Thesis for The Degree of Doctor of Philosophy

Dynamic Management of Multi-Core Processor Resources to Improve Energy Efficiency under Quality-of-Service Constraints

Mehrzad Nejat

Division of Computer Engineering
Department of Computer Science & Engineering
Chalmers University of Technology
Gothenburg, Sweden, 2022

Dynamic Management of Multi-Core Processor Resources to Improve Energy Efficiency under Quality-of-Service Constraints

Mehrzad Nejat

Thesis supervisor:
Prof. Per Stenström, Chalmers University of Technology, Sweden

Thesis co-supervisors:
Prof. Miquel Pericàs, Chalmers University of Technology, Sweden
Dr. Madhavan Manivannan, Chalmers University of Technology, Sweden

Examiner:
Prof. Fredrik Dahlgren, Ericsson & Chalmers University of Technology, Sweden

Opponent:
Prof. José F. Martinez, Cornell University, United States

Grading Committee:
Prof. Magnus Jahre, Norwegian University of Science and Technology, Norway
Prof. Lars Lundberg, Blekinge Institute of Technology, Sweden
Prof. Carole-Jean Wu, Facebook & Arizona State University, United States

Chairman:
Prof. Fredrik Dahlgren, Ericsson & Chalmers University of Technology, Sweden

ISBN 978-91-7905-644-5
Doktorsavhandlingar vid Chalmers tekniska högskola, Ny serie nr 5110.
ISSN 0346-718X

Technical Report No 216D
Department of Computer Science & Engineering
Division of Computer Engineering
Chalmers University of Technology
Gothenburg, Sweden

This thesis has been prepared using LaTeX.
Printed by Chalmers Reproservice,
Gothenburg, Sweden 2022.

I like to dedicate this work to my wife Farzaneh, for her support and kindness during the difficult times.

What makes humans stand out in the world is curiosity to ask questions, persistence in seeking answers, and innovation in building solutions. But, what makes them so powerful in discovering and changing the world is ***cooperation*** *and* ***coordination****, with a common goal to build a better future for every one.*

Abstract

With the current technology trends, the number of computers and computation demand is increasing dramatically. In addition to different economic and environmental costs at a large scale, the operational time of battery-powered devices is dependent on how efficiently the computer processors consume energy.

Computer processors generally consist of several processing cores and a hierarchy of cache memory that includes both private and shared cache capacity among the cores. A resource management algorithm can adjust the configuration of different core and cache resources at regular intervals during run-time, according to the dynamic characteristics of the workload.

A typical resource management policy is to maximize performance, in terms of processing speed or throughput, without exceeding the power and thermal limits. However, this can lead to excessive energy expenditure since a higher performance does not necessarily increase the value of the outcome. For example, increasing the frame-rate of multi-media applications beyond a certain target will not improve user experience considerably. Therefore, applications should be associated with Quality-of-Service (QoS) targets. This way, the resource manager can search for configurations with minimum energy that does not violate the performance constraints of any application. To achieve this goal, we propose several resource management schemes as well as hardware and software techniques for performance and energy modeling, in three papers that constitute this thesis.

In the first paper, we demonstrate that, in many cases, independent management of resources such as per-core dynamic voltage-frequency scaling (DVFS) and cache partitioning fails to save a considerable energy without causing any performance degradation. Therefore, we present a coordinated resource management algorithm that saves considerable energy by exploring different combinations of resource allocations to all applications, at regular intervals during run-time. This scheme is based on simplified analytical performance and energy models and a multi-level reduction technique for reducing the dimensions of the multi-core configuration space.

In the second paper, we extend the coordinated resource management with dynamic adaptation of the core micro-architectural resources. This way, we include instruction- and memory-level parallelism, ILP and MLP, resp., in the resource trade-offs together with per-core DVFS and cache partitioning. This provides a powerful means to further improve energy savings. Additionally, to enable this scheme, we propose a hardware technique that improves the accuracy of performance and energy prediction for different core sizes and cache partitionings.

Finally, in the third paper, we demonstrate that substantial improvements in energy savings are possible by allowing short-term deviations from the baseline performance target. We measure these deviations by introducing a parameter called slack. Based on this, we present Cooperative Slack Management (CSM) that finds opportunities to generate slack at low energy cost and utilize it later to save more energy in the same or even other processor cores. This way, we also ensure that the performance consistently remains ahead of the baseline target in every core.

Keywords

Energy Efficiency, Quality-of-Service (QoS), Multi-core Resource Management, Cache Partitioning, Re-configurable Core Architecture, Dynamic Voltage-Frequency Scaling (DVFS)

Acknowledgment

I like to express my sincere gratitude to my main supervisor Prof. Per Stenström, not only for his significant and persistent support during my Ph.D, but also for everything I have learned from him. Similarly, I appreciate the support of my co-supervisors Prof. Miquel Pericàs and Dr. Madhavan Manivannan. I am also thankful to Prof. Fredrik Dahlgren who has been my examiner and Dr. Vassilis Papaefstathiou for his help during the first year of my PhD.

During this period of my life, I had the privilege to work alongside outstanding professors and colleagues such as Pedro, Ioannis, Lars, Risat, Sven, Petros, Albin, Ahsen, Nadja, Alexandra, Angelos, Evangelos, Dimitry, Parajith, Bhavi, Stavros, Fazeleh, Amir, Shirin, Hannaneh, Piyumal, Pirah, Jing, Monica, and many more. But, I specially appreciate many insightful discussions with Waqar in our office.

I like to emphasize that this journey was not possible without the support of my family. During the difficult days, I found calmness with my wife Farzaneh who has always cared for me. I am also grateful to my parents, parents in law, my brother, and my sisters in law. Without their sacrifices and encouragement, I could not achieve any success in my life.

This research has been funded by the European Research Council, under the MECCA project, contract number ERC-2013-AdG 340328, and the European Processor Initiative, under the project number 800928, and the Swedish Research Council (Vetenskapsrådet) under the Approximate Algorithms and Computing Systems Project.

List of Publications

Appended publications

This thesis is based on the following publications:

I Mehrzad Nejat, Madhavan Manivannan, Miquel Pericàs, Per Stenström "Coordinated Management of DVFS and Cache Partitioning under QoS Constraints to Save Energy in Multi-Core Systems"
Journal of Parallel and Distributed Computing (JPDC), 2020

II Mehrzad Nejat, Madhavan Manivannan, Miquel Pericàs, Per Stenström "Coordinated Management of Processor Configuration and Cache Partitioning to Optimize Energy under QoS Constraints"
The 34th IEEE International Parallel & Distributed Processing Symposium (IPDPS), 2020.

III Mehrzad Nejat, Madhavan Manivannan, Miquel Pericàs, Per Stenström "Cooperative Slack Management: Saving Energy of Multi-Core Processors by Trading Performance Slack Between QoS Constrained Applications"
ACM Transactions on Architecture and Code Optimization (TACO), 2022.

The papers will be referred to in the thesis using their Roman numerals.

Other publications

Paper I is an extended version of the following publication which is not included in this thesis:

[a] Mehrzad Nejat, Miquel Pericàs, Per Stenström "QoS-Driven Coordinated Management of Resources to Save Energy in Multicore Systems"
in IEEE International Parallel and Distributed Processing Symposium (IPDPS), 2019

Contents

Chapter 1

Introduction

With the current technology trends, the number of computers and the computation demands are increasing dramatically worldwide. The emergence of new applications such as autonomous vehicles and digital currencies, and the rapid expansion of existing application domains such as cloud computing or cyber-physical systems, are a few examples of these trends. Consequently, reducing the energy consumption per unit of computation is necessary for achieving sustainable development and avoiding severe economic and environmental problems. To better understand the scale of this issue, HiPEAC vision 2019 [1] mentions that according to some projections, "by 2030 ICT will consume the equivalent of half of the global electricity production of 2014". It also states that "Computing systems need to be orders of magnitude more energy efficient" to succeed in their expectations. On the other hand, as a growing number of computers are powered by batteries, the operational time and usability of these systems depend on how efficiently their processors consume energy.

Hence, this thesis targets the important and challenging goal of improving the energy efficiency of computers, ranging from battery-powered devices to high-performance data centers. In this chapter, the necessary background information is provided along with the statement of the thesis objective, an overview of the related work, the problem formulation, challenges, and a summary of the contributions of the thesis. Finally, the thesis organization is described at the end of this chapter.

1.1 Background

Today, a typical microprocessor is designed with a multi-core architecture consisting of several processing cores or processors. Each core contains a private cache memory hierarchy (usually two levels), while a third-level larger so called Last Level Cache (LLC) is shared among all cores. When the processor cannot find a particular data in the cache hierarchy, it issues an off-chip access to the main memory to collect that data. These accesses are a major source of energy consumption, especially for memory-intensive applications. On the other hand, processor cores are also responsible for a significant portion of energy consumption when executing different instructions. Therefore, this thesis focuses on reducing these two dominant energy components.

The energy of a processor core has a quadratic relationship to its voltage. For example, reducing the voltage by half can save 75% of the core energy. But, the operating frequency must be reduced accordingly due to an increased circuit delay. Most processors are capable of performing such a change during run-time, using a technique called Dynamic Voltage-Frequency Scaling (DVFS). Furthermore, the core energy consumption can also be reduced by deactivating sections of its micro-architectural components [2]. But, this may lower the performance by limiting the number of instructions and memory accesses that can be executed in parallel — a.k.a instruction- and memory-level parallelism, ILP & MLP, respectively.

To reduce the memory access energy, the number of cache misses should be minimized by utilizing the cache capacity efficiently. For example, providing a larger LLC share to a memory-intensive application, at the right time, can reduce the number of memory accesses significantly. But, this requires other applications to give up some of their allocated

LLC capacity. Therefore, such a decision cannot be taken without considering its effect on all applications in the system. Dynamic reallocation of LLC capacity among different applications can be performed via cache partitioning techniques such as Intel cache allocation technology [3].

The trade-offs mentioned above can be regulated by a resource management scheme that adapts the resource configurations according to the dynamic characteristics of the applications in the multi-core workload. The resource manager can be implemented as a lightweight software handler that is regularly invoked during run-time. At each invocation, it can collect information about dynamic program characteristics from hardware performance monitoring counters. Using this information, and based on the predetermined policies, the resource management algorithm can modify the resource configurations such as per-core DVFS, core sizes, and partitioning of the shared cache capacity, during run-time.

A typical objective in processor resource management schemes is to utilize the resources to maximize performance in terms of processing speed or throughput. However, in many applications, additional performance does not produce additional value all the time. A simple example is multi-media applications with a certain performance target in terms of frames-per-second rate. Increasing the processing speed beyond this target can only cause excessive energy expenditure without any considerable improvement in user experience. Additionally, there are scenarios when the value of energy-saving escalates. For instance, when the battery charge of a smartphone is low, or when the cost of electricity increases for a data-center, a controlled reduction in performance may be acceptable to enable additional energy savings.

Therefore, applications should be associated with QoS targets that specify an acceptable lower-bound on performance. This enables the resource manager to optimize the resource configurations with the objective to minimize energy while meeting the performance constraints for all applications. Furthermore, by adjusting the QoS targets, this approach allows controlled trade-offs between energy-saving and performance. For example, if an application can tolerate a 30% increase in its execution time, the resource manager may find opportunities to save more energy with a relaxed performance constraint. This can provide additional value, e.g., in terms of reduced cost of electricity or extended battery-life.

1.2 Objective

Based on the background provided in the previous section, we establish the objective of this thesis as follows:

> **In a multi-core processor that runs several independent applications, the goal of this thesis is to reduce the energy consumption of the system, as much as possible, while maintaining QoS, expressed as performance targets, for all applications.**

This objective is the fundamental target of the three papers – Paper I – III – this thesis is based on. These papers are summarized later in Chapter 3. In the next section, we provide a brief overview of the related work and explain the gap in the literature that is addressed by these papers.

1.3 Related Work

When a processor runs a particular program, the utilization of different resources varies dynamically according to the program characteristics. Therefore, several previous works such as [4–10] propose resource management schemes that adapt different resource configurations during run-time according to the QoS targets of a particular application. However, these works do not consider the problem of sharing a resource such as cache capacity between several QoS-constrained applications.

Resource management becomes more challenging when considering several applications that share the resources of a multi-core processor. In order to provide QoS guarantees, prior work such as [11] limit the number of QoS-constrained applications to a single one. Such pessimistic over-allocation leads to inefficient use of resources. Therefore, to improve system utilization without violating QoS, the following solution is proposed. Assuming a pool of best-effort applications with no specific performance requirements assumed, excess resources are dynamically reclaimed from the QoS-constrained application when they are not fully utilized. This way, system utilization is improved by running best-effort jobs as long as it does not affect the performance of the QoS-constrained application.

There are several other studies such as [12–14] that propose a similar approach which support multiple QoS-constrained applications instead of one. However, the approach taken in these works does not take energy consumption into account. As mentioned in Section 1.1, maximizing system performance — either in terms of throughput or utilization — can lead to excessive energy expenditure if applications are not associated with a QoS target. This is the case for best-effort applications. Furthermore, the improvements achieved with this approach depends on the presence of a sufficient number of best-effort applications that can tolerate unlimited performance degradation.

In another recent work [15], a reinforcement-learning mechanism is proposed that adapts core mapping and DVFS with the objective to reduce power consumption while meeting the QoS targets. However, there are three shortcomings in this approach. First, saving power can potentially contradict with saving energy. A resource configuration with lower power may consume higher energy if it leads to significantly longer execution time. Second, two influential processor resources, namely, partitioning of the shared cache capacity, and dynamic adaptation of the core micro-architectural components are not considered. The former has a substantial effect on the number of memory accesses, while the latter affects instruction- and memory-level parallelism, ILP & MLP, respectively. Thus, they have significant impact on both energy and performance. Third, solutions that are entirely based on machine-learning techniques are either too slow or too inaccurate when assuming no prior information about the processor workload, and when the size of the configuration space is too large. Alternatively, analytical models based on computer architecture knowledge can provide faster and more accurate predictions.

1.4 Problem Statement

As mentioned earlier, we consider a multi-core processor that runs a set of independent QoS-constrained applications. For simplicity, we assume the same number of applications as the number of cores. We envision an online resource management scheme that controls processor configurations including per-core DVFS and LLC partitioning at regular intervals during run-time. Additionally, we consider a simple adaptive core architecture that can dynamically switch between a few core sizes (small, medium, large). This can be done by adding logic for deactivating/reactivating sections of different micro-architectural components in each core.

We assume that a baseline resource configuration provides sufficient performance to meet the QoS targets of all applications that are scheduled to run on the processor. The baseline configuration can be, e.g., equal partitioning of LLC together with a set of core frequencies and sizes. Hence, we formulate the main research problem as follows:

> **How can the resource manager select a configuration at each invocation, that minimizes energy consumption without causing any performance degradation compared to the baseline configuration for all applications?**

This problem formulation makes the resource manager independent from applications. Regardless of what type of application runs on each core, it guarantees that the same performance, or higher, is delivered to every application according to the initial resource allocations. But, during run-time, resources can be re-distributed under the hood, based on the dynamic program characteristics. This way, the objective stated in Section 1.2 can be achieved.

However, the implementation of a resource management scheme that is capable of addressing the above problem is challenging. Here, we briefly explain some of these challenges at a high level.

First, at each invocation, the resource manager needs to know the effect of any reconfiguration on the performance of all applications as well as the system energy. Assuming no prior information about the applications and their dynamic run-time behavior, and considering the large number of potential combinations of resource configurations, making such a prediction is difficult. A prediction with low accuracy may lead to QoS violations and higher energy consumption and would contradict the stated objective of this thesis (Section 1.2).

Second, the overheads imposed by the resource manager, in terms of both execution time and energy, must be negligible. A large overhead can lead to additional energy consumption and performance degradation. Furthermore, it can limit the number of invocations of the resource manager during run-time. This may reduce the effectiveness of the resource manager in reacting to short-term variations in the workload behavior, and achieving the energy-saving objective. Therefore, it is essential to have an efficient algorithm for exploring a large multi-dimensional configuration space at each invocation of the resource manager. Furthermore, the overheads must remain negligible even for systems with a larger number of processing cores. In other words, the proposed scheme must be scalable.

Third, consider a scenario when several applications are competing for additional cache space. If no application can tolerate any performance degradation compared to their baseline resource setting, it may seem impossible to make any change in the system without violating the performance constraint of one application. This may prevent energy-saving, e.g., by lowering the core frequency or size, or a more efficient partitioning of cache.

1.5 Contributions

To address the challenges explained earlier and realize the objective stated for this thesis, three papers are published which make the following contributions:

- **Paper-I:**

    (1) An online resource management approach is presented that coordinates the control of on-chip cache space and per-core DVFS to save processor energy while respecting the QoS of every application in the workload.

    (2) To reduce the overheads and achieve scalability, a heuristic algorithm is proposed for finding the configuration that minimizes processor energy. A linear time complexity with respect to the number of cores is achieved by pruning the large multi-dimensional configuration space in several levels based on the estimations made by analytical performance and energy models.

    (3) The effect of coordinated and independent DVFS and cache partitioning is quantitatively studied for a range of different mixes of application categories and QoS targets. It is demonstrated how energy savings improve for different workloads when relaxing the performance targets for all, or a subset of applications in the workloads. Furthermore, the sensitivity of energy savings to the baseline settings is analyzed.

- **Paper-II:** This paper builds on the approach presented in Paper-I by considering an adaptive core architecture. Processor energy can be reduced further by extending the scope of the resource manager to include micro-architectural components of each core. The contributions of this paper can be summarized as follows:

    (1) Based on a systematic analysis of the resource trade-offs for different application categories, it is demonstrated in which workload scenarios the combination of core resizing with DVFS and cache partitioning can provide a considerably higher energy saving compared to the previous work (Paper-I).

    (2) A resource management scheme is presented that improves energy efficiency by dynamic adaptation of the core micro-architecture in coordination with per-core DVFS and LLC partitioning, while respecting the QoS constraints of all applications.

    (3) A hardware design is proposed that improves the accuracy of performance and energy modeling. This improvement is achieved through a heuristic mechanism that predicts MLP for a range of different core sizes and cache allocations.

- **Paper-III:** While Paper-I and Paper-II focus on energy optimization across the resource configuration space at each invocation of the resource manager, Paper-III extends the optimization by making trade-offs at different points in time. To enable such trade-offs we need to keep track of short-term variations in processing speed with a parameter, called *slack*. The contributions of this paper can be summarized as follows:

(1) Two important insights are presented: (a) The possibility to create slack at a lower energy cost compared to the energy-saving enabled by using it at a later point in time; and (b) the possibility to transfer slack from one QoS application to another, through the shared resource, that enables more energy-saving opportunities.

(2) Cooperative Slack Management (CSM) is introduced as a low-overhead resource management scheme to save substantial processor energy without affecting the QoS of any application in the workload.

(3) A sampling technique is designed to improve the modeling accuracy dynamically over run-time. This technique works with a hybrid approach that supports both active and passive sampling. The former enforces a predefined set of configurations to be sampled as soon as possible, while the latter passively collects samples without interfering with the resource manager.

(4) In addition to CSM, a local slack management (LSM) algorithm is designed that runs independently on each core. Over a range of different workload scenarios and baseline assumptions, it is demonstrated when the cooperative approach provides significant improvements and when an independent approach is good enough.

1.6 Organization

The rest of this thesis is organized as follows. Chapter 2 explains the experimental methodology and the simulation framework developed in this thesis. A summary of each paper is provided in Chapter 3. Chapter 4 presents the concluding remarks and the potential future work. Finally, the three papers are appended to the end of this thesis.

Chapter 2

Experimental Methodology

The experiments conducted in this thesis are based on simulations. A primary reason for this choice is that we evaluate architectural concepts that are not available in commercial products. This includes adaptive processor core micro-architectures and new hardware performance monitoring counters. However, available computer architecture simulators do not fulfill the requirements for our experiments out-of-the-box. This chapter starts by elaborating these shortcomings in Section 2.1. Next, the simulation framework designed to address these issues is described in Section 2.2. Finally, the evaluation metrics are discussed in Section 2.3.

2.1 Background

A fundamental challenge with computer architecture simulators is the prohibitively long simulation time, especially for full execution of benchmark applications such as SPEC-CPU-2006 [16] with reference input sets. To address this challenge, a technique called SimPoint analysis was proposed by Sherwood et al. [17]. SimPoint was inspired by the intuition that computer programs exhibit a dynamic behavior during run-time that can be categorized into several program phases. For example, consider a simple program that consists of two sequential loops. The first loop mainly contains memory instructions while the second loop is mostly computational instructions. Program behavior is considerably different between the execution of each loop. Therefore, this program can be divided into a *memory-intensive* phase followed by a *compute-intensive* phase.

SimPoint breaks the entire instruction stream of a program into consecutive slices with a fixed instruction count. It then uses a clustering algorithm to categorize slices with similar characteristics into several program phases. Finally, it selects one slice from each phase as the best representative of the average phase behavior. It also calculates the phase weights as the percentage of slices that belong to that phase. By simulating only the representative slices and using the phase weights, an estimate of the average program behavior can be derived in a fraction of time compared to simulating the entire program. For example, given the average cycles-per-instruction (CPI) count for each phase, a weighted average CPI can be derived for the entire program execution. Furthermore, the simulation of representative slices can be executed in parallel. Therefore, this technique can potentially reduce the simulation time by several orders of magnitude.

However, this simulation methodology assumes a fixed system configuration for the entire program execution. If a configuration such as core frequency or cache partitioning changes during the execution time, the simulation results for different program slices that belong to the same phase can be significantly different. Therefore, this methodology is not applicable for evaluating dynamic resource management schemes.

Nevertheless, the outputs of the SimPoint technique can be used as part of a two-level simulation framework that is capable of modeling the effect of dynamic changes in processor configuration at regular intervals during run-time. This is inspired by the idea proposed in [18] for improving the simulation of simultaneous multi-threading.

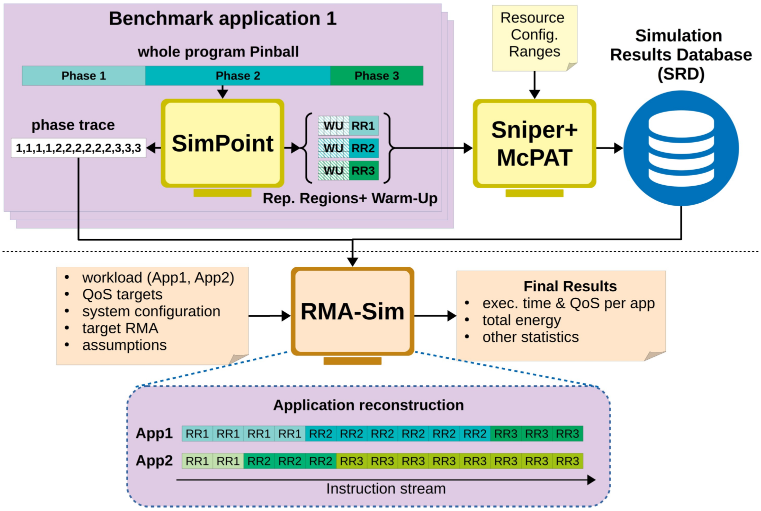

Figure 2.1: Overview of the simulation framework.

2.2 Simulation Framework

Figure 2.1 shows an overview of the simulation framework used in the papers in this thesis. It consists of two steps which are separated by a dashed line. The goal of the first step is to create a database that holds detailed architectural simulation results for a range of different processor configurations for the representative slice of each program phase. This is done using the SimPoint technique together with Sniper [19, 20] and McPAT [21] simulators that perform detailed performance and power/energy modeling, respectively. Once the database is ready for all the applications in the multi-core workload, it is used by an in-house developed resource-management-algorithm simulator (RMA-Sim) to perform different experiments.

This process starts by SimPoint which analyzes the entire instruction stream of a particular benchmark application. We refer to this instruction stream as the "*whole program pinball*"[1]. In the simplified example in Figure 2.1, each application consists of three continuous phases highlighted by different shades of green. SimPoint produces two outputs that are used by this framework. First, a representative region (RR)[2] is selected for each phase — as explained in Section 2.1. To improve the accuracy of the architectural simulations, each representative region is preceded with another region to warm-up (WU) the memory hierarchy. Second, a phase trace is produced that lists the phase numbers for every consecutive program slice.

One advantage of this framework is that every detailed architectural simulation in the first step can be performed independently in parallel. Therefore, given sufficient computing resources, the simulation results database for all benchmark applications can be generated in a short amount of time. For example, for program slices of 100-M instructions, Sniper+McPAT simulations can be done in less than 1-hour, even when using the most accurate performance model in Sniper, namely ROB. Another advantage is that the generated database can be used for an unlimited number of experiments. It can also be extended with processor configurations or new benchmark applications that are not simulated before.

To perform an experiment with the resource-management-algorithm simulator, we need

[1] A pinball [22] is a program instruction stream generated by Intel PinPlay tool [23]. The purpose of this tool is to have consistent and reproducible results by recording the non-deterministic events during a program execution. The pinballs used in the papers in this thesis are collected from the Sniper simulator website [24].

[2] The terms "region", "slice", and "interval" are used interchangeably in this thesis which refer to a section of program instruction-stream with a fixed number of instructions.

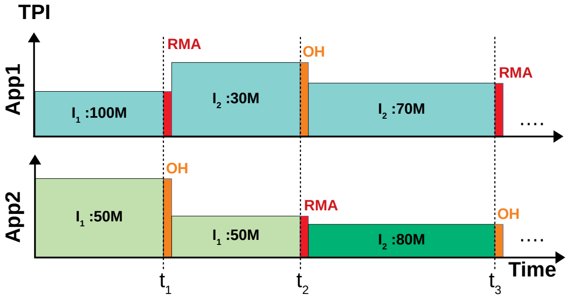

Figure 2.2: Run-time behavior of RMA-Sim.

to specify initial information such as the multi-core workload, QoS targets in terms of the baseline configuration, other system configuration parameters, the target resource-management-algorithm, etc. The simulator, which is an in-house software developed in Python, uses the phase trace of each application together with the database to regenerate a proxy of applications under the influence of the resource manager. This process is based on a simplifying assumption, derived from SimPoint, that every program slice belonging to a particular phase is identical to the representative slice of that phase. This is depicted in Figure 2.1 under “Application reconstruction”.

To further explain the mechanics of this simulation, we use the simplified example shown in Figure 2.2. In this example, the workload consists of two applications that run on two processing cores. The resource-management-algorithm is invoked on each core after executing 100-M instructions. The simulation starts with the baseline resource configuration for the first interval (I_1) of each application. The simulator determines the corresponding phase number for I_1 from the phase trace. Using the phase numbers and the baseline resource configuration, the average time-per-instruction (TPI) for both applications can be collected from the database. Therefore, it can estimate the time to the next event, i.e., the earliest end of a 100-M interval in any core. In this case, App1 with the lower time-per-instruction finishes its first interval (I_1) earlier at time t_1. At this point, App2 is still in the middle of its first interval and executed only 50-M instructions. The resource-management-algorithm is invoked by App-1 which decides a new resource configuration. After accounting the overheads and updating the simulation statistics for all cores, the time to the next event (t_2) is estimated in a similar fashion. The simulation continues until the finish criteria is met. The finish criteria can be, e.g., when every application has executed at least a predefined number of instructions or one round of execution. When applications in the workload reach the end of their execution before the finish criteria is met, they are restarted.

2.3 Evaluation Metrics

We consider two kinds of evaluations metrics in this thesis which are related to performance and energy consumption.

2.3.1 QoS

QoS is defined in this work as the performance achieved using the baseline resource configuration. Hence, one goal of the resource-management-algorithm is to deliver an average performance greater than or equal to the baseline configuration for each application. To evaluate this target, every experiment starts with an *Idle* resource manager that maintains the baseline configuration for the entire simulation. This way, the baseline execution time is measured for one round of execution of each application in the workload.

Next, another simulation is performed using the target resource-management-algorithm. During this simulation, the execution time of each round of every application is recorded. If the average execution time per one round of execution is smaller than or equal to the baseline results, we assume that the QoS target is achieved for this application. Otherwise, a QoS violation is reported. For example, if the average execution time is 5% longer compared to the baseline results, this value is reported as the QoS violation for this application.

2.3.2 Energy-Saving

During each simulation, two types of energy values are recorded. First, a *per-application* energy is recorded for each application that consists of the static and dynamic energy of the core (including private caches), the dynamic energy of the last-level-cache, the dynamic energy of the main memory, and the dynamic energy of Network-on-Chip. Second, an *application-independent* energy is recorded for the entire system which includes the static energy of the last-level-cache cache and Network-on-Chip. The static energy of the main memory is considered out-of-scope since it is not affected by the proposed resource management schemes.

The total instruction count varies significantly from one application to another in each workload. Therefore, to have a fair comparison, we measure the energy consumption for executing a fixed number of instructions on each core. This number corresponds to the largest application (in terms of instruction count) in the workload, or the entire experiment. Hence, the simulator adds the *per-application* energy components for executing the predefined number of instructions from each application to the *application-independent* system energy for the entire simulation time. The resulting value is then compared to the baseline simulation result (using Idle resource manager). For example, if the total energy value is 80% of the baseline result, a 20% energy-saving is reported by for the target RMA.

2.4 Summary

To evaluate the proposed resource management schemes, which include novel architectural concepts, the experimental methodology is based on simulations. However, the available simulation tools and methodologies such as Sniper and SimPoint do not fulfill the requirements for these experiments out-of-the-box.

Hence, a novel 2-step simulation framework is developed. The first step uses SimPoint, Sniper, and McPAT to generate a database that contains detailed architectural simulation results for a range of resource configurations for every program phase. Due to the high level of parallelism, this database can be generated in less than 1-hour, given sufficient computing resources. Later, it can be used by the second step of the framework to conduct an unlimited number of experiments. This step includes an in-house developed simulator that regenerates a proxy of benchmark applications while modeling the effect of different resource-management-algorithms during run-time. The computational demand and the execution time of this step is dramatically lower compared to architectural simulations. Therefore, this methodology enables a large number of experiments in a reasonable time.

Chapter 3

Summary of papers

3.1 Paper-I

3.1.1 Background

To reduce the energy consumption of multi-core processors, hardware resources can be throttled to deliver sufficient performance without degrading user experience. Therefore, applications must be associated with QoS targets that define a minimum constraint on performance. These constraints can be used by a resource management scheme with the objective to optimize processor energy.

Processor resources such as cache and voltage-frequency (VF) are typically controlled independently. For example, in previous work [7–9] the processor VF is controlled to meet a specific performance target; while in other work such as [25] the shared Last Level Cache (LLC) is partitioned to minimize the total number of cache misses. Such independent resource management approaches are not effective in a system with performance constraints on all applications. Imagine a scenario when the performance of every application in the workload is sensitive to their allocated shares of LLC capacity. An independent LLC controller cannot change the LLC partitioning and guarantee the same performance for every application in the workload at the same time. Similarly, the VF of processor cores cannot be reduced without causing any performance degradation. Hence, the baseline resource configuration cannot be changed due to performance constraints which prevents energy saving. Alternatively, if the processor resources are controlled in a coordinated fashion for all applications, the resource manager can explore a multi-dimensional configuration space. Therefore, it may find other resource combinations that provide at least the same performance for every application, but at a lower energy consumption.

3.1.2 Problem Statement

Paper-I attempts to answer the following question:

> **Assuming a baseline configuration of processor resources, including the core VF and the partitioning of LLC capacity, provides sufficient performance to meet the QoS targets of all applications in a multi-programmed workload; how can a resource manager dynamically find other resource configurations at run-time that minimize processor energy consumption while meeting the performance targets of all applications?**

3.1.3 Proposed approach

To address the above question, the approach presented in this paper is to design a coordinated Resource Management Algorithm (RMA) that combines the control of per-core Dynamic Voltage-Frequency Scaling (DVFS) and partitioning of the LLC. Such RMA requires a means to predict the performance and energy consumption of all applications across a

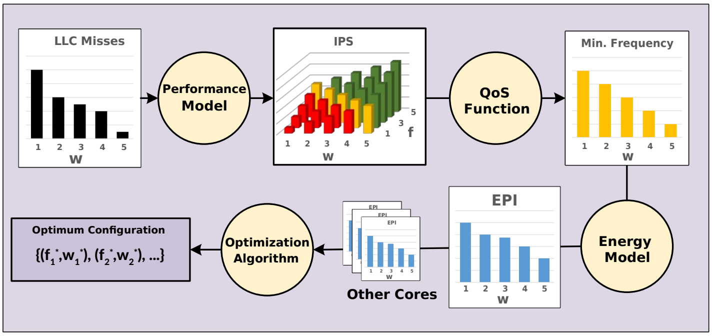

Figure 3.1: Overview of the proposed RMA (Figure 3 from *Paper-I*).

large multi-dimensional resource configuration space. This operation must be performed at regular intervals during run-time to react to frequent program phase changes in the workload. Therefore, the imposed overheads must remain sufficiently small, even when scaling to a larger number of cores. Furthermore, the RMA must work with the assumption of no prior knowledge about the applications.

Figure 3.1 shows an overview of the proposed RMA. The RMA that is implemented as a light-weight software handler is invoked on each core at regular intervals after executing a fixed number of instructions. It starts by collecting statistics from hardware Performance Monitoring Counters (PMC) and the Auxiliary Tag Directory (ATD) [25]. ATD is a well-known hardware extension that emulates the operation of the tag directory of the main cache. It provides a prediction of the number of cache misses as a function of the number of allocated cache ways (w), called cache-miss profile. A simple analytical performance model uses this cache-miss profile together with other PMC statistics to estimate instructions-per-second (IPS) as a function of w and core frequency (f). At this point, most of the resource configuration space can be pruned based on the QoS constraint as follows. For every w, a minimum frequency ($f_{min}(w)$) is found with IPS greater than or equal to the baseline configuration. This is depicted by the yellow bars in Figure 3.1. Next, using another simple analytical model, energy-per-instruction (EPI) is evaluated for every pair of w and $f_{min}(w)$. This value captures the energy of core, cache hierarchy, and the main memory that is associated with the execution of each application.

Once an energy curve is available for all cores, they are passed to the global optimization algorithm. This algorithm recursively reduces each pair of curves into one, until the optimum LLC allocation ($\{w_j^*\}$) is found that minimizes the sum of energy values over all cores ($\sum_j \mathrm{EPI}_j(w_j)$), such that the sum of allocations ($\sum_j w_j$) remains equal to the total number of ways in LLC. Finally, the new partitioning is applied to LLC and the per-core DVFS is set according to $f_{min,j}(w_j^*)$ for each core j.

3.1.4 Contributions

The main contributions of this paper are threefold:

(1) An online resource management approach is presented that coordinates the control of on-chip cache space and per-core DVFS to save processor energy while respecting the QoS of every application in the workload.

(2) To reduce the overheads and achieve scalability, a heuristic algorithm is proposed for finding the configuration that minimizes processor energy. A linear time complexity with respect to the number of cores is achieved by pruning the large multi-dimensional configuration space in several levels based on the estimations made by analytical performance and energy models.

(3) The effect of coordinated and independent DVFS and cache partitioning is quantitatively studied for a range of different mixes of application categories and QoS targets. It is demonstrated how energy savings improves for different workloads when relaxing the performance targets for all, or a subset of applications in the workloads. Furthermore, the sensitivity of energy savings to the baseline settings is analyzed.

3.1.5 Summary of results

To evaluate the proposed RMA, SPEC-CPU-2006 [16] benchmarks are categorized based on two criteria: Memory intensity and cache sensitivity. These criteria help to analyze the resource trade-offs in the system. For example, the effect of core frequency scaling is considerably different between an application that is dominated by memory accesses, and another application that is dominated by compute instructions. On the other hand, the effect of cache size on the number of memory accesses is dependent on memory access patterns. For instance, the number of memory accesses does not necessarily reduce when increasing the allocated cache space to a memory-intensive application. More specifically, we categorize an application as *memory-intensive* if the number of Misses Per Kilo Instructions (MPKI) on the baseline LLC allocation is greater than a threshold. Otherwise it is counted as *compute-intensive*. On the other hand, if the variation in MPKI for different LLC allocations around the baseline is above another threshold, the application is considered as *cache-sensitive*, otherwise *cache-insensitive*.

Therefore, applications can be categorized into four types. For example, a memory-intensive and cache-sensitive application benefits significantly from a larger cache allocation and its core frequency can be lowered to save energy. But, the performance of a compute-intensive and cache-insensitive application is mostly dependent on the core frequency, while it can give up its share of cache to other applications. Hence, several 4-core and 8-core multi-programmed workloads are generated based on different combinations of these categories. These workloads are simulated using the framework described in Chapter 2 to perform a quantitative analysis of these trade-offs in a range of different workload scenarios.

In the first experiment, energy savings are evaluated for each workload using the proposed combined RMA along with an RMA that controls only LLC partitioning. An RMA that controls only DVFS cannot save energy without degrading the performance. According to the experimental results, up to 18% and 14% of system energy can be saved using the Combined RMA in 4-core and 8-core systems, respectively. The average energy saving is 6% in both cases. In comparison, the Partitioning RMA can save only 1% and 2% on average in 4-core and 8-core systems, respectively. The combined RMA is most effective in the majority of workloads that include a cache sensitive application. In a few cases where all the applications are cache insensitive, there is even a small increase in system energy due to a limited modeling accuracy.

To evaluate the effect of modeling error, perfect models with no prediction error are used in a different experiment. With these models, the combined RMA saves on average 8% of system energy which is very close to the realistic results with the presented analytical models. In 13 cases out of 80 applications in the 4-core workloads, the modelling error leads to a QoS violation i.e. average execution time longer than the baseline [1]. The average value of these violations is 3% with a maximum of 9%. Regarding the 8-core results, 15 violations occurred out of 80 applications with an average and maximum value of 3% and 7%, respectively.

The energy savings can be further improved if users can tolerate a bounded reduction in performance. To evaluate this trade-off, in the next experiment, the performance constraints are gradually relaxed up to 80% longer execution time compared to the baseline. According to this experiment that uses perfect models, energy savings with the proposed scheme can improve up to 29% and on average 17% with only a limited relaxation of the QoS target (around 40% longer execution time). Furthermore, the energy savings are evaluated and analyzed in several scenarios where the QoS target is relaxed only for a subset of the workload. The sensitivity of energy savings to the choice of the baseline VF is also studied.

The overhead of the proposed RMA is evaluated based on a software implementation in C language. The number of executed instructions are measured to be less than 40K which is 0.04% of an execution interval with 100M instructions. The overheads imposed for collecting the required statistics and applying the new resource settings are presented and analyzed in details in this paper.

3.1.6 Conclusion

The experimental results confirm that the proposed RMA provides an effective solution for the main research problem, i.e., saving processor energy without causing any performance degradation for any application. Using simple analytical models and a heuristic algorithm, the RMA can assess the performance of every application and the processor energy in

[1] Values below 1% are considered negligible.

a large configuration space, while imposing negligible run time overhead. Therefore, it can dynamically adapt to the program phase changes in the workload and find the most energy-efficient resource configuration that meets the QoS targets of all applications, at each invocation during run-time.

3.2 Paper-II

3.2.1 Background

The previous work (Paper-I) proposes a multi-core resource management approach that coordinates per-core DVFS and LLC partitioning. It shows that such coordination between processor resources can enable energy savings without causing a performance degradation for any application in the workload. However, the scope for energy saving is limited when considering only DVFS and cache partitioning as the control knobs. For example, imagine a scenario when the LLC share of one application is reduced in favor of another application that benefits more from the LLC capacity. To avoid a performance degradation in the first application, the resource manager may need to increase the VF of the corresponding core at a quadratic energy cost. Therefore a considerable portion of the energy-savings achieved in the rest of the system is canceled by this energy cost. In fact, scaling the core VF has no effect on the stall time due to additional memory accesses. Therefore, DVFS becomes less effective as the number of cache misses increases.

Alternatively, if the resource manager has control over the core micro-architectural components to increase the issue width and instruction window length, i.e., the size of the reorder buffer, the required performance improvement can be achieved via parallelism. Unlike DVFS, increasing instruction-level-parallelism (ILP) can lead to a higher memory-level-parallelism (MLP) which may reduce the memory stall time. Furthermore, this performance improvement can be achieved at a lower energy cost compared to DVFS.

3.2.2 Problem Statement

This paper attempts to answer the following question:

> **Assuming that a multi-core processor resource manager has control over the size of the core micro-architectural components as well as per-core DVFS and partitioning of the shared LLC; how can it dynamically find trade-offs between these resources at run-time that minimizes processor energy consumption without degrading the performance of any application?**

3.2.3 Proposed approach

This paper addresses the above problem in two steps. First, it systematically analyzes the resource trade-offs in a range of workload mixes. Second, it proposes a resource management scheme that can select between a few sizes for each processing core (small, medium, large). This can be done with minimal overhead by adding some logic to activate/deactivate sections of micro-architectural components such as issue-width, reorder buffer, reservation stations, and load/store queues. The resource manager dynamically controls the size of each processing core in coordination with per-core DVFS and LLC partitioning across all applications in the workload. Compared to Paper-I, the addition of the third control parameter, i.e., the core size, imposes new challenges. The simplifying assumption of constant MLP in the analytical performance and energy models in Paper-I is no longer valid. Therefore, a means to predict the change in MLP over different core sizes and cache allocations is required. Furthermore, the imposed overheads must remain negligible with the addition of another dimension in the resource configuration space for each core.

Figure 3.2 shows an overview of the proposed resource manager. The resource-management-algorithm is invoked regularly on each core after executing a fixed number of instructions. It starts by collecting statistics of the past execution interval from hardware PMCs. These statistics are used by simple analytical models to predict the performance and energy consumption associated with the execution of a particular application as a function of resource configuration, including the core size (c), frequency (f), and the number of allocated ways (w) in LLC.

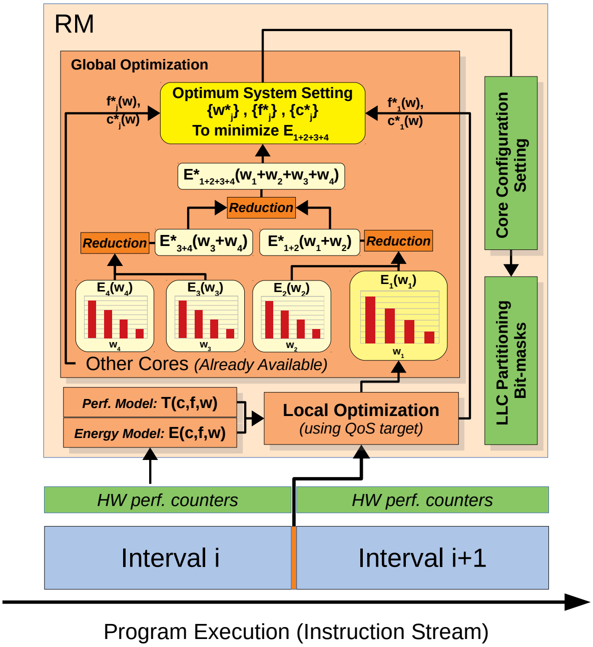

Figure 3.2: Overview of the proposed resource manager (Figure 3 from Paper-II).

To search for the optimal processor configuration in an efficient way, the local optimization algorithm removes two dimensions from the configuration space of each core as follows. For every possible w, a combination of c and f is found that minimizes the energy without lowering the performance compared to the baseline configuration. This step results in a one-dimensional energy curve ($E(w)$) for the core that invoked the resource manager. In the next step, the global optimization algorithm, which contains the energy curves of all cores ($E_j(w_j)$ for each core j), finds an optimum distribution of LLC ways ($\{w_j^*\}$) that minimizes the sum of energy values, such that $\sum_j w_j$ remains equal to the total number of ways in LLC. The approach used in the last step is similar to the global optimization algorithm in Paper-I. It recursively reduces each pair of energy curves into one, until the optimum distribution is found. A main advantage of this approach is its linear time complexity when scaling the number of cores. Once the optimum LLC partitioning ($\{w_j^*\}$) is found, the corresponding frequency and size of each core ($\{f_j^*\}$ and $\{c_j^*\}$) can be derived easily from the outcome of the local optimization.

As mentioned earlier, the analytical models in this work must be capable of predicting the effect of MLP variation for different values of c and w[2]. Therefore, a new hardware PMC is designed in this work based on the the Auxiliary Tag Directory (ATD) [25]. The original ATD has a counter for each w that counts the memory accesses that are predicted to miss given a cache allocation of w ways. This provides no information regarding MLP which is necessary when considering an adaptive core architecture. A variation in MLP can significantly affect performance since the cache misses that overlap with a leading miss do not contribute to the memory stall time [26, 27]. Hence, the proposed hardware extension contains different counters for each possible c and w that use a heuristic mechanism to detect and ignore the predicted cache misses that are overlapping with an earlier miss. The overhead of this hardware extension is estimated to be less than 300 bytes per core.

[2] Changing the core frequency has no effect MLP.

3.2.4 Contributions

This paper makes the following contributions:

(1) Based on a systematic analysis of the resource trade-offs for different application categories, it is demonstrated in which workload scenarios the combination of core resizing with DVFS and cache partitioning can provide a considerably higher energy saving compared to the previous work (Paper-I).

(2) A resource management scheme is presented that improves energy efficiency by dynamic adaptation of the core micro-architecture in coordination with per-core DVFS and LLC partitioning, while respecting the QoS constraints of all applications.

(3) A hardware design is proposed that improves the accuracy of performance and energy modeling. This improvement is achieved through a heuristic mechanism that predicts MLP for a range of different core sizes and cache allocations.

3.2.5 Summary of results

In order to analyze the resource trade-offs between different applications under QoS constraints, the SPEC CPU2006 benchmarks [16] are categorised based on two attributes: *Cache Sensitivity* and *Parallelism Sensitivity*. Similar to Paper-I, an application is counted as *cache-sensitive (CS)* if the variation in MPKI for different cache allocations around the baseline is above a specific threshold. Otherwise, it is counted as *cache-insensitive (CI)*. On the other hand, if a change in core size leads to a certain variation in MLP, the application is counted as *parallelism-sensitive (PS)*, otherwise *parallelism-insensitive (PI)*. This leads to four different categories (CS-PS, CS-PI, CI-PS, and CI-PI). These categories help in better understanding and analyzing the resource trade-offs in different workload scenarios. We analyze a simple case of a two-core system that runs a two-application multi-programmed workload. In this case, there are 16 possible mixes of application categories. For all possible mixes, the resource trade-offs are analyzed for three resource management schemes: RM1 that controls only the LLC partitioning; RM2 that coordinately controls per-core DVFS and LLC partitioning according to Paper-I; and RM3 (the proposed scheme in this work) that controls the size and VF setting of each core along with LLC partitioning. RM1 is not effective in most of the cases. But, when comparing RM2 with RM3, four interesting scenarios are detected:

(1) RM3 considerably improves the energy savings compared to RM2.

(2) The energy savings are comparable with both RM3 and RM2.

(3) Only RM3 can save a considerable amount of energy while RM2 is ineffective.

(4) Both RM3 and RM2 are ineffective.

According to this analysis, in 12 out of 16 possible mixes, the proposed scheme (RM3) substantially improves the energy savings. The resource management schemes are evaluated on several 4-core and 8-core workloads created for each scenario. The simulation framework described in Chapter 2 is used to run the experiments.

The experimental results show the same trend as expected from the analysis. In Scenario-1, the proposed RM3 saves up to 17.6% and on average 14% of system energy. The energy savings with RM3 are up to 60% larger compared to RM2 in this scenario. In Scenario-2, the energy savings are up to 10% and on average 5% while the results are similar with both RM2 and RM3. RM3 can save up to 11% and on average 8.5% of system energy in Scenario-3 while RM2 is not very effective. Finally, both RM2 and RM3 are not effective in Scenario-4.

Both the energy savings and QoS results can be affected by modeling error. In the next experiment, the proposed modeling technique (Model-3) based on the new hardware support is evaluated against two simple models: Model-2 that assumes constant MLP according to Paper-I and Model-1 that estimates total memory stall time as the product of the total number of cache misses and average latency of a single memory access. A comprehensive analysis is performed to evaluate the probability of QoS violations at each program interval (100M instructions). According to this analysis, Model-3 has 3% probability of QoS violation which is 32% and 46% smaller compared to Model-2 and Model-1, respectively. Furthermore, Model-3 substantially improves the expected value and standard deviation of violations by 49% and 26%, respectively, compared to Model-2. The experimental results on energy savings also shows a considerable improvement with Model-3. The weighted average energy savings is 10% with Model-3 compared to 7% and 5% with Model-2 and Model-1, respectively.

To evaluate the overheads imposed by RM3, the number of executed instructions are measured for a software implementation in C language. The resulting values are 18K, 40K, and 67K instructions for systems with 2, 4, and 8 cores, respectively. These values are less than 0.1% of a 100M instruction interval. The overheads imposed by other components of the system are also analyzed in details.

3.2.6 Conclusion

The experimental results confirm that the proposed scheme provides an effective solution for the main research problem, i.e., saving processor energy without causing any performance degradation for any application. It uses a novel hardware technique to evaluate performance and energy for a wide range of resource configurations in a negligible time with improved accuracy. Therefore, it can dynamically exploit resource trade-offs based on ILP/MLP to further reduce system energy under per-application performance constraints.

3.3 Paper-III

3.3.1 Background

The previous work (Paper-I and Paper-II) shows that by coordinating the control of multiple resources, including dynamic core resizing, per-core DVFS, and LLC partitioning, the resource manager can explore a multi-dimensional configuration space that expands to include all applications in the workload. Therefore, it can find resource trade-offs in this enlarged configuration space that reduce processor energy without lowering the performance of any application. This performance constraint is enforced at every invocation of the resource-management-algorithm during run-time.

However, it is not really necessary to maintain a constant performance, e.g., in terms of instructions-per-second (IPS), at all points in time, to satisfy the QoS targets of applications. A QoS-constrained application requires to finish a certain computation task within a particular time frame, which we call a *QoS window*. A key insight driving this paper is that allowing short-term variations in the performance target, within each QoS window, enables new opportunities to substantially reduce processor energy. In order to keep track of the deviation from the baseline performance target, a parameter called slack is defined. Slack can be generated or consumed for a particular application by raising or lowering the performance compared to the baseline target, respectively.

For each application, the cost of generating slack and the benefit of consuming slack, in terms of energy, is variable depending on the dynamic characteristics of the multi-program workload running on the multi-core processor. For example, when the LLC share of a memory-intensive application is increased, it enjoys a performance boost that generates slack at no energy cost. In fact, the energy is reduced due to fewer memory accesses. But, it can generate even more slack by raising the core VF at the same time[3], which imposes an energy cost.

Such cost may be justified as it can enable a larger energy saving in the future. Imagine that, in the same example scenario, another application enters a memory-intensive phase which leads to contention in LLC. It may be possible to find a more efficient LLC partitioning that significantly reduces the total number of memory accesses and the processor energy. But, this partitioning is prohibited if the baseline performance constraints are enforced for all applications. In this case, the accumulated slack from the previous resource management intervals can change the game by creating a safe leeway that allows the more efficient configuration. The resource manager insures that no application experiences a performance degradation by keeping the slack deposit in a safe range, instead of maintaining a fixed processing speed at all times.

[3] In fact, the coordinated resource manager reduces the core VF, as much as possible, to turn the potential performance boost from cache partitioning into energy-saving. But, for generating additional slack, it can select a relatively higher VF setting compared to the minimum acceptable VF.

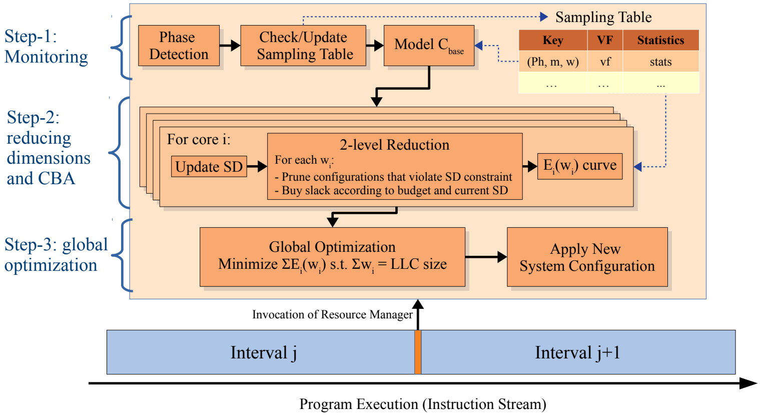
Step-1: Monitoring
Step-2: reducing dimensions and CBA
Step-3: global optimization
Phase Detection
Check/Update Sampling Table
Model C_{base}
Sampling Table
Key
VF
Statistics
(Ph, m, w)
vf
stats
…
…
...
For core i:
Update SD
2-level Reduction
For each w_i:
- Prune configurations that violate SD constraint
- Buy slack according to budget and current SD
$E_i(w_i)$ curve
Global Optimization
Minimize $\Sigma E_i(w_i)$ s.t. Σw_i = LLC size
Apply New System Configuration
Invocation of Resource Manager
Interval j
Interval j+1
Program Execution (Instruction Stream)

Figure 3.3: Overview of the proposed resource manager (Figure 2b from Paper-III).

3.3.2 Problem Statement

This paper attempts to answer the following question:

> **Assuming a resource manager that keeps track of short-term deviations from the baseline performance target of each application using a parameter called slack; how can it dynamically find slack trade-offs, across different applications at different points in time, that reduce processor energy while continuously keeping the slack deposit of every application in a safe range that meets the QoS targets?**

3.3.3 Proposed approach

To address the above question, this paper first explains two key insights by analyzing motivational data for a simplified educational example. As a first insight, it demonstrates that in certain conditions, slack can be generated at a relatively low energy cost. It also shows other conditions in which slack can be consumed to save a larger amount of energy. As a second insight, it shows the possibility to transfer slack from one application to another through the shared resource, i.e., LLC. By extending the slack trade-offs to include all applications, the opportunities to save processor energy increases dramatically.

Based on these insights, a new resource management scheme called **Cooperative Slack Management (CSM)** is designed. An overview of this scheme is presented in Figure 3.3. Similar to Paper-I and Paper-II, the resource manager is invoked on each core after executing a fixed number of instructions. The operation of the resource-management-algorithm can be divided into three steps. First, it collects the required statistics of the past interval and performs the initial tasks that enable the analytical performance and energy models. The second step starts by updating the slack deposit (SD) for each core i. Based on the updated SD, it evaluates slack trade-offs for every possible LLC allocation to this core (w_i) while reducing the dimensions of the optimization problem in two levels. This step leads to a one dimensional energy curve ($E_i(w_i)$) for each core i. Finally, in the third step, a global optimization algorithm finds an optimum distribution of LLC ways that minimizes the sum of energy values for all cores. The global optimization algorithm is similar to the one used in Paper-I and Paper-II.

Compared to the previous work, this paper requires a higher modeling accuracy, particularly for the baseline configuration. A modeling error on the performance of the baseline configuration propagates to the slack calculation and accumulates in the SD value. Therefore, it can lead to a large QoS violation; especially since the resource manager attempts to use as much slack as possible to improve energy savings. To alleviate this problem, a dynamic sampling technique is proposed in this work that improves the modeling accuracy during run-time. In a hybrid approach, it can *actively* enforce selected configurations to be sampled as soon as possible, as well as *passively* collect samples based on the outcome the resource manager decisions. Hence, the sampling mechanism can be tuned to provide a relatively higher accuracy for selected configurations, such as the baseline, while imposing minimal interference with the normal operation of the resource manager.

3.3.4 Contributions

The main contributions of this paper can be summarized as follows:

(1) Two important insights are presented: (a) The possibility to create slack at a lower energy cost compared to the energy-saving enabled by using it at later point in time; and (b) the possibility to transfer slack from one QoS application to another, through the shared resource, that enables more energy-saving opportunities.

(2) Cooperative Slack Management (CSM) is introduced as a low-overhead resource management scheme to save substantial processor energy without affecting the QoS of any application in the workload.

(3) A sampling technique is designed to improve the modeling accuracy dynamically over run-time. This technique works with a hybrid approach that supports both active and passive sampling. The former enforces a predefined set of configurations to be sampled as soon as possible, while the latter passively collects samples without interfering with the resource manager.

(4) In addition to CSM, a local slack management (LSM) algorithm is designed that runs independently on each core. Over a range of different workload scenarios and baseline assumptions, it is demonstrated when the cooperative approach provides significant improvements and when an independent approach is good enough.

3.3.5 Summary of results

To evaluate the proposed approach, two workload scenarios are studied. An interesting scenario that demonstrates the advantage of CSM is when there is contention in LLC. In such a scenario, the applications in the workload are sensitive to their allocated cache capacity such that a change in the baseline cache allocation can cause a considerable change in the number of cache misses. More specifically, we categorize an application as *cache-sensitive* if the average MPKI on the baseline LLC allocation is above a certain threshold and changing the LLC share around the baseline allocation leads to a large enough change in MPKI.

Hence, in Scenario-1 random mixes of cache-sensitive applications from the SPEC-CPU-2006 benchmarks [16] are selected to create several multi-programmed workloads. Additionally, to provide a more comprehensive evaluation, the workloads in Scenario-2 are created by randomly selecting any application from the benchmark suite.

As a reference for comparison, two other resource management schemes are simulated along with CSM using the framework explained in Chapter 2. The first scheme is the resource manager presented in Paper-I, which is referred to as "*Coordinated Core configuration and Cache Partitioning*" (C3P). The second scheme is an extended version of C3P that independently performs Local Slack Management (LSM) for each core.

In the first set of experiments, three baseline configurations are studied. First, to evaluate a high-performance target, the baseline is set to the largest core size and the highest VF level. While C3P and C3P+LSM are effective only for a few workloads in Scenario-2, CSM saves up to 35% and on average 16% (Scenario-1) and 10% (Scenario-2) energy. For comparison, the average values with C3P+LSM are only 1% and 6%, respectively. If we eliminate the effect of modeling error, the average energy savings with CSM increase to 22% (Scenario-1) and 17% (Scenario-2). The second baseline is a mid-range performance target based on a medium core size and a mid-range VF. In this case, C3P and C3P+LSM are also effective in both workload scenarios as they can use a larger core size to achieve a performance boost when needed. Consequently, the energy saving results are, on average, comparable for the three resource management schemes: 17% (C3P), 20% (C3P+LSM), and 20% (CSM) for Scenario-1, and 13% (C3P), 14% (C3P+LSM), and 12.5% (CSM) for Scenario-2. Finally, for the third baseline configuration, a fixed core architecture is assumed along with highest core VF. Similar to the first baseline, CSM shows significant improvement compared to C3P and C3P+LSM, in this case. It can save up to 36.5% and on average 22% (Scenario-1) and 10%

(Scenario-2) while C3P+LSM saves on average only 1% (Scenario-1) and 6.5% (Scenario-2). When using idealistic models, CSM can save up to 41% and on average 25% (Scenario-1) and 17% (Scenario-2).

While the first set of experiments were conducted with 4-core workloads, the second set of experiments evaluates the scalability of CSM to larger number of cores. Therefore, experiments similar to the first baseline configuration (high-performance) were performed with 8-core and 16-core workloads in the same two scenarios. The results for the three resource management schemes show a consistent overall trend to the 4-core simulations.

Finally, in the last set of experiments, sensitivity of CSM to different parameters is evaluated. The CSM algorithm contains three constant parameters that determine a lower (ϵ) and upper (SD_{thr}) bound SD and a budget that limits the energy cost for generating additional slack (E_{budget}). These parameters can be used to adjust the behavior of CSM for different systems. While the experiments show negligible sensitivity to the value of SD_{thr}, energy savings improve consistently when the value of ϵ is reduced. When changing ϵ from 0 to -5%, the average energy savings increase from 20% to 28.5% in Scenario-1 and from 14% to 23% in Scenario-2. This change is equivalent to relaxing the performance target to allow QoS violations of up to 5%, i.e., extending the execution time deadline to 105% of the baseline. On the other hand, the sensitivity to E_{budget} does not show a consistent trend. It shows a sweet-spot around $E_{budget} = 6\%$[4] that saves, on average, 22% (Scenario-1) and 17% (Scenario-2) energy. The reason for such behavior is the presence of two contradictory effects. On the one hand, raising E_{budget} leads to a larger SD that enables more energy saving in the future. On the other hand, this action increases the energy cost for generating slack in the current interval. Therefore, there is sweet-spot that maximises the overall energy savings.

3.3.6 Conclusion

According the experimental results, the proposed cooperative approach shows dramatic improvements in energy-savings compared to previous work and even when it is extended with independent slack management on each core. The largest improvements are achieved in scenarios when there is contention in the cache, when the performance target is high, and when the core architecture is fixed. The experiments also shows a similar trend when the system scales to a larger number of cores. This confirms that CSM can effectively take advantage of the presented insights, about the potential of slack management in a cooperative way, to save significant processor energy while maintaining QoS for all applications.

[4] The value of energy budget is set as the percentage of the system energy estimated for the baseline configuration at each invocation of the resource-management-algorithm.

Chapter 4

Concluding Remarks and Future Work

In this thesis, different resource management approaches are studied in the context of a multi-core processor that runs a multi-programmed workload. The main objective is to reduce the energy consumption as much as possible, while maintaining QoS for all applications in the system. In this case, we assume that the QoS target is met if no application experiences any performance degradation compared to a baseline allocation of resources.

To this end, three different resource management approaches are presented in this thesis. First, we demonstrate that independent management of resources, such as per-core DVFS and cache partitioning, in many cases fail to save a considerable amount of energy without causing any performance degradation. To address this problem, we present a resource management scheme (in Paper-I) that controls these resources in a coordinated fashion, across all applications. Therefore, it can explore a multi-dimensional configuration space to find alternative configurations that meet the performance targets at a lower energy. This process is performed regularly during run-time to adapt to the dynamic characteristics of the processor workload.

Next, we extend the scope of coordinated resource management (in Paper-II) to include dynamic adaptation of the micro-architectural resources of each core. This approach improves the energy savings by exploiting instruction and memory level parallelism (ILP and MLP, respectively) in the resource trade-offs, in coordination with DVFS and cache partitioning.

Finally, we demonstrate (in Paper-III) that energy savings can improve substantially by allowing short-term deviations from the baseline performance targets, in a controlled way. Here, we introduce a measurable concept, called *slack*, that tracks these deviations to avoid any QoS violation. Based on this, we present Cooperative Slack Management (CSM) that attempts to find opportunities to generate slack when the energy cost is low, and consume it later when it saves a large amount of energy. We show that slack can be transferred between QoS-constrained applications, which enables new opportunities for energy saving. Therefore, CSM can save significant energy, even in scenarios when the resource management schemes in previous work (Paper-II) and an extended version with independent local slack management are not effective.

The proposed resource management schemes are designed based on simple analytical performance and energy models. The accuracy of these models affect both the QoS targets and energy savings. Therefore, in Paper-II, we present a low-overhead hardware extension that improves the accuracy of modeling MLP for different core sizes and a range of possible cache allocations. Next, in Paper-II, a dynamic sampling technique is proposed that improves the accuracy of the models during run-time.

Furthermore, to conduct experiments based on new hardware techniques, a novel simulation framework is designed in this work (detailed in Chapter 2). This framework enabled extensive experiments with full execution of benchmark applications in a wide range of scenarios. This is achieved by a technique based on SimPoint [17] that reduces the simulation time by several orders of magnitude. Using this technique, the simulator regenerates a proxy of the benchmark instruction streams in a multi-programmed workload that runs on a multi-core system under the control of a specific resource-management-algorithm.

This study leads to several interesting questions for future work. For example, CSM uses a parameter called "energy budget" that determines an upper bound for the energy cost of generating slack. This is currently a static parameter with a value that is derived empirically. But, according to the sensitivity study in Paper-III, there is a sweet spot that maximizes the energy savings. Therefore, one research question, for future work, is how to design a new approach that dynamically adapts the energy budget according to the workload characteristics during run-time.

Another potential direction is to study the possibility of using slack in a more extreme way as follows. Assuming high contention in the shared cache; a subset of applications with sufficient slack can be halted to alleviate the contention. This way, the remaining applications can utilize the entire cache capacity and enjoy a significant performance boost in addition to energy saving. Therefore, they can collect a considerable slack that can be used in the future to return the favor to the halted applications.

Finally, the proposed sampling technique in Paper-III operates in a hybrid scheme that supports both passive and active sampling of selected configurations. Thus, another research question is how to select the best resource configurations for active sampling that maximizes the modeling accuracy while imposing minimal interference with the normal operation of the resource manger? These are some avenues that can be explored in future that build on the research in this thesis.

Bibliography

[1] M. Duranton, K. De Bosschere, B. Coppens, C. Gamrat, M. Gray, H. Munk, E. Ozer, T. Vardanega, and O. Zendra, *The HiPEAC Vision 2019*. HiPEAC CSA, Jan. 2019. [Online]. Available: https://hal.inria.fr/hal-02314184

[2] D. Albonesi, R. Balasubramonian, S. Dropsbo, S. Dwarkadas, E. Friedman, M. Huang, V. Kursun, G. Magklis, M. Scott, G. Semeraro, P. Bose, A. Buyuktosunoglu, P. Cook, and S. Schuster, "Dynamically tuning processor resources with adaptive processing," *Computer*, vol. 36, no. 12, pp. 49–58, 2003.

[3] A. Herdrich, E. Verplanke, P. Autee, R. Illikkal, C. Gianos, R. Singhal, and R. Iyer, "Cache QoS : From Concept to Reality in the Intel ® Xeon ® Processor E5- 2600 v3 Product Family," in *2016 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2016, pp. 657–668.

[4] M. Azhar, M. Pericàs, and P. Stenström, "Sac: Exploiting execution-time slack to save energy in heterogeneous multicore systems," in *Proceedings of the 48th International Conference on Parallel Processing*, ser. ICPP 2019. Association for Computing Machinery, 2019, pp. 1–12.

[5] M. Azhar, P. Stenström, and V. Papaefstathiou, "Sloop: Qos-supervised loop execution to reduce energy on heterogeneous architectures," *ACM Transactions on Architecture and Code Optimization*, vol. 14, no. 4, pp. 1–25, 2017.

[6] M. G. Moghaddam and C. Ababei, "Dynamic energy management for chip multi-processors under performance constraints," *Microprocessors and Microsystems*, vol. 54, pp. 1–13, 2017. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0141933117301606

[7] R. P. Pothukuchi, A. Ansari, P. Voulgaris, and J. Torrellas, "Using multiple input, multiple output formal control to maximize resource efficiency in architectures," in *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, 2016, pp. 658–670.

[8] J. Suh, C.-T. Huang, and M. Dubois, "Dynamic mips rate stabilization for complex processors," *ACM Transactions on Architecture and Code Optimization*, vol. 12, no. 1, apr 2015. [Online]. Available: https://doi.org/10.1145/2714575

[9] K. Choi, K. Dantu, W.-C. Cheng, and M. Pedram, "Frame-based dynamic voltage and frequency scaling for a mpeg decoder," in *Proceedings of the 2002 IEEE/ACM International Conference on Computer-Aided Design*, ser. ICCAD '02. New York, NY, USA: Association for Computing Machinery, 2002, p. 732–737. [Online]. Available: https://doi.org/10.1145/774572.774680

[10] C. Hughes, J. Srinivasan, and S. Adve, "Saving energy with architectural and frequency adaptations for multimedia applications," in *Proceedings. 34th ACM/IEEE International Symposium on Microarchitecture. MICRO-34*, 2001, pp. 250–261.

[11] D. Lo, L. Cheng, R. Govindaraju, P. Ranganathan, and C. Kozyrakis, "Heracles: Improving resource efficiency at scale," in *Proceedings of the 42nd Annual International Symposium on Computer Architecture*, ser. ISCA '15. New York, NY, USA: Association for Computing Machinery, 2015, p. 450–462. [Online]. Available: https://doi.org/10.1145/2749469.2749475

[12] T. Patel and D. Tiwari, "Clite: Efficient and qos-aware co-location of multiple latency-critical jobs for warehouse scale computers," in *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2020, pp. 193–206.

[13] S. Chen, C. Delimitrou, and J. F. Martínez, “Parties: Qos-aware resource partitioning for multiple interactive services,” in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS ’19. New York, NY, USA: Association for Computing Machinery, 2019, p. 107–120. [Online]. Available: https://doi.org/10.1145/3297858.3304005

[14] H. Kasture and D. Sanchez, “Ubik: Efficient cache sharing with strict qos for latency-critical workloads,” in *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS ’14. New York, NY, USA: Association for Computing Machinery, 2014, p. 729–742. [Online]. Available: https://doi.org/10.1145/2541940.2541944

[15] R. Nishtala, V. Petrucci, P. Carpenter, and M. Sjalander, “Twig: Multi-agent task management for colocated latency-critical cloud services,” in *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2020, pp. 167–179.

[16] J. L. Henning, “Spec cpu2006 benchmark descriptions,” *ACM SIGARCH Computer Architecture News*, vol. 34, no. 4, pp. 1–17, 2006.

[17] T. Sherwood, E. Perelman, G. Hamerly, and B. Calder, “Automatically characterizing large scale program behavior,” in *Proceedings of the 10th International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS X. New York, NY, USA: Association for Computing Machinery, 2002, p. 45–57. [Online]. Available: https://doi.org/10.1145/605397.605403

[18] M. Van Biesbrouck, T. Sherwood, and B. Calder, “A co-phase matrix to guide simultaneous multithreading simulation,” in *IEEE International Symposium on - ISPASS Performance Analysis of Systems and Software, 2004*, 2004, pp. 45–56.

[19] “The Sniper Multi-Core Simulator,” http://snipersim.org Online; last modified: Feb 2019.

[20] T. E. Carlson, W. Heirman, S. Eyerman, I. Hur, and L. Eeckhout, “An evaluation of high-level mechanistic core models,” *ACM Transactions on Architecture and Code Optimization*, vol. 11, no. 3, aug 2014. [Online]. Available: https://doi.org/10.1145/2629677

[21] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi, “Mcpat: An integrated power, area, and timing modeling framework for multicore and manycore architectures,” p. 469–480, 2009. [Online]. Available: https://doi.org/10.1145/1669112.1669172

[22] H. Patil and T. E. Carlson, “Pinballs: Portable and shareable user-level checkpoints for reproducible analysis and simulation,” in *Proceedings of the Workshop on Reproducible Research Methodologies (REPRODUCE)*, vol. 2, 2014.

[23] H. Patil, C. Pereira, M. Stallcup, G. Lueck, and J. Cownie, “Pinplay: a framework for deterministic replay and reproducible analysis of parallel programs,” in *Proceedings of the 8th annual IEEE/ACM international symposium on Code generation and optimization*, 2010, pp. 2–11.

[24] “Pinballs,” http://snipersim.org/w/Pinballs Online; last modified: Sep 2014.

[25] M. K. Qureshi and Y. N. Patt, “Utility-based cache partitioning: A low-overhead, high-performance, runtime mechanism to partition shared caches,” in *2006 39th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO’06)*, 2006, pp. 423–432.

[26] B. Su, J. L. Greathouse, J. Gu, M. Boyer, L. Shen, and Z. Wang, “Implementing a leading loads performance predictor on commodity processors,” in *2014 USENIX Annual Technical Conference (USENIX ATC 14)*. Philadelphia, PA: USENIX Association, Jun. 2014. [Online]. Available: https://www.usenix.org/conference/atc14/technical-sessions/presentation/su

[27] R. Miftakhutdinov, E. Ebrahimi, and Y. N. Patt, “Predicting performance impact of dvfs for realistic memory systems,” in *2012 45th Annual IEEE/ACM International Symposium on Microarchitecture*, 2012, pp. 155–165.