesi, Grigorios Magklis, and Michael L. Scott. Dynamically trading frequency for complexity in a gals microprocessor. In *Proceedings of the 37th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO 37, pages 157–168, Washington, DC, USA, 2004. IEEE Computer Society.

[69] Nam Duong, Dali Zhao, Taesu Kim, Rosario Cammarota, Mateo Valero, and Alexander V. Veidenbaum. Improving cache management policies using dynamic reuse distances. In

Proceedings of the 2012 45th Annual IEEE/ACM International Symposium on Microarchitecture, MICRO-45, pages 389–400, Washington, DC, USA, 2012. IEEE Computer Society.

[70] Yasuko Eckert, Srilatha Manne, Michael J. Schulte, and David A. Wood. Something old and something new: P-states can borrow microarchitecture techniques too. In *Proceedings of the 2012 ACM/IEEE International Symposium on Low Power Electronics and Design*, ISLPED '12, pages 385–390, New York, NY, USA, 2012. ACM.

[71] David Eklov and Erik Hagersten. StatStack: Efficient modeling of lru caches. In *Proceedings of the 2010 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2010.

[72] Joel S. Emer and Douglas W. Clark. A characterization of processor performance in the Vax-11/780. In *Proceedings of the 11th Annual International Symposium on Computer Architecture*, ISCA '84, pages 301–310, New York, NY, USA, 1984. ACM.

[73] Facebook. PUE/WUE–Prineville. `https://www.facebook.com/PrinevilleDataCenter/app/399244020173259/`.

[74] Babak Falsafi and Thomas F. Wenisch. *A Primer on Hardware Prefetching*. Synthesis Lectures on Computer Architecture. Morgan & Claypool Publishers, 2014.

[75] Babak Falsafi and David A. Wood. Modeling cost/performance of a parallel computer simulator. *ACM Trans. Model. Comput. Simul.*, 7(1):104–130, January 1997.

[76] Wes Felter, Karthick Rajamani, Tom Keller, and Cosmin Rusu. A performance-conserving approach for reducing peak power consumption in server systems. In *Proceedings of the 19th Annual International Conference on Supercomputing*, ICS '05, pages 293–302, New York, NY, USA, 2005. ACM.

[77] Krisztián Flautner, Nam Sung Kim, Steve Martin, David Blaauw, and Trevor Mudge. Drowsy caches: Simple techniques for reducing leakage power. In *Proceedings of the 29th Annual International Symposium on Computer Architecture*, ISCA '02, pages 148–157, Washington, DC, USA, 2002. IEEE Computer Society.

[78] Michael J. Flynn. Some computer organizations and their effectiveness. *IEEE Transactions on Computers*, C-21(9):948–960, Sept 1972.

[79] Jayesh Gaur, Mainak Chaudhuri, and Sreenivas Subramoney. Bypass and insertion algorithms for exclusive last-level caches. In *Proceedings of the 38th Annual International Symposium on Computer Architecture*, ISCA '11, pages 81–92, New York, NY, USA, 2011. ACM.

[80] Soraya Ghiasi, Jason Casmira, and Dirk Grunwald. Using IPC variation in workloads with externally specified rates to reduce power consumption. In *In Workshop on Complexity Effective Design*, 2000.

[81] Dan Gibson and David A. Wood. Forwardflow: A scalable core for power-constrained CMPs. In *Proceedings of the 37th Annual International Symposium on Computer Architecture*, ISCA '10, pages 14–25, New York, NY, USA, 2010. ACM.

[82] Ricardo Gonzalez and Mark Horowitz. Energy dissipation in general purpose microprocessors. *IEEE Journal of Solid-State Circuits*, 31(9):1277–1284, Sep 1996.

[83] Google. Efficiency: How we do it. `http://www.google.com/about/datacenters/efficiency/internal/index.html#measuring-efficiency`.

[84] Ann Gordon-Ross, Pablo Viana, Frank Vahid, Walid Najjar, and Edna Barros. A one-shot configurable-cache tuner for improved energy and performance. In *Proceedings of the conference on Design, automation and test in Europe*, DATE '07, pages 755–760, San Jose, CA, USA, 2007. EDA Consortium.

[85] Edward H. Gornish and Alexander Veidenbaum. An integrated hardware/software data prefetching scheme for shared-memory multiprocessors. *International Journal of Parallel Programming*, 27(1):35–70, February 1999.

[86] Kinshuk Govil, Edwin Chan, and Hal Wasserman. Comparing algorithm for dynamic speed-setting of a low-power CPU. In *Proceedings of the 1st Annual International Conference on Mobile Computing and Networking*, MobiCom '95, pages 13–25, New York, NY, USA, 1995. ACM.

[87] Venkatraman Govindaraju, Chen-Han Ho, and Karthikeyan Sankaralingam. Dynamically specialized datapaths for energy efficient computing. In *Proceedings of the 2011 IEEE 17th International Symposium on High Performance Computer Architecture*, HPCA '11, pages 503–514, Washington, DC, USA, 2011. IEEE Computer Society.

[88] Graph 500. Graph 500 reference implementation v2.1.4. `http://www.graph500.org/referencecode`.

[89] Daniel Grund. *Static Cache Analysis for Real-Time Systems – LRU, FIFO, PLRU*. PhD thesis, Saarland University, 2012.

[90] Daniel Grund and Jan Reineke. Estimating the performance of cache replacement policies. In *MEMOCODE '08: Proceedings of the 6th IEEE/ACM International Conference on Formal Methods and Models for Codesign*, pages 101–111, June 2008.

[91] Fei Guo and Yan Solihin. An analytical model for cache replacement policy performance. In *Proceedings of the Joint International Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS '06/Performance '06, pages 228–239, New York, NY, USA, 2006. ACM.

[92] Yao Guo, Pritish Narayanan, Mahmoud Abdullah Bennaser, Saurabh Chheda, and Csaba Andras Moritz. Energy-efficient hardware data prefetching. *IEEE Transactions On Very Large Scale Integration Systems*, 19(2):250–263, February 2011.

[93] S. Gupta and H. Zhou. Spatial locality-aware cache partitioning for effective cache sharing. In *44th International Conference on Parallel Processing (ICPP)*, pages 150–159, Sept 2015.

[94] Saurabh Gupta, Hongliang Gao, and Huiyang Zhou. Adaptive cache bypassing for inclusive last level caches. In *IPDPS*, pages 1243–1253. IEEE Computer Society, 2013.

[95] Daniel Hackenberg, Robert Schöne, Thomas Ilsche, Daniel Molka, Joseph Schuchart, and Robin Geyer. An energy efficiency feature survey of the Intel Haswell processor. In *Workshop on High-Performance Power-Aware Computing*, 2015.

[96] Marcus Hähnel, Björn Döbel, Marcus Völp, and Hermann Härtig. Measuring energy consumption for short code paths using RAPL. *SIGMETRICS Performance Evaluation Review*, 40(3):13–17, January 2012.

[97] Hadi Hajimiri, Kamran Rahmani, and Prabhat Mishra. Compression-aware dynamic cache reconfiguration for embedded systems. *Sustainable Computing: Informatics and Systems*, 2(2):71–80, 2012.

[98] P. Hammarlund, A.J. Martinez, A.A. Bajwa, D.L. Hill, E. Hallnor, Hong Jiang, M. Dixon, M. Derr, M. Hunsaker, R. Kumar, R.B. Osborne, R. Rajwar, R. Singhal, R. D'Sa, R. Chappell, S. Kaushik, S. Chennupaty, S. Jourdan, S. Gunther, T. Piazza, and T. Burton. Haswell: The fourth-generation Intel Core processor. *IEEE Micro*, 34(2):6–20, Mar 2014.

[99] Heather Hanson, Stephen W. Keckler, Soraya Ghiasi, Karthick Rajamani, Freeman Rawson, and Juan Rubio. Thermal response to DVFS: Analysis with an Intel Pentium M. In *Proceedings of the 2007 International Symposium on Low Power Electronics and Design*, ISLPED '07, pages 219–224, New York, NY, USA, 2007. ACM.

[100] A. Hartstein, V. Srinivasan, T. R. Puzak, and P. G. Emma. Cache miss behavior: is it $\sqrt{2}$? In *Proceedings of the 3rd conference on Computing frontiers*, CF '06, pages 313–320, New York, NY, USA, 2006. ACM.

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

[102] Hewlett-Packard Corporation, Intel Corporation, Microsoft Corporation, Phoenix Technologies Ltd., and Toshiba Corporation. Advanced configuration and power interface specification, revision 5.0, December 2011.

[103] Mark D. Hill and Alan Jay Smith. Evaluating associativity in CPU caches. *IEEE Transactions on Computers*, 38(12):1612–1630, December 1989.

[104] Urs Hölzle and Luiz André Barroso. *The Datacenter As a Computer: An Introduction to the Design of Warehouse-Scale Machines*. Morgan & Claypool Publishers, first edition, 2009.

[105] M. Horowitz. Computing's energy problem (and what we can do about it). In *IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, pages 10–14, Feb 2014.

[106] Chung-Hsing Hsu and S.W. Poole. Revisiting server energy proportionality. In *2013 42nd International Conference on Parallel Processing (ICPP)*, pages 834–840, Oct 2013.

[107] Electronic Educational Devices Inc. Watts Up? .Net. `https://www.wattsupmeters.com/secure/products.php?pn=0&wai=0&spec=2`.

[108] Uptime Institute. 2014 data center industry survey, 2014.

[109] Intel Corporation. Cache monitoring technology and cache allocation technology. `http://www.intel.com/content/www/us/en/communications/cache-monitoring-cache-allocation-technologies.html`.

[110] Intel Corporation. Enhanced Intel SpeedStep technology for the Intel Pentium M processor, March 2004.

[111] Intel Corporation. Intel turbo boost technology in Intel Core microarchitecture (Nehalem) based processors, Nov 2008.

[112] Intel Corporation. Disclosure of H/W prefetcher control on some Intel processors. `https://software.intel.com/en-us/articles/disclosure-of-hw-prefetcher-control-on-some-intel-processors`, September 2014.

[113] Intel Corporation. Intel 64 and IA-32 architectures software developer's manual, vol. 3B: System programming guide, part 2, 2015.

[114] Engin Ipek, Meyrem Kirman, Nevin Kirman, and Jose F. Martinez. Core fusion: Accommodating software diversity in chip multiprocessors. In *Proceedings of the 34th Annual International Symposium on Computer Architecture*, ISCA '07, pages 186–197, New York, NY, USA, 2007. ACM.

[115] Canturk Isci, Alper Buyuktosunoglu, Chen-Yong Cher, Pradip Bose, and Margaret Martonosi. An analysis of efficient multi-core global power management policies: Maximizing performance for a given power budget. In *Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO 39, pages 347–358, Washington, DC, USA, 2006. IEEE Computer Society.

[116] Sanjeev Jahagirdar, Varghese George, Inder Sodhi, and Ryan Wells. Power management of the third generation Intel Core micro architecture formerly codenamed Ivy Bridge. In *Hot Chips 24*, 2012.

[117] Aamer Jaleel, William Hasenplaugh, Moinuddin Qureshi, Julien Sebot, Simon Steely, Jr., and Joel Emer. Adaptive insertion policies for managing shared caches. In *Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques*, PACT '08, pages 208–219, New York, NY, USA, 2008. ACM.

[118] Aamer Jaleel, William Hasenplaugh, Moinuddin Qureshi, Julien Sebot, Simon Steely, Jr., and Joel Emer. Adaptive insertion policies for managing shared caches. In *Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques*, PACT '08, pages 208–219, New York, NY, USA, 2008. ACM.

[119] Aamer Jaleel, Kevin B. Theobald, Simon C. Steely, Jr., and Joel Emer. High performance cache replacement using re-reference interval prediction (RRIP). In *Proceedings of the 37th Annual International Symposium on Computer Architecture*, ISCA '10, pages 60–71, New York, NY, USA, 2010. ACM.

[120] Daniel A. Jiménez. Insertion and promotion for tree-based PseudoLRU last-level caches. In *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO-46, pages 284–296, New York, NY, USA, 2013. ACM.

[121] Md Kamruzzaman, Steven Swanson, and Dean M. Tullsen. Underclocked software prefetching: More cores, less energy. *IEEE Micro*, 32(4):32–41, July 2012.

[122] Svilen Kanev, Kim Hazelwood, Gu-Yeon Wei, and David Brooks. Tradeoffs between power management and tail latency in warehouse-scale applications. In *2014 IEEE International Symposium on Workload Characterization (IISWC)*, pages 31–40, Oct 2014.

[123] Ramakrishna Karedla, J. Spencer Love, and Bradley G. Wherry. Caching strategies to improve disk system performance. *Computer*, 27(3):38–46, March 1994.

[124] Dimitris Kaseridis, Jeffrey Stuecheli, and Lizy K. John. Bank-aware dynamic cache partitioning for multicore architectures. In *Proceedings of the 2009 International Conference on Parallel Processing*, ICPP '09, pages 18–25, Washington, DC, USA, 2009. IEEE Computer Society.

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

[126] Stefanos Kaxiras, Zhigang Hu, and Margaret Martonosi. Cache decay: Exploiting generational behavior to reduce cache leakage power. In *Proceedings of the 28th Annual International Symposium on Computer Architecture*, ISCA '01, pages 240–251, New York, NY, USA, 2001. ACM.

[127] K. Kedzierski, M. Moreto, F.J. Cazorla, and M. Valero. Adapting cache partitioning algorithms to pseudo-LRU replacement policies. In *Proceedings of the 24th IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 1–12, 2010.

[128] G. Keramidas, C. Datsios, and S. Kaxiras. A framework for efficient cache resizing. In *International Conference on Embedded Computer Systems (SAMOS)*, pages 76–85, July 2012.

[129] R. E. Kessler, Mark D. Hill, and David A. Wood. A comparison of trace-sampling techniques for multi-megabyte caches. *IEEE Trans. Comput.*, 43(6):664–675, June 1994.

[130] Hyungjun Kim, Pritha Ghoshal, Boris Grot, Paul V. Gratz, and Daniel A. Jiménez. Reducing network-on-chip energy consumption through spatial locality speculation. In *Proceedings of the Fifth ACM/IEEE International Symposium on Networks-on-Chip*, NOCS '11, pages 233–240, New York, NY, USA, 2011. ACM.

[131] Jonathan D. Kron, Brooks Prumo, and Gabriel H. Loh. Double-DIP: Augmenting DIP with adaptive promotion policies to manage shared L2 caches. In *Proceedings of the 2nd Workshop on Chip Multiprocessor Memory Systems and Interconnects (CMP-MSI)*, 2008.

[132] Subhasis Laha. *Accurate Low-cost Methods for Performance Evaluation of Cache Memory Systems*. PhD thesis, University of Illinois, Urbana, IL, USA, 1988.

[133] Subhasis Laha, Janak H. Patel, and Ravishankar K. Iyer. Accurate low-cost methods for performance evaluation of cache memory systems. *IEEE Trans. Comput.*, 37(11):1325–1336, November 1988.

[134] H. Q. Le, W. J. Starke, J. S. Fields, F. P. O'Connell, D. Q. Nguyen, B. J. Ronchetti, W. M. Sauer, E. M. Schwarz, and M. T. Vaden. IBM POWER6 microarchitecture. *IBM Journal of Research and Development*, 51(6):639–662, November 2007.

[135] D. Lee, J. Choi, J. H. Kim, S. H. Noh, S. L. Min, Y. Cho, and C. S. Kim. LRFU: A spectrum of policies that subsumes the least recently used and least frequently used policies. *IEEE Transactions on Computers*, 50(12):1352–1361, December 2001.

[136] J. Li and J.F. Martinez. Dynamic power-performance adaptation of parallel computation on chip multiprocessors. In *The Twelfth International Symposium on High-Performance Computer Architecture*, pages 77–87, Feb 2006.

[137] Xiaodong Li, Ritu Gupta, Sarita V. Adve, and Yuanyuan Zhou. Cross-component energy management: Joint adaptation of processor and memory. *ACM Transactions on Architecture Code Optimization*, 4(3), September 2007.

[138] Fang Liu, Fei Guo, Yan Solihin, Seongbeom Kim, and Abdulaziz Eker. Characterizing and modeling the behavior of context switch misses. In *Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques*, PACT '08, pages 91–101, New York, NY, USA, 2008. ACM.

[139] Yanpei Liu, Stark C. Draper, and Nam Sung Kim. SleepScale: Runtime joint speed scaling and sleep states management for power efficient data centers. In *Proceeding of the 41st Annual International Symposium on Computer Architecuture*, ISCA '14, pages 313–324, Piscataway, NJ, USA, 2014. IEEE Press.

[140] Yu Liu and Wei Zhang. Exploiting stack distance to estimate worst-case data cache performance. In *Proceedings of the 2009 ACM Symposium on Applied Computing*, SAC '09, pages 1979–1983, New York, NY, USA, 2009. ACM.

[141] David Lo, Liqun Cheng, Rama Govindaraju, Luiz André Barroso, and Christos Kozyrakis. Towards energy proportionality for large-scale latency-critical workloads. In *Proceeding of the 41st Annual International Symposium on Computer Architecuture*, ISCA '14, pages 301–312, Piscataway, NJ, USA, 2014. IEEE Press.

[142] Gabriel H. Loh and Mark D. Hill. Efficiently enabling conventional block sizes for very large die-stacked DRAM caches. In *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO-44, pages 454–464, New York, NY, USA, 2011. ACM.

[143] Chi-Keung Luk, Robert Cohn, Robert Muth, Harish Patil, Artur Klauser, Geoff Lowney, Steven Wallace, Vijay Janapa Reddi, and Kim Hazelwood. Pin: Building customized program analysis tools with dynamic instrumentation. In *Proceedings of the 2005 ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '05, pages 190–200, New York, NY, USA, 2005. ACM.

[144] Mingsong Lv, Nan Guan, Jan Reineke, Reinhard Wilhelm, and Wang Yi. A survey on static cache analysis for real-time systems. *Leibniz Transactions on Embedded Systems*, 3(1):05–1–05:48, 2016.

[145] Kai Ma, Xue Li, Ming Chen, and Xiaorui Wang. Scalable power control for many-core architectures running multi-threaded applications. In *Proceedings of the 38th Annual International Symposium on Computer Architecture*, ISCA '11, pages 449–460, New York, NY, USA, 2011. ACM.

[146] R Manikantan, Kaushik Rajan, and R Govindarajan. Probabilistic shared cache management (PriSM). In *Proceedings of the 39th Annual International Symposium on Computer Architecture*, ISCA '12, pages 428–439, Washington, DC, USA, 2012. IEEE Computer Society.

[147] Srilatha Manne, Artur Klauser, and Dirk Grunwald. Pipeline gating: Speculation control for energy reduction. In *Proceedings of the 25th Annual International Symposium on Computer Architecture*, ISCA '98, pages 132–141, Washington, DC, USA, 1998. IEEE Computer Society.

[148] Jason Mars, Lingjia Tang, Robert Hundt, Kevin Skadron, and Mary Lou Soffa. Bubble-Up: Increasing utilization in modern warehouse scale computers via sensible co-locations. In *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO-44, pages 248–259, New York, NY, USA, 2011. ACM.

[149] Milo M. K. Martin, Daniel J. Sorin, Bradford M. Beckmann, Michael R. Marty, Min Xu, Alaa R. Alameldeen, Kevin E. Moore, Mark D. Hill, and David A. Wood. Multifacet's general execution-driven multiprocessor simulator (GEMS) toolset. *SIGARCH Computer Architecture News*, 33(4):92–99, November 2005.

[150] R. L. Mattson, J. Gecsei, D. R. Slutz, and I. L. Traiger. Evaluation techniques for storage hierarchies. *IBM Systems Journal*, 9(2):78–117, June 1970.

[151] David Meisner, Brian T. Gold, and Thomas F. Wenisch. PowerNap: Eliminating server idle power. In *Proceedings of the 14th International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS XIV, pages 205–216, New York, NY, USA, 2009. ACM.

[152] David Meisner, Christopher M. Sadler, Luiz André Barroso, Wolf-Dietrich Weber, and Thomas F. Wenisch. Power management of online data-intensive services. In *Proceedings of the 38th Annual International Symposium on Computer Architecture*, ISCA '11, pages 319–330, New York, NY, USA, 2011. ACM.

[153] David Meisner and Thomas F. Wenisch. Does low-power design imply energy efficiency for data centers? In *Proceedings of the 17th IEEE/ACM International Symposium on Low-power Electronics and Design*, ISLPED '11, pages 109–114, Piscataway, NJ, USA, 2011. IEEE Press.

[154] David Meisner and Thomas F. Wenisch. DreamWeaver: Architectural support for deep sleep. In *Proceedings of the Seventeenth International Conference on Architectural Support for*

Programming Languages and Operating Systems, ASPLOS XVII, pages 313–324, New York, NY, USA, 2012. ACM.

[155] Ke Meng, Russ Joseph, Robert P. Dick, and Li Shang. Multi-optimization power management for chip multiprocessors. In *Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques*, PACT '08, pages 177–186, New York, NY, USA, 2008. ACM.

[156] Microsoft. Microsoft's cloud infrastructure: Datacenters and network fact sheet, June 2015.

[157] Asit K. Mishra, Aditya Yanamandra, Reetuparna Das, Soumya Eachempati, Ravi Iyer, N. Vijaykrishnan, and Chita R. Das. RAFT: A router architecture with frequency tuning for on-chip networks. *Journal of Parallel and Distributed Computing*, 71(5):625–640, May 2011.

[158] Gorden E. Moore. Cramming more components onto integrated circuits. *Electronics*, 38(8):114–117, April 1965.

[159] Miquel Moreto, Francisco J. Cazorla, Alex Ramirez, and Mateo Valero. MLP-aware dynamic cache partitioning. In *Proceedings of the 3rd International Conference on High Performance Embedded Architectures and Compilers*, HiPEAC'08, pages 337–352, Berlin, Heidelberg, 2008. Springer-Verlag.

[160] Trevor N. Mudge. Power: A first class design constraint for future architecture and automation. In *Proceedings of the 7th International Conference on High Performance Computing*, HiPC '00, pages 215–224, London, UK, UK, 2000. Springer-Verlag.

[161] Priya Nagpurkar, Chandra Krintz, Michael Hind, Peter F. Sweeney, and V. T. Rajan. Online phase detection algorithms. In *Proceedings of the International Symposium on Code Generation and Optimization*, CGO '06, pages 111–123, Washington, DC, USA, 2006. IEEE Computer Society.

[162] Siva G. Narendra and Anantha Chandrakasan. *Leakage in Nanometer CMOS Technologies (Series on Integrated Circuits and Systems)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2005.

[163] Alon Naveh, Efraim Rotem, Avi Mendelson, Simcha Gochman, Rajshree Chabukswar, Karthik Krishnan, and Arun Kumar. Power and thermal management in the Intel Core Duo processor. *Intel Technology Journal*, 10, 2006.

[164] NRDC and Anthesis. Data center efficiency assessment–scaling up energy efficiency across the data center industry: Evaluating key drivers and barriers, Aug 2014.

[165] Lena Olson and Mark D. Hill. Probabilistic directed writebacks for exclusive caches. Technical Report TR-1831, University of Wisconsin-Madison, February 2016.

[166] OVH. `https://www.ovh.com/ca/en/about-us/green-it.xml`.

[167] Xiaoyue Pan and Bengt Jonsson. A modeling framework for reuse distance-based estimation of cache performance. In *IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pages 62–71. IEEE Computer Society, March 2015.

[168] Sangyoung Park, Jaehyun Park, Donghwa Shin, Yanzhi Wang, Qing Xie, M. Pedram, and Naehyuck Chang. Accurate modeling of the delay and energy overhead of dynamic voltage and frequency scaling in modern microprocessors. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 32(5):695–708, May 2013.

[169] Vidyadhar Phalke and Bhaskarpillai Gopinath. An inter-reference gap model for temporal locality in program behavior. In *Proceedings of the 1995 ACM SIGMETRICS Joint International Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS '95/PERFORMANCE '95, pages 291–300, New York, NY, USA, 1995. ACM.

[170] Fred Pollack. Pollack's rule. `https://en.wikipedia.org/wiki/Pollack's_Rule`.

[171] George Prekas, Mia Primorac, Adam Belay, Christos Kozyrakis, and Edouard Bugnion. Energy proportionality and workload consolidation for latency-critical applications. In *Proceedings of the Sixth ACM Symposium on Cloud Computing*, SoCC '15, pages 342–355, New York, NY, USA, 2015. ACM.

[172] Thomas Roberts Puzak. *Analysis of Cache Replacement-algorithms*. PhD thesis, University of Massachusetts Amherst, 1985.

[173] Moinuddin K. Qureshi, Aamer Jaleel, Yale N. Patt, Simon C. Steely, and Joel Emer. Adaptive insertion policies for high performance caching. In *Proceedings of the 34th Annual International Symposium on Computer Architecture*, ISCA '07, pages 381–391, New York, NY, USA, 2007. ACM.

[174] Moinuddin K. Qureshi, Daniel N. Lynch, Onur Mutlu, and Yale N. Patt. A case for MLP-aware cache replacement. In *Proceedings of the 33rd Annual International Symposium on Computer Architecture*, ISCA '06, pages 167–178, Washington, DC, USA, 2006. IEEE Computer Society.

[175] Moinuddin K. Qureshi and Yale N. Patt. Utility-based cache partitioning: A low-overhead, high-performance, runtime mechanism to partition shared caches. In *Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO 39, pages 423–432, Washington, DC, USA, 2006. IEEE Computer Society.

[176] Moinuddin K. Qureshi, David Thompson, and Yale N. Patt. The V-Way cache: Demand based associativity via global replacement. In *Proceedings of the 32Nd Annual International Symposium on Computer Architecture*, ISCA '05, pages 544–555, Washington, DC, USA, 2005. IEEE Computer Society.

[177] Krishna K. Rangan, Gu-Yeon Wei, and David Brooks. Thread motion: Fine-grained power management for multi-core systems. In *Proceedings of the 36th Annual International Symposium on Computer Architecture*, ISCA '09, pages 302–313, New York, NY, USA, 2009. ACM.

[178] Parthasarathy Ranganathan, Sarita Adve, and Norman P. Jouppi. Reconfigurable caches and their application to media processing. In *Proceedings of the 27th Annual International Symposium on Computer Architecture*, ISCA '00, pages 214–224, New York, NY, USA, 2000. ACM.

[179] Jan Reineke. *Caches in WCET Analysis*. PhD thesis, Universität des Saarlandes, November 2008.

[180] Jan Reineke and Daniel Grund. Relative competitive analysis of cache replacement policies. In *Proceedings of the 2008 ACM SIGPLAN-SIGBED Conference on Languages, Compilers, and Tools for Embedded Systems*, LCTES '08, pages 51–60, New York, NY, USA, 2008. ACM.

[181] Efraim Rotem, Alon Naveh, Avinash Ananthakrishnan, Eliezer Weissmann, and Doron Rajwan. Power-management architecture of the Intel microarchitecture code-named Sandy Bridge. *IEEE Micro*, 32(2):20–27, March 2012.

[182] B. Rountree, D.H. Ahn, B.R. de Supinski, D.K. Lowenthal, and M. Schulz. Beyond DVFS: A first look at performance under a hardware-enforced power bound. In *2012 IEEE 26th International Parallel and Distributed Processing Symposium Workshops PhD Forum (IPDPSW)*, pages 947–953, May 2012.

[183] Martino Ruggiero, Andrea Acquaviva, Davide Bertozzi, and Luca Benini. Application-specific power-aware workload allocation for voltage scalable MPSoC platforms. *International Conference on Computer Design (ICCD)*, pages 87–93, 2005.

[184] Frederick Ryckbosch, Stijn Polfliet, and Lieven Eeckhout. Trends in server energy proportionality. *Computer*, 44(9):69–72, September 2011.

[185] D. Sanchez and C. Kozyrakis. The ZCache: Decoupling ways and associativity. In *43rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 187–198, Dec 2010.

[186] 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.

[187] Daniel Sanchez, Luke Yen, Mark D. Hill, and Karthikeyan Sankaralingam. Implementing signatures for transactional memory. In *Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO 40, pages 123–133, Washington, DC, USA, 2007. IEEE Computer Society.

[188] Sandia National Laboratories. High performance conjugate gradients v2.1. `https://software.sandia.gov/hpcg/download.php`, January 2014.

[189] Ruchira Sasanka, Christopher J. Hughes, and Sarita V. Adve. Joint local and global hardware adaptations for energy. In *Proceedings of the 10th International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS X, pages 144–155, New York, NY, USA, 2002. ACM.

[190] Rathijit Sen and David A. Wood. Cache power budgeting for performance. Technical Report TR-1791, University of Wisconsin-Madison, April 2013.

[191] Rathijit Sen and David A. Wood. Reuse-based online models for caches. In *Proceedings of the ACM SIGMETRICS/International Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS '13, pages 279–292, New York, NY, USA, 2013. ACM.

[192] Rathijit Sen and David A. Wood. Electronic computer providing power/performance management, July 2016. US Patent 9383797.

[193] Joseph Sharkey, Alper Buyuktosunoglu, and Pradip Bose. Evaluating design tradeoffs in on-chip power management for CMPs. In *Proceedings of the 2007 International Symposium on Low Power Electronics and Design*, ISLPED '07, pages 44–49, New York, NY, USA, 2007. ACM.

[194] John P. Shen and Mikko H. Lipasti. *Modern Processor Design: Fundamentals of Superscalar Processors*. McGraw-Hill, first edition, July 2004.

[195] Timothy Sherwood, Erez Perelman, Greg Hamerly, Suleyman Sair, and Brad Calder. Discovering and exploiting program phases. *IEEE Micro*, 23(6):84–93, November 2003.

[196] Xudong Shi, Feiqi Su, Jih-Kwon Peir, Ye Xia, and Zhen Yang. Modeling and stack simulation of CMP cache capacity and accessibility. *IEEE Transactions on Parallel and Distributed Systems*, 20(12):1752–1763, 2009.

[197] T. Shyamkumar, N. Muralimanohar, J. H. Ahn, and N. P. Jouppi. CACTI 5.1. Technical Report HPL-2008-20, Hewlett Packard Labs, 2008.

[198] B. Sinharoy, R. N. Kalla, J. M. Tendler, R. J. Eickemeyer, and J. B. Joyner. POWER5 system microarchitecture. *IBM Journal of Research and Development*, 49(4/5):505–521, July 2005.

[199] Kevin Skadron, Mircea R. Stan, Wei Huang, Sivakumar Velusamy, Karthik Sankaranarayanan, and David Tarjan. Temperature-aware microarchitecture. In *Proceedings of the 30th Annual International Symposium on Computer Architecture*, ISCA '03, pages 2–13, New York, NY, USA, 2003. ACM.

[200] A. J. Smith. A comparative study of set associative memory mapping algorithms and their use for cache and main memory. *IEEE Transactions on Software Engineering*, SE-4(2):121–130, March 1978.

[201] Alan Jay Smith. Cache memories. *ACM Computing Surveys*, 14(3):473–530, September 1982.

[202] David C. Snowdon, Etienne Le Sueur, Stefan M. Petters, and Gernot Heiser. Koala: A platform for OS-level power management. In *Proceedings of the 4th ACM European Conference on Computer Systems*, EuroSys '09, pages 289–302, New York, NY, USA, 2009. ACM.

[203] Kimming So and Rudolph N. Rechtschaffen. Cache operations by MRU change. *IEEE Transactions on Computers*, 37(6):700–709, June 1988.

[204] S. Song, C. Y. Su, R. Ge, A. Vishnu, and K. W. Cameron. Iso-energy-efficiency: An approach to power-constrained parallel computation. In *IEEE International Parallel Distributed Processing Symposium (IPDPS)*, pages 128–139, May 2011.

[205] SPEC. SPECpower_ssj. `https://www.spec.org/power_ssj2008/`, 2008.

[206] SPEC. SPEC OMP2012. `http://www.spec.org/omp2012/`, 2012.

[207] V. Spiliopoulos, S. Kaxiras, and G. Keramidas. Green governors: A framework for continuously adaptive DVFS. In *International Green Computing Conference and Workshops (IGCC)*, pages 1–8, July 2011.

[208] Srinath Sridharan, Gagan Gupta, and Gurindar S. Sohi. Adaptive, efficient, parallel execution of parallel programs. In *Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '14, pages 169–180, New York, NY, USA, 2014. ACM.

[209] Viji Srinivasan, Edward S. Davidson, and Gary S. Tyson. A prefetch taxonomy. *IEEE Transactions on Computers*, 53(2):126–140, February 2004.

[210] Andreas Stiller. Green IT Cube: Hocheffizientes supercomputer-domizil eingeweiht. `http://www.heise.de/newsticker/meldung/Green-IT-Cube-Hocheffizientes-Supercomputer-Domizil-eingeweiht-3082605.html`, 2016.

[211] Harold S. Stone and Dominique Thiebaut. Footprints in the cache. In *Proceedings of the 1986 ACM SIGMETRICS Joint International Conference on Computer Performance Modelling, Measurement and Evaluation*, SIGMETRICS '86/PERFORMANCE '86, pages 4–8, New York, NY, USA, 1986. ACM.

[212] Jay E. Strum. Binomial matrices. *The Two-Year College Mathematics Journal*, 8(5):pp. 260–266, 1977.

[213] Balaji Subramaniam and Wu-chun Feng. Towards energy-proportional computing for enterprise-class server workloads. In *Proceedings of the 4th ACM/SPEC International Conference on Performance Engineering*, ICPE '13, pages 15–26, New York, NY, USA, 2013. ACM.

[214] G. E. Suh, L. Rudolph, and S. Devadas. Dynamic partitioning of shared cache memory. *Journal of Supercomputing*, 28(1):7–26, April 2004.

[215] G. Edward Suh, Srinivas Devadas, and Larry Rudolph. A new memory monitoring scheme for memory-aware scheduling and partitioning. In *Proceedings of the 8th International Symposium on High-Performance Computer Architecture*, HPCA '02, pages 117–128, Washington, DC, USA, 2002. IEEE Computer Society.

[216] K. T. Sundararajan, T. M. Jones, and N. Topham. Smart cache: A self adaptive cache architecture for energy efficiency. In *International Conference on Embedded Computer Systems (SAMOS)*, pages 41–50, July 2011.

[217] K. T. Sundararajan, V. Porpodas, T. M. Jones, N. P. Topham, and B. Franke. Cooperative partitioning: Energy-efficient cache partitioning for high-performance CMPs. In *18th IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 1–12, Feb 2012.

[218] Supermicro. SuperServer 5018D-MTF. `http://www.supermicro.com/products/system/1U/5018/SYS-5018D-MTF.cfm`.

[219] Masamichi Takagi and Kei Hiraki. Inter-reference gap distribution replacement: An improved replacement algorithm for set-associative caches. In *Proceedings of the 18th Annual International Conference on Supercomputing*, ICS '04, pages 20–30, New York, NY, USA, 2004. ACM.

[220] David K. Tam, Reza Azimi, Livio B. Soares, and Michael Stumm. RapidMRC: Approximating L2 miss rate curves on commodity systems for online optimizations. In *Proceedings of the 14th International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS XIV, pages 121–132, New York, NY, USA, 2009. ACM.

[221] Joel M. Tendler, Steve Dodson, Steve Fields, Hung Le, and Balaram Sinharoy. POWER4 system microarchitecture. *IBM Journal of Research and Development*, 46(1):5–25, January 2002.

[222] Niki C. Thornock and J. Kelly Flanagan. Facilitating level three cache studies using set sampling. In *Proceedings of the 32Nd Conference on Winter Simulation*, WSC '00, pages 471–479, San Diego, CA, USA, 2000. Society for Computer Simulation International.

[223] Georgios Varsamopoulos and Sandeep K. S. Gupta. Energy proportionality and the future: Metrics and directions. In *39th International Conference on Parallel Processing Workshops (ICPPW)*, pages 461–467, Sept 2010.

[224] Augusto Vega, Alper Buyuktosunoglu, Heather Hanson, Pradip Bose, and Srinivasan Ramani. Crank it up or dial it down: Coordinated multiprocessor frequency and folding control. In *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO-46, pages 210–221, New York, NY, USA, 2013. ACM.

[225] Alexander V. Veidenbaum, Weiyu Tang, Rajesh Gupta, Alexandru Nicolau, and Xiaomei Ji. Adapting cache line size to application behavior. In *Proceedings of the 13th International Conference on Supercomputing*, ICS '99, pages 145–154, New York, NY, USA, 1999. ACM.

[226] Ganesh Venkatesh, Jack Sampson, Nathan Goulding, Saturnino Garcia, Vladyslav Bryksin, Jose Lugo-Martinez, Steven Swanson, and Michael Bedford Taylor. Conservation cores: Reducing the energy of mature computations. In *Proceedings of the Fifteenth Edition of ASPLOS on Architectural Support for Programming Languages and Operating Systems*, ASPLOS XV, pages 205–218, New York, NY, USA, 2010. ACM.

[227] W. H. Wang, J.-L. Baer, and H. M. Levy. Organization and performance of a two-level virtual-real cache hierarchy. In *Proceedings of the 16th Annual International Symposium on Computer Architecture*, ISCA '89, pages 140–148, New York, NY, USA, 1989. ACM.

[228] Yasuko Watanabe, John D. Davis, and David A. Wood. WiDGET: Wisconsin decoupled grid execution tiles. In *Proceedings of the 37th Annual International Symposium on Computer Architecture*, ISCA '10, pages 2–13, New York, NY, USA, 2010. ACM.

[229] Mark Weiser, Brent Welch, Alan Demers, and Scott Shenker. Scheduling for reduced CPU energy. In *Proceedings of the 1st USENIX Conference on Operating Systems Design and Implementation*, OSDI '94, Berkeley, CA, USA, 1994. USENIX Association.

[230] Wikipedia. 80 Plus. `http://en.wikipedia.org/wiki/80_Plus`.

[231] D. Wong and M. Annavaram. Implications of high energy proportional servers on cluster-wide energy proportionality. In *HPCA*, pages 142–153, Feb 2014.

[232] Daniel Wong and Murali Annavaram. KnightShift: Scaling the energy proportionality wall through server-level heterogeneity. In *Proceedings of the 2012 45th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO-45, pages 119–130, Washington, DC, USA, 2012. IEEE Computer Society.

[233] Steven Cameron Woo, Moriyoshi Ohara, Evan Torrie, Jaswinder Pal Singh, and Anoop Gupta. The SPLASH-2 programs: Characterization and methodological considerations. In *Proceedings of the 22nd Annual International Symposium on Computer Architecture*, ISCA '95, pages 24–36, New York, NY, USA, 1995. ACM.

[234] Carole-Jean Wu, Aamer Jaleel, Will Hasenplaugh, Margaret Martonosi, Simon C. Steely, Jr., and Joel Emer. SHiP: Signature-based hit predictor for high performance caching. In *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO-44, pages 430–441, New York, NY, USA, 2011. ACM.

[235] Xiaoya Xiang, Chen Ding, Hao Luo, and Bin Bao. HOTL: A higher order theory of locality. In *Proceedings of the Eighteenth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '13, pages 343–356, New York, NY, USA, 2013. ACM.

[236] Yuejian Xie and Gabriel H. Loh. PIPP: Promotion/insertion pseudo-partitioning of multi-core shared caches. In *In Proceedings of the 36th International Symposium on Computer Architecture*, pages 174–183, 2009.

[237] Se-Hyun Yang, Babak Falsafi, Michael D. Powell, and T. N. Vijaykumar. Exploiting choice in resizable cache design to optimize deep-submicron processor energy-delay. In *Proceedings of the 8th International Symposium on High-Performance Computer Architecture*, HPCA '02, pages 151–161, Washington, DC, USA, 2002. IEEE Computer Society.

[238] Se-Hyun Yang, Michael D. Powell, Babak Falsafi, Kaushik Roy, and T. N. Vijaykumar. An integrated circuit/architecture approach to reducing leakage in deep-submicron high-performance I-caches. In *Proceedings of the Seventh International Symposium on High-Performance Computer Architecture (HPCA)*, pages 147–157, 2001.

[239] Doe Hyun Yoon, Min Kyu Jeong, and Mattan Erez. Adaptive granularity memory systems: A tradeoff between storage efficiency and throughput. In *Proceedings of the 38th Annual International Symposium on Computer Architecture*, ISCA '11, pages 295–306, New York, NY, USA, 2011. ACM.

[240] Chuanjun Zhang, Frank Vahid, and Walid Najjar. A highly configurable cache architecture for embedded systems. In *Proceedings of the 30th Annual International Symposium on Computer architecture*, ISCA '03, pages 136–146, New York, NY, USA, 2003. ACM.

[241] Kuangyu Zheng, Xiaodong Wang, Li Li, and Xiaorui Wang. Joint power optimization of data center network and servers with correlation analysis. In *Proceedings IEEE INFOCOM*, pages 2598–2606, April 2014.

[242] Yutao Zhong, Xipeng Shen, and Chen Ding. Program locality analysis using reuse distance. *ACM Transactions on Programming Languages and Systems*, 31(6):20:1–20:39, August 2009.