Thesis for The Degree of Licentiate of Engineering

QoS Driven Coordinated Management of Resources to Save Energy in Multi-Core Systems

Mehrzad Nejat

Division of Computer Engineering
Department of Computer Science & Engineering
Chalmers University of Technology and Gothenburg University
Gothenburg, Sweden, 2019

QoS Driven Coordinated Management of Resources to Save Energy in Multi-Core Systems

Mehrzad Nejat

Copyright ©2019 Mehrzad Nejat
except where otherwise stated.
All rights reserved.

Technical Report No 204L
ISSN 1652-876X
Department of Computer Science & Engineering
Division of Computer Engineering
Chalmers University of Technology and Gothenburg University
Gothenburg, Sweden

This thesis has been prepared using LaTeX.
Printed by Chalmers Reproservice,
Gothenburg, Sweden 2019.

"Everything is theoretically impossible, until it is done."
- Robert Heinlein

Abstract

Reducing the energy consumption of computing systems is a necessary endeavor. However, saving energy should not come at the expense of degrading user experience. To this end, in this thesis, we assume that applications running on multi-core processors are associated with a quality-of-service (QoS) target in terms of performance constraints. This way, hardware resources can be throttled to minimize energy expenditure without violating the QoS requirements.

Typical resource management schemes control different resources such as processor cores and on-chip cache memory independently. These approaches are not effective under performance constraints for all applications. Therefore, this thesis presents multi-core resource management schemes that coordinately control several resources in a unified algorithm. This way, the resource manger can find trade-offs between resource allocations to different applications to reduce system-level energy consumption, while still meeting the QoS targets expressed as performance constraints for every application.

Implementing a coordinated resource management scheme that dynamically adapts to varying run time behavior of a multi-programmed workload without any prior knowledge about the applications is a challenging task. Two different schemes are presented in this thesis to address this challenge. Both schemes are invoked at regular intervals during program execution. They employ simple and, yet, sufficiently accurate analytical models and a novel hardware technique to predict the effect of different resource allocations on performance and energy for each application. Using a heuristic method, the multi-dimensional system configuration space is pruned in several levels to find the optimum resource settings, with respect to energy efficiency, in a negligible time.

In the first scheme a resource management algorithm is presented that coordinates the control of voltage-frequency (VF) of each processor core with partitioning of the on-chip cache space. In the second scheme, a re-configurable processor is considered in which sections of the core micro-architectural resources can be dynamically deactivated to save energy. The resource manager can reactivate these sections, at the proper time, to increase instruction and memory level parallelism (ILP/MLP). This introduces new trade-offs between processor core size, VF settings, and the allocation of cache space for each application. By exploiting these trade-offs, the second scheme improves the energy savings compared to the first scheme considerably.

The proposed schemes are evaluated using a novel simulation framework. This framework estimates the effect of different resource management algorithms on full execution of benchmark applications in a multi-programmed workload. According to the experimental results, the proposed schemes can save up to 18% of system energy while respecting the performance constraints of all applications. The average energy savings are 6% and 10% with the first and second schemes, respectively. Further experiments on the first scheme shows that energy savings can potentially improve up to 29% if the users can tolerate a bounded reduction in performance that leads to 40% longer execution time.

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

Acknowledgment

I am most grateful to my main supervisor Professor Per Stenström who spent a great deal of his time teaching me the fundamentals of scientific research. Not only did he patiently helped me through the challenges in my studies, but also he changed the way I think, define problems, and develop solutions.

I am also grateful to my co-supervisors Associate Professor Miquel Pericàs and Dr. Madhavan Manivannan. I received many insightful comments and help from them during my studies. Over the past year, we spent many hours each week discussing the detailed technical challenges in my work with Madhavan. I am also thankful to Vassilis Papaefstathiou who helped me in the first years of my PhD studies and Fredrik Dahlgren who has been my examiner.

I appreciate all my colleagues and friends, Petros, Ahsen, Prajith, Stavros, Evangelos, Albin, Stefano, Alexandra, Nadja, Fatemeh, Fazeleh, Amir, Hannaneh, Shirin, Arman, Mohammand, Pirah, Jing and many others who created a nice and friendly work environment. I specially thank my friend and office mate Waqar with whom I had many interesting discussions over the past years. I also thank other staff, especially Monica, Evan, and Rolf who provided helpful support during my work at Chalmers.

Finally, I express my deepest gratitude to my family, especially my parents, my lovely wife Farzaneh, my parents in law, my brother, and my sisters in law. Without their support, sacrifices, and encouragement I could not achieve any success in my work.

This research has been funded by the European Research Council, under the MECCA project, contract number ERC-2013-AdG 340328 and Swedish Research Council (Vetenskapsr\aa det) 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"
Technical report arXiv:1911.05101 [cs.AR]
In submission to the Journal of Parallel and Distributed Computing (JPDC)

II Mehrzad Nejat, Madhavan Manivannan, Miquel Pericàs, Per Stenström "Coordinated Management of Processor Configuration and Cache Partitioning to Optimize Energy under QoS Constraints"
Technical report arXiv:1911.05114 [cs.AR]
In submission to The 34th IEEE International Parallel & Distributed Processing Symposium (IPDPS 2020).

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:

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)

Contents

Abstract **v**

Acknowledgement **vii**

List of Publications **ix**

1 Introduction **1**

2 Experimental Methodology **3**

3 Summary of the Papers **5**
3.1 Summary of Paper I 5
3.2 Summary of Paper II 7

4 Concluding Remarks **11**

5 Paper I **15**

6 Paper II **35**

Chapter 1

Introduction

Improving energy efficiency of computing systems is a necessary endeavour across the entire application spectrum. At one end, the operational time of battery-powered devices ranging from embedded and cyber-physical systems to smartphones and lap-tops directly depend on their energy consumption. In fact, this has been a bottleneck for revolutionary leaps in consumer electronics such as smart glasses with limited battery capacity and high computational demands [1]. On the other end, energy consumption in servers and data centers, for both computation and cooling imposes a substantial operation and capital cost. In a broader perspective, with the exponential increase in computational demands worldwide, if the energy per computation remains unchanged, computing systems may soon become a major contributor to global warming and climate change. In fact, some projections suggest that by 2030 ICT will consume the equivalent of half of the global electricity production of 2014 [2].

Today, most of the computers are built from microprocessor chips that comprise multiple processing cores and an on-chip cache memory, referred to as multi-core systems. Most of the computational energy in a multi-core system is typically consumed by the processor cores executing instructions and off-chip accesses to the main memory. This energy can be reduced by controlling the processor and on-chip cache resources. The energy consumption of the processor can be substantially reduced by lowering its voltage-frequency (VF) setting. It is possible to save even more energy by deactivating sections of processor micro-architectural components [3, 4]. On the other hand, the number of off-chip memory accesses usually depends on how the on-chip cache space is managed. Numerous studies have shown that the number of accesses to the main memory can be reduced substantially if the cache space is efficiently partitioned among the processor cores [5–7]

On-chip processor and cache resources are typically controlled with the objective of maximizing system performance. However, this can lead to excessive energy expenditure. In many applications, an increase in performance beyond a certain level does not produce additional value. This can be determined by a quality-of-service (QoS) target. One example is multi-media applications in which the QoS is defined as a specific frames per second rate. Frame rates higher than the QoS target will not improve user experience considerably. Therefore, if every application running on the system is associated with a QoS target in terms of performance constraints, resources can be throttled to minimize energy consumption without degrading user experience.

A workload with performance constraints on all applications is usually not considered in prior works. In this scenario, typical schemes that control processor and cache resources independently lose their effectiveness. For example, any change in the default distribution of cache space among the workload, may lead to a performance degradation that violates the constraints of some applications. However, if processor and cache resources are managed in a coordinated fashion, it becomes possible to make trade-offs between different resources.

Hence, in this thesis I aim at answering the following question. Assuming that the QoS targets of all applications can be met by a baseline processor and cache setting, how can we dynamically find a more efficient combination of resource settings that reduces system energy without degrading the performance of any application?

This thesis is based on two papers. *Paper I* makes the following main contributions:

(1) A resource management scheme that dynamically controls the on-chip cache space and the VF setting of each processor core in a coordinated fashion to maximize system-level energy-efficiency while respecting the QoS targets of every application.

(2) A heuristic algorithm to assess a large multi-dimensional space of all possible combinations of resource settings at low overhead.

(3) Detailed evaluation, via a novel simulation framework, that efficiently estimates the effect of the resource management algorithm on full benchmark executions comprising thousands of billions of instructions.

Paper II improves the energy efficiency further by adding the control of micro-architectural resources of each processor core. This enables more efficient trade-offs between processor and cache resources at a reduced VF. The main contributions of *Paper II* are the following:

(1) A systematic analysis to understand the trade-offs between processor micro-architecture, VF setting, and cache space under QoS constraints for all applications.

(2) A resource manager that dynamically adapts processor and cache settings to workload behavior during execution in order to improve system-level energy efficiency under QoS constraints.

(3) A hardware design that enables prediction of performance and energy for different combinations of processor and cache settings, with negligible overhead and improved accuracy.

The rest of the thesis is organized as follows. Chapter 2 describes the experimental methodology and the simulation framework. A summary of each paper is presented in Chapter 3. Finally, Chapter 4 concludes the thesis.

Chapter 2

Experimental Methodology

To evaluate the proposed resource management schemes in this thesis, a multi-level simulation framework was designed and implemented. This framework was designed to capture the full execution of benchmark applications in a multi-core system that is controlled by different resource management algorithms (RMA). This chapter briefly explains each level of this framework.

Detailed architectural simulation of full benchmark executions that include hundreds to thousands of billions of instructions takes prohibitively long time. Therefore, a common approach in computer architecture research is to simulate only a short interval e.g. one billion instructions. But, this approach cannot capture the long-term behavior of benchmarks as they undergo different phases during their full execution. Capturing the long term behavior of benchmarks and the effect of resource management decisions at different program phases in the multi-core workload is necessary for this study.

Therefore, I designed a simulation framework based on the idea presented by Van Biesbrouck et al. in [8]. An overview of this framework is illustrated in Figure 2.1. It starts by running SimPoint analysis [9] on *whole program Pinballs* for *SPEC CPU 2006* benchmarks, downloaded from the Sniper website [10]. During this analysis, the program instruction stream is divided into slices with fixed instruction count (100M in this study). These slices are then clustered into several groups, called phases, and one slice is selected as the representative for each phase. Another 100M instructions prior to each representative slice is included which is used in the following step to warm-up the caches before detailed simulations. SimPoint also produces a phase weight, according to the number of slices that belong to each phase, as well as a phase trace. This trace contains the sequence of phases that a program will visit throughout the entire instruction stream. In the next step, after warming up the caches, detailed architectural simulation is performed on the representative slices, using Sniper-7.2 [11, 12] plus McPAT [13] for a range of resource settings. The results of these simulations including performance and power estimations are then collected in a database. The simulations for different program phases and resource settings are independent from each other. Therefore, they can be executed in parallel in a short time, if enough processing nodes are available.

The RMA simulator is designed to estimate the effect of different RMAs on a specific multi-programmed workload in which every application is executed at least one complete round. It starts by collecting the phase traces generated during SimPoint analysis for all the applications in the workload. It uses these traces together with the simulation results database to generate a proxy of full benchmark executions in the multi-core system under the control of a RMA. Figure 2.2 shows a simple example to explain this process in a two core system. The simulation starts in the first program interval[1] (I_1) in each application. The phase trace determines the corresponding phase for each application. The average time per instruction (TPI) is collected from the simulation results database for these program phases at the baseline resource setting. Using this information, the RMA simulator finds the next global event (t_1) which is the time when the application with lowest TPI finishes one interval (100M instructions). The RMA is invoked on the core running that application to

[1] Here, program "interval" and "slice" are used interchangeably.

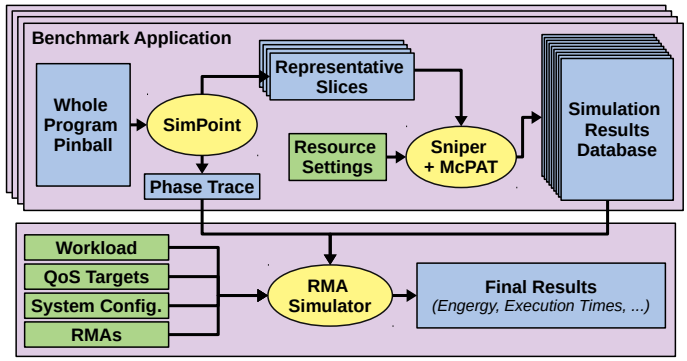

Figure 2.1: Overview of the simulation method.

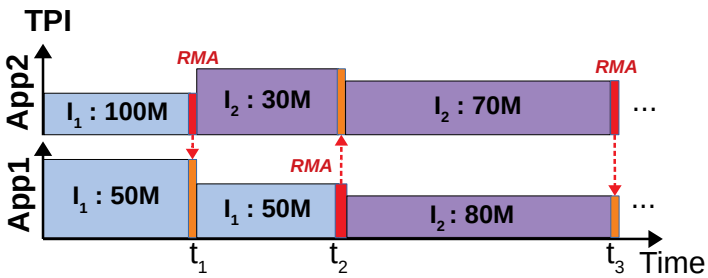

Figure 2.2: Run time behavior of the RMA simulator.

find a new resource setting [2]. After applying the new resource settings, the corresponding overheads are added to the simulation results for each core depending on the change in their resource allocations. The next global event (t_2) is found in a similar fashion. This process continues until the end of simulation.

One advantage of this framework is that it can simulate long-term phase behavior of full benchmark executions in a multicore workload, — corresponding to thousands of billions of instructions — in a short period of time [3]. This enables several experiments with different RMAs, different QoS targets, and different basic assumptions, in a reasonable time. This is possible as the detailed architectural simulation is performed in one step prior to RMA simulations and the same simulation result database can be used for all the experiments. On the other hand, this framework has two major limitations. First, it cannot capture the small variations between the program slices that belong to the same phase. It is based on the simplifying assumption that all these slices behave exactly similar to the representative slice. Second, it can only simulate multi-core systems with shared resources that are strictly partitioned among the cores. For example, it cannot simulate a cache or memory bandwidth partition that belongs to multiple cores. Therefore, in this study, I assumed a memory controller that equally partitions the available bandwidth among the cores.

[2] At the first invocation of the RMA, the performance counter statistics of the last interval is not yet available for the other application. This is needed for analytical performance and energy modeling in the RMA. Therefore, the RMA keeps the baseline resource setting at this point. However, this issue is not considered in this example for simplification.

[3] It was less than one hour in my experiments, excluding the time to generate the simulation results database in the first step. The first step can potentially be done in a similar time frame if sufficient processing nodes are available.

Chapter 3

Summary of the Papers

3.1 Summary of Paper I

Energy consumption of a multi-core system can be reduced by throttling its hardware resources. But, this usually comes at the expense of performance degradation. Therefore, a quality-of-service (QoS) target in terms of a minimum performance constraint is needed to enable energy optimization without degrading user experience.

A typical approach to manage resources of a multi-core system is to independently control each component, possibly with a different objective. For example, in [14–16] the voltage-frequency (VF) of the processor is controlled to meet a specific performance target; while, on the other hand, the shared last level cache (LLC) is usually partitioned to minimize the total number of cache misses [7]. Nevertheless, in a system under per-application performance constraints, independent schemes are not very effective. For example, if the performance of all applications is sensitive to their LLC allocation, an independent LLC controller cannot change the partitioning and guarantee the same performance for every application at the same time.

Hence, this paper attempts to answer the following question. Assuming that the QoS targets of every application in a multi-programmed workload is met with a baseline allocation of resources e.g. a mid-range core VF and equal distribution of LLC shares; how can a combination of resource allocations be identified dynamically that reduces system energy without degrading the performance of any application?

This paper's approach to address the above question is to design a unified resource management algorithm (RMA) that coordinately controls both per-core dynamic voltage-frequency scaling (DVFS) and partitioning of the LLC. To this end, a technique is needed to predict the performance and energy consumption of each application as a function of its resource allocation during run time. This technique must be fast enough to react to frequent phase changes in the workload with negligible overhead. Plus, it should not depend on any prior knowledge about the applications. Furthermore, the proposed algorithm must assess a large multi-dimensional configuration space efficiently without imposing a considerable overhead. In addition, This algorithm should be scalable to larger number of cores.

The main contributions of this paper are threefold: i) An online resource management scheme that coordinately controls per-core DVFS and LLC partitioning to maximize system-level energy efficiency under QoS constraints for all applications. ii) A heuristic algorithm to find the optimal resource settings in polynomial time. iii) Detailed evaluation of different resource management schemes, via a novel simulation framework that captures the phase changes during full execution of benchmark applications.

The proposed RMA is invoked by each core at regular intervals after executing a fixed number of instructions. It starts by collecting statistics of the past interval from hardware performance counters and an Auxiliary Tag Directory (ATD) [7]. ATD is a commonly used technique to emulate the operation of the main cache and predict the number of cache misses as a function of different number of allocated cache ways. Figure 3.1 shows an overview of the proposed RMA. It starts by collecting the cache miss profile from the ATD. This profile is used together with other statistics from hardware performance counters by a simple analytical model that predicts the performance as a function of the number of allocated

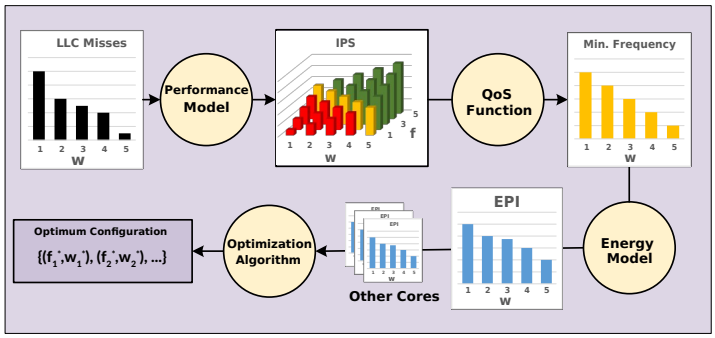

Figure 3.1: Overview of the proposed RMA (Figure 3 from *Paper I*).

LLC ways (w) and core frequency (f). The average number of instructions per second (IPS) is used here as a metric for performance. At this step, a large part of the configuration space can be pruned using the QoS target as follows. For every possible w, a minimum frequency ($f_{min}(w)$) is found that provides a performance higher or equal to the baseline resource allocation. Using another simple analytical model, for every pair of w and $f_{min}(w)$, an average energy per instruction (EPI) value is calculated. This value captures the energy consumption of the core and main memory accesses.

In the next step, the energy curve (as a function of w) is passed to the global optimization algorithm which already contains the last energy curves of other cores. It recursively reduces each pair of curves into one until an optimum set of $\{w_j\}$ is found for each core j that minimizes system energy while the sum of w_j values equals the LLC associativity. Finally, this new LLC partitioning is applied together with the corresponding f_{min} values for each core.

To evaluate the proposed RMA, SPEC CPU2006 benchmarks are categorized into four types based on two criteria: Memory intensity and cache sensitivity. If the number of Misses Per Kilo Instructions (MPKI) on the baseline LLC allocation is greater than a threshold, the application is counted as memory intensive, otherwise 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 counted as cache sensitive, otherwise cache insensitive. Several 4-core and 8-core workloads are generated based on different combinations of these categories. These workloads are simulated using the framework explained in Chapter 2

In the first experiment, energy savings are evaluated for each workload using the proposed combined RMA along with a 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 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

[1] Values below 1% are considered negligible.

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 then 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.

According to the experimental results, the proposed RMA provides an effective solution for the main research problem. It uses simple analytical models and a heuristic algorithm to assess both performance and energy in a large configuration space with negligible run time overhead. Therefore, it can dynamically find the most efficient resource setting to meet QoS targets of all applications with minimum energy consumption.

3.2 Summary of Paper II

Paper I (an extended version of [17]) proposes a resource management scheme that coordinates per-core dynamic voltage frequency scaling (DVFS) and LLC partitioning to reduce system energy while respecting the QoS targets of all applications. However, the scope for energy reduction using DVFS and cache partitioning is limited because of the following reason. When an application with a reduced cache share attempts to compensate the performance impact of additional cache misses by increasing core voltage-frequency (VF), it imposes a quadratic energy cost. Nevertheless, this action has no effect on memory stall time. Alternatively, if the resource manager (RM) can change the size of the processor core by activating more micro-architectural resources, it can achieve the required performance improvement by exploiting more instruction and memory level parallelism (ILP/MLP). This action has a lower energy cost compared to DVFS and potentially a strong impact on memory stall time.

Therefore, this paper attempts to answer the following question. Assuming that the QoS target of every application is met with a baseline resource setting, e.g. mid-range core size and VF, and equal partitioning of LLC; how to find new resource trade-offs when the RM has control over the size of core micro-architecture as well as VF and LLC partitioning that leads to a reduction in system energy without lowering the performance of any application?

This paper's approach to address this problem, includes two steps. First, a systematic analysis is performed on resource trade-offs in a range of workload mixes. Second, a resource management scheme is developed that dynamically controls the core size and VF along with LLC partitioning in a coordinated fashion. Compared to *Paper I*, the addition of a third control parameter i.e. size of the core micro-architecture, imposes new challenges. The simplifying assumption of constant MLP in *Paper I* is not acceptable when predicting the performance and energy on a different core size. Furthermore, the overhead of the resource management algorithm must remain negligible with the addition of a third dimension in the configuration space of each core.

Hence, the main contributions of this paper are the following: i) a systematic analysis to understand the trade-offs between core and cache resources for different workload scenarios, based on application characteristics. ii) An online resource management scheme that exploits the trade-offs introduced by leveraging ILP/MLP when coordinating the control of core size and VF with LLC partitioning under per-application performance constraints. iii) A hardware design to estimate the effect of MLP on cache misses for a range of core sizes and cache allocations. This hardware support enables fast and accurate prediction of performance and energy as a function of resource settings at run time. iv) Detailed evaluation and analysis of energy savings with the proposed scheme and the effect of modeling error on QoS violations.

An overview of the proposed scheme is depicted in Figure 3.2. The resource management algorithm is invoked on each core after executing a fixed number of instructions. It starts by collecting statistics from hardware performance counters to predict the effect of different resource settings (core size, core VF, and LLC allocation) on performance and energy, using simple analytical models. In the local optimization, the QoS target is used to prune the three-dimensional configuration space as follows. For every possible LLC way allocation w, an optimum core size and VF ($c^*(w)$ and $f^*(w)$) is found that meets the QoS target at minimum energy. The results constitute an energy curve as a function of w. The new energy curve for the current core is passed to the global optimization step where the last

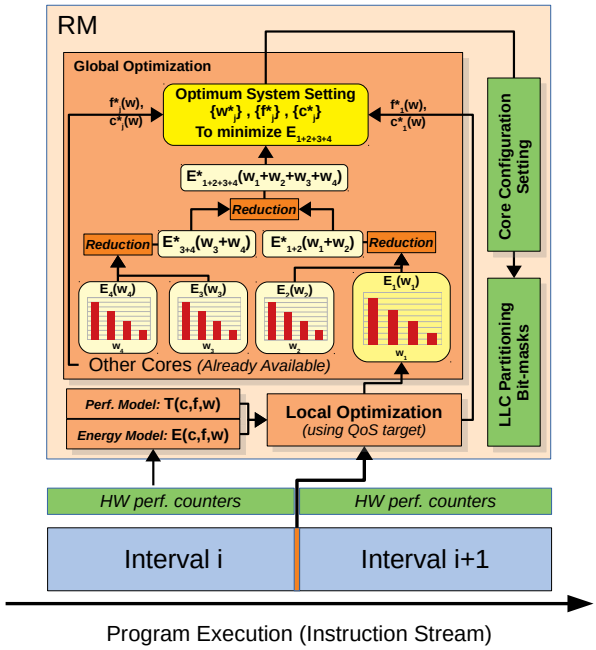

Figure 3.2: Overview of the proposed resource manager (Figure 3 from *Paper II*).

energy curves of other cores is already available. During this step which is similar to the optimization in *Paper I*, each pair of curves are recursively reduced until the final optimum setting is found.

The analytical models used during local optimization, require a means to estimate the effect of MLP for different core configurations and LLC allocations. Therefore, a hardware technique is designed in this work based on Auxiliary Tag Directory (ATD) [7]. While the original ATD counts the total number of cache misses for different number of cache ways, the proposed hardware extension uses a heuristic method to detect and ignore overlapping cache misses for different core sizes and cache allocations. The cache misses that overlap with a leading miss do not affect the program execution time [18, 19]. The overhead of this hardware extension is estimated to be less than 300 bytes per core.

In order to analyze the resource trade-offs between different applications under QoS constraints, the SPEC CPU2006 benchmarks are categorised based on two attributes: *Cache Sensitivity* and *Parallelism Sensitivity*. Similar to *Paper I*, an application is counted as cache sensitive if the variation in MPKI for different cache allocations around the baseline is above a specific threshold. Otherwise, it is counted as cache insensitive. On the other hand, if a change in core size leads to a certain variation in MLP, the application is counted as parallelism sensitive, otherwise parallelism insensitive. For all possible combinations of application categories, 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 same simulation framework as 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 two-, four-, and eight-core systems, respectively, which is less than 0.1% of a 100M instruction interval. The overheads imposed by other components of the system is also analyzed in details.

According to the experimental results, the proposed scheme provides an effective solution for the main research problem. It uses a novel hardware technique to evaluate performance and energy for a wide range of resource settings 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.

Chapter 4

Concluding Remarks

This thesis studies resource management schemes to improve the energy efficiency of multi-core processors. If a performance constraint is specified for every application in a multi-programmed workload, the hardware resources can be throttled to minimize energy consumption while satisfying the quality-of-service (QoS) requirements. Even though independent controllers for different resources such as configuration of the processor cores and partitioning of the shared cache simplifies the problem, such approaches are not effective under per-application performance constraints. Alternatively, a coordinated scheme that controls all the resources in a unified algorithm can evaluate all the resource trade-offs to optimize system energy.

Two different coordinated resource management schemes are presented in this thesis. The first scheme (detailed in *Paper I*) unifies the control of per-core dynamic voltage-frequency scaling (DVFS) and partitioning of the shared cache to find more efficient combinations of resource settings to reduce system energy under performance constraints. The second scheme (detailed in *Paper II*) improves the energy savings by exploiting new trade-offs when the resource manager can dynamically change the size of the processor micro-architecture in coordination with DVFS and cache partitioning.

Using simple analytical models and a novel hardware technique, I show in this thesis that the proposed schemes can estimate performance and energy for a wide range of resource settings in one step. In the next step, a heuristic algorithm prunes the multi-dimensional system configuration space in several levels to find the optimum resource setting at a negligible run time overhead.

There are several interesting questions for the future work. First, the proposed resource management algorithms have a short term optimization scope that only considers the upcoming program execution interval. They do not have any memory of the past events or any speculations about the future. It would be interesting to study how we can collect and use this information to improve the energy savings. Second, according to the experimental results the energy savings depend on the workload characteristics. It would be interesting to study how we can use this information to guide the system scheduler to collocate applications more efficiently. Finally, the proposed schemes depend on specific hardware support that enables fast and accurate analytical modeling. Can we instead use software techniques such as program phase tables and feedback loops to improve the accuracy of the models and reduce the probability of QoS violations? These are some avenues I would like to explore that build on the research in this thesis.

Bibliography

[1] N. M. Kumar, N. Kumar Singh, and V. K. Peddiny, "Wearable smart glass: Features, applications, current progress and challenges," in *2018 Second International Conference on Green Computing and Internet of Things (ICGCIoT)*, Aug 2018.

[2] 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

[3] D. H. Albonesi, R. Balasubramonian, S. G. Dropsbo, S. Dwarkadas, E. G. Friedman, M. C. Huang, V. Kursun, G. Magklis, M. L. Scott, G. Semeraro, P. Bose, A. Buyuktosunoglu, P. W. Cook, and S. E. Schuster, "Dynamically tuning processor resources with adaptive processing," *Computer*, 2003.

[4] Y. Eckert, S. Manne, M. J. Schulte, and D. A. Wood, "Something old and something new: P-states can borrow microarchitecture techniques too," in *ISLPED*, 2012.

[5] L. Funaro, O. A. Ben-Yehuda, and A. Schuster, "Ginseng: Market-driven llc allocation," in *USENIX ATC 16*, 2016.

[6] H. Dybdahl and P. Stenstrom, "An Adaptive Shared/Private NUCA Cache Partitioning Scheme for Chip Multiprocessors," in *HPCA*, 2007.

[7] M. Qureshi and Y. Patt, "Utility-based cache partitioning: A low-overhead, high-performance, runtime mechanism to partition shared caches," in *MICRO*, 2006.

[8] M. Van Biesbrouck, T. Sherwood, and B. Calder, "A co-phase matrix to guide simultaneous multithreading simulation," in *ISPASS*, 2004.

[9] T. Sherwood, E. Perelman, G. Hamerly, and B. Calder, "Automatically characterizing large scale program behavior," *ACM SIGARCH Computer Architecture News*, 2002.

[10] "Pinballs," http://snipersim.org/w/Pinballs Online; last modified: Sep 2014.

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

[12] T. E. Carlson, W. Heirman, S. Eyerman, I. Hur, and L. Eeckhout, "An Evaluation of High-Level Mechanistic Core Models," *ACM TACO*, 2014.

[13] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi, "McPAT 1.0: An Integrated Power, Area, and Timing Modeling Framework for Multicore Architectures," *Micro-42*, 2009.

[14] J. Suh, C.-T. Huang, and M. Dubois, "Dynamic MIPS Rate Stabilization for Complex Processors," *ACM TACO*, 2015.

[15] K. Choi, K. Dantu, W.-C. Cheng, and M. Pedram, "Frame-based dynamic voltage and frequency scaling for a MPEG decoder," in *ICCAD '02*, 2002.

[16] R. P. Pothukuchi, A. Ansari, P. Voulgaris, and J. Torrellas, "Using multiple input, multiple output formal control to maximize resource efficiency in architectures," in *ISCA*, 2016.

[17] M. Nejat, M. Pericas, and P. Stenstrom, "Qos-driven coordinated management of resources to save energy in multi-core systems," in *IPDPS*, 2019.

[18] B. Su, J. L. Greathouse, J. Gu, M. Boyer, L. Shen, and Z. Wang, "Implementing a leading loads performance predictor on commodity processors," in *USENIX ATC*, 2014.

[19] R. Miftakhutdinov, E. Ebrahimi, and Y. N. Patt, "Predicting performance impact of dvfs for realistic memory systems," in *Micro*, 2012.