Thesis for The Degree of Doctor of Philosophy

Adaptive Task Scheduling and Resource Management Techniques for Improving Energy Efficiency on Multi-core Systems

Jing Chen

Division of Computer and Network Systems
Department of Computer Science & Engineering
Chalmers University of Technology
Gothenburg, Sweden, 2023

Adaptive Task Scheduling and Resource Management Techniques for Improving Energy Efficiency on Multi-core Systems

Jing Chen

Thesis supervisor:
Prof. Miquel Pericàs, Chalmers University of Technology, Sweden

Thesis co-supervisors:
Dr. Madhavan Manivannan, Chalmers University of Technology, Sweden
Dr. Bhavishya Goel, Chalmers University of Technology, Sweden

Examiner & Chairman:
Prof. Per Stenström, Chalmers University of Technology, Sweden

Opponent:
Prof. Dimitrios Nikolopoulous, Virginia Tech, United States

Grading Committee:
Prof. Martin Schulz, Technical University of Munich, Germany
Prof. Osman Unsal, Barcelona Supercomputing Center, Spain
Prof. Michael O'Boyle, University of Edinburgh, United Kingdom

Deputy Committee:
Prof. Philippas Tsigas, Chalmers University of Technology, Sweden

ISBN 978-91-7905-967-5
Doktorsavhandlingar vid Chalmers tekniska högskola, Ny serie nr 5433.
ISSN 0346-718X

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

This thesis has been prepared using LaTeX.
Printed by Chalmers Reproservice,
Gothenburg, Sweden 2023.

Abstract

The growing impact of energy on operational cost and system robustness becomes a strong motivation for improving energy efficiency in parallel computing systems, in addition to performance. Hardware features such as core asymmetry and Dynamic Voltage and Frequency Scaling (DVFS) aim to provide opportunities for energy-efficient computing. However, it also complicates parallel application development. Task-based parallel programming models have been shown to be a powerful approach for developing parallel applications, allowing developers to express parallelism in the form of tasks. To achieve the goal of energy-efficient execution of a task-based application on multi-core platforms, it is essential to understand application characteristics, underlying platform capabilities and their complex interplay in order to determine appropriate task schedule and resource allocation. Consequently, this thesis introduces four task schedulers - ERASE, STEER, JOSS and SWEEP - tailored for diverse platform capabilities and energy efficiency metrics.

ERASE targets reducing CPU energy consumption in non-user-controllable DVFS environments. The scheduler includes four modules: online performance modeling and power profiling modules provide runtime with execution time and power predictions; core activity tracing offers the instantaneous task parallelism and the task scheduler combines these information to enable CPU energy consumption predictions and dynamically determine the best resource allocation for each task. Moreover, ERASE is designed for quick adaptation to external DVFS changes.

STEER investigates the potential CPU energy savings by leveraging core asymmetry, CPU DVFS, and task characteristics. STEER comprises two predictive models for performance and power predictions, and a task scheduler that utilizes models for energy predictions and then identifies the best resource allocation and frequency settings for tasks. Additionally, STEER employs adaptive scheduling algorithms based on task granularity to handle DVFS overheads and coordinates cluster frequency tuning to mitigate interference from concurrent tasks on cluster-based platforms.

JOSS demonstrates that taking memory energy into account is crucial for reducing total energy consumption, even in the absence of a memory DVFS knob. The scheduler leverages knobs of core asymmetry, CPU DVFS, memory DVFS and task characteristics. JOSS builds a set of models using multivariate polynomial regression, providing predictions for the execution time, average CPU power and memory power of each task, when tuning the aforementioned knobs individually and simultaneously, to facilitate the scheduling decision in task scheduler. Furthermore, JOSS supports exploring energy reduction with and without performance constraints.

SWEEP leverages application attributes, especially inter-task parallelism, together with hardware knobs to predict the impact of task distributions and local task scheduling decisions on the global execution time and energy consumption. SWEEP is designed for exploring various energy performance trade-offs. It first categorizes application execution into high parallelism and low parallelism phases, determined by instantaneous inter-task parallelism. It applies different task scheduling algorithms for high and low parallelism phases respectively, predicting trade-offs associated with different configurations and determining the best task distribution, local task schedules and DVFS settings accordingly.

The four schedulers address the challenges of achieving energy efficiency in diverse computing environments and targeting various energy efficiency metrics for task-based parallel applications. They present a comprehensive approach of integrating predictive models and adaptive scheduling algorithms to fully exploit the capability of multi-core platforms for both energy savings and energy performance trade-offs.

Keywords: Energy Efficiency, Task Scheduling, Performance Modeling, Power Modeling, Dynamic Voltage and Frequency Scaling (DVFS), Runtime System

Acknowledgment

I would like to express my sincere gratitude to my main supervisor, Prof. Miquel Pericàs, for his significant and persistent support during my Ph.D. journey. I am grateful for the knowledge gained under his guidance and appreciate his mentorship. I would also like to thank my co-supervisors, Dr. Madhavan Manivannan and Dr. Bhavishya Goel, for their support and valuable feedback throughout my research.

I am thankful to Prof. Per Stenström, who served as my examiner, and Dr. Mustafa Abduljabbar for his help during my Ph.D. I am also grateful to the outstanding professors and colleagues with whom I had the privilege to work alongside, including Pedro, Ioannis, Lars, Risat, Pirah, Sonia, Hari, Minyu, Nikela, Mehrzad, Qi, Neethu, Magnus, Nadja, Waqar, Fareed, Mateo, Stavroula, Piyumal, Siavash, Konstantinos, Panagiotis, Monica, Arne, and many more.

I would like to express my deepest gratitude to my supportive family, especially my parents, my grandmother and my sister, who have been with me every step of the way. Finally, I would like to extend a special thanks to my partner Franz. Your love, encouragement and unwavering support have been instrumental throughout the journey.

This research has been funded by the European Union Horizon 2020 research and innovation program under grant agreement No.780681 (https://legato-project.eu/). This research has also received funding from the European High-Performance Computing Joint Undertaking (JU) under grant agreement No.956702 (https://eprocessor.eu). The JU receives support from the European Union's Horizon 2020 research and innovation program and Spain, Sweden, Greece, Italy, France, and Germany. The computations were enabled by resources provided by the Swedish National Infrastructure for Computing (SNIC), partially funded by the Swedish Research Council through grant agreement No.2018-05973 (https://www.vr.se/).

List of Publications

Appended publications

This thesis is based on the following publications:

[I] **Jing Chen**, Madhavan Manivannan, Mustafa Abduljabbar, and Miquel Pericàs
"ERASE: Energy Efficient Task Mapping and Resource Management for Work Stealing Runtimes"
Published in ACM Transactions on Architecture and Code Optimization (TACO) 2022.

[II] **Jing Chen**, Madhavan Manivannan, Bhavishya Goel, Mustafa Abduljabbar, and Miquel Pericàs
"STEER: Asymmetry-aware Energy Efficient Task Scheduler for Cluster-based Multicore Architectures"
Published in the 34th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD) 2022.

[III] **Jing Chen**, Madhavan Manivannan, Bhavishya Goel, and Miquel Pericàs
"JOSS: Joint Exploration of CPU-Memory DVFS and Task Scheduling for Energy Efficiency"
Published in the 52nd International Conference on Parallel Processing (ICPP) 2023.

[IV] **Jing Chen**, Madhavan Manivannan, Bhavishya Goel, and Miquel Pericàs
"SWEEP: Adaptive Task Scheduling for Exploring Energy Performance Trade-offs"
Accepted in the 38th IEEE International Parallel and Distributed Processing Symposium (IPDPS) 2024.

Other publications

The following publications are not included in the thesis.

[a] **Jing Chen**, Madhavan Manivannan, Mustafa Abduljabbar, and Miquel Pericàs
"Towards an Energy Aware Task Scheduler for Asymmetric Architectures"

Published in the 12th Nordic Workshop on Multi-Core Computing (MCC), 2019

[b] **Jing Chen**, Pirah Noor Soomro, Mustafa Abduljabbar, Madhavan Manivannan, and Miquel Pericàs
"Scheduling Task-parallel Applications in Dynamically Asymmetric Environments"
Published in the 49th International Conference on Parallel Processing - ICPP Workshops SRMPDS, 2020

Contents

Chapter 1

Introduction

1.1 Background

Parallel computing systems have predominantly prioritized performance over energy efficiency, leading to exorbitant power and thermal demands that negatively affect system robustness and operating costs. For instance, enhancing energy efficiency is now critical in high performance computing (HPC) not only for cost savings but also for advancing into the exascale supercomputing era. In mobile devices, improving energy efficiency directly correlates with extending battery life and enhancing user experience.

Multi-core processors, which are the workhorses in such systems, consequently accommodate several hardware features that target energy-efficient execution of applications. Dynamic voltage and frequency scaling (DVFS) is a widely adopted hardware feature that introduces the dynamic asymmetry for improving energy efficiency [1]. It provides opportunities to reduce power consumption by throttling the voltage and frequency of system components, e.g. CPU cores and memory subsystem, potentially with little to no performance degradation. Core asymmetry is another popular hardware feature incorporated in several architecture designs comprising multiple core types (single-ISA) with different micro-architectures onto a single die. This is referred to as static asymmetry, due to its fixed feature that cannot change at runtime. In addition, designs with asymmetric core-clusters reduce the cost and complexity of implementing DVFS by grouping cores of the same type into clusters, wherein cores in the same cluster can only operate at a common voltage and frequency setting. Such architectures have been adopted in systems ranging from mobile devices to HPC machines [2–9].

These features available in hardware provide an opportunity to execute applications in a wide range of settings with varying power consumption and performance and enable the exploration of different trade-offs between performance and power/energy consumption. The metric used for assessing the trade-offs can target a single objective, such as reducing energy consumption, or it can target combined objectives that prioritize energy savings and performance differently.

A common parallelization scheme to exploit such parallel computing systems is task-parallelism, which has been implemented in several production runtime

systems, e.g. Cilk [10], TBB [11], StarPU [12], and OpenMP's explicit tasks [13]. It involves expressing parallelism in an application in the form of tasks and their dependencies that are generated and resolved dynamically during execution. Different tasks exhibit diverse characteristics, such as computational intensity and memory access patterns. An application comprising tasks can be modeled as a directed acyclic graph (DAG), wherein independent tasks that do not have any outstanding dependency can be executed in parallel and is referred to as inter-task parallelism. In particular, exposing fine-grained inter-task parallelism allows programmers to efficiently scale applications to larger platforms. In addition, exploiting potential intra-task parallelism by partitioning a single task into smaller tasks and running on multiple cores helps make use of idle resources and reduce system idle energy [14, 15]. This is also referred to as task moldability in the thesis.

Work stealing is a well-known approach for scheduling task-parallel applications and is implemented in several production runtimes [10, 11, 13]. It has been shown to be effective with sufficient inter-task parallelism available and can scale to large core counts, while still ensuring load balancing even on asymmetric architectures [16, 17]. In work stealing schedulers, there is a local task queue associated with each core, which serves to hold the ready tasks (i.e. tasks whose dependencies are satisfied). When a core is idle, it first looks for ready tasks in its own task queue. If it does not find any, it attempts to steal tasks from the task queues of other cores. However, work stealing applied in its original form does not produce energy-efficient schedules. The two principles on which work stealing is based, namely the work-first principle and random victim selection, are neither aware of task characteristics nor of the cores' performance and energy profiles. As a result, such scheduling techniques can detrimentally impact energy consumption and performance of parallel applications.

Achieving energy-efficient execution of a task-based application on multi-core platforms entails understanding application characteristics, underlying platform capabilities and their complex interplay in order for a scheduler to determine appropriate task schedule and resource allocation. On one hand, the available hardware knobs (core asymmetry, CPU DVFS and memory DVFS) and diverse application attributes (inter-task parallelism, intra-task parallelism and task characteristics) present great opportunities for trade-offs between performance and power consumption. On the other hand, these knobs introduce additional dimensions to the task scheduling problem that is already computationally hard. Effectively navigating this complexity requires traversing a large search space to figure out the best task schedule and resource management decisions. Furthermore, the proposed scheduling techniques should offer adaptivity to cater to various energy efficiency optimization targets and diverse platform environment settings.

The goal of the thesis is to propose adaptive task scheduling and resource management techniques to tackle the challenges of achieving energy-efficient task scheduling for task-based applications on multi-core architectures.

1.2 Related Work

There is growing interest in improving energy efficiency for parallel applications running on multi-core architectures, which can be broadly grouped into three categories.

The first category includes a group of proposals that demonstrate the potential of using per-core DVFS to improve energy efficiency. HERMES [18] is a work-stealing runtime that coordinates DVFS frequencies associated with each core. The proposed approach introduces a workpath-sensitive algorithm, which slows down the frequency of thief cores (those stealing tasks from other cores' task queues), and a workload-sensitive algorithm that samples the queue workload size and selects appropriate DVFS frequency based on it. CATA [19] tunes the per-core DVFS based on task criticality, accelerating the cores that execute critical tasks and setting the cores that execute non-critical tasks to low-frequency power-efficient states. CATA also implements a hardware component to mitigate the performance overhead of DVFS reconfiguration. Acun et al. [20] use per-core DVFS and adopt an online history-based approach for performance and power predictions by sampling with every possible frequency. AAWS [21] targets asymmetric platforms and proposes several techniques (work-pacing, work-sprinting, and work-mugging strategies along with per-core DVFS) depending on each core's state (stealing vs executing). However, the aforementioned schedulers make restrictive assumptions about the capabilities of the underlying hardware platform and consequently have limited effectiveness on platforms featuring cluster-based DVFS.

The second category consists of several studies that propose diverse scheduling techniques to enhance the energy efficiency of multi-core platforms incorporating cluster-based DVFS. AEQUITAS [22] proposes a round-robin time-slicing algorithm on top of HERMES, where each active core within a core cluster controls DVFS for a short interval in a round-robin manner. Costero et al. [23] present a group of frequency scaling policies based on task criticality and workloads, as well as task scheduling policies for idling or switching off clusters of an asymmetric cluster-based platform based on the workload. However, they evaluate these policies individually. Shafik et al. [24] search for the best concurrency and frequency in a time-interval manner on a cluster-based symmetric platform, without considering the impact of core asymmetry and task characteristics. CHRT [25] is a phase-based scheduler that predicts task placement, cluster frequency, and number of cores for each execution phase based on only task criticality. CHRT has limited applicability for dynamic unfolding DAGs where criticality cannot be easily determined without explicit programmer support. Furthermore, the performance and power models in CHRT treat all tasks equally regardless of diverse task characteristics. Notably, none of these prior works leverage intra-task parallelism to enhance energy efficiency. These schedulers also do not fully exploit available hardware knobs, especially memory DVFS. Most importantly, they lack support for exploring various energy performance trade-off metrics.

Another line of related work targets energy and/or performance for single-threaded applications or multi-programmed applications instead of task-based applications. MemScale [26] targets energy consumption in the memory subsystem. They leverage dynamic profiling, performance and power modeling to

guide DVFS of memory controller and frequency scaling of memory channels and DRAM. David et al. [27] propose an intuitive algorithm that detects memory bandwidth utilization for tuning DVFS of memory subsystem. Sundriyal et al [28] aim to minimize system power consumption given the performance loss tolerance, with a proposed performance and a power model based on performance monitoring counters (PMC) to determine the best joint frequency setting for the entire application in a time window-based manner. CoScale [29] is an epoch-based framework for multi-programmed workloads, by collecting PMCs for model prediction and searching for the best frequency pair using a gradient-descent method.

1.3 Problem Statement

Fine-grained tasking and externally controlled DVFS: Applications often expose fine-grained task parallelism for achieving better scalability and load balancing in multi-core and many-core architectures [30, 31]. With fine-grained tasking, the execution time of tasks are in the order of microseconds, which makes the DVFS reconfiguration overheads non-negligible. Therefore, leveraging per-task DVFS is impractical in this case. Moreover, it is common that in a multi-user OS DVFS control is commonly not under control of the application but restricted to the kernel [32], the system administrator [33], or power management frameworks such as GEOPM [34]. Consequently, it is crucial to design an adaptive energy-efficient task scheduling technique that applies to fine-grained tasking with low overheads and can be reactive to externally controlled DVFS. Paper I aims to address the following question:

Question I: *How can we reduce the energy consumption of running fine-grained tasking parallel applications on multi-core platforms with externally controlled DVFS?*

Energy reduction with user-manageable CPU DVFS: Enabling CPU DVFS from user space, along with core asymmetry and diverse task characteristics, presents a complex, multi-dimensional task scheduling problem for achieving energy reduction. A large search space needs to be explored to identify the best task schedule and DVFS setting for each task, i.e. <cluster, number of cores, cluster frequency>. In addition, DVFS reconfiguration overheads, particularly significant for fine-grained tasks with execution times in the order of microseconds, can diminish the benefits of frequency throttling. Therefore, it is important to devise an adaptive frequency throttling strategy considering various task granularities. Furthermore, on platforms with cluster-level DVFS, it is possible that multiple tasks are scheduled to run on the same cluster and different frequencies are selected to achieve energy savings. Consequently, coordinating cluster frequency tuning is crucial to avoid destructive DVFS tuning interference. Paper II aims to address the following question:

Question II: *How to reduce energy consumption by leveraging core asymmetry, CPU DVFS, task characteristics in conjunction when running a task-based application on multi-core platforms featuring cluster-level DVFS?*

Leveraging memory DVFS to improve energy efficiency: In response

to the growing memory demand in emerging multi-core architectures, main memory bandwidth and capacities have also been steadily increasing, making memory energy a significant contributor to total energy consumption. Consequently, hardware vendors have started to integrate memory DVFS to further improve energy efficiency. Several works have highlighted the importance and the benefit of leveraging memory DVFS, since it opens up many opportunities for establishing trade-offs between performance and energy consumption [26–29]. Leveraging all the available knobs, i.e. core asymmetry, CPU DVFS, memory DVFS in conjunction with various task characteristics, leads to increased complexity in task scheduling. In addition, it is crucial that the scheduler aims to reduce the total energy consumption while still maintaining a good level of performance. Paper III aims to address the following questions:

Question III: *How to investigate the impact of tuning the available knobs, individually and collectively, on performance, CPU energy and memory energy consumption of a single task? How to determine the best task schedule and appropriate frequency pair to explore energy reduction with and without performance constraints for a task-based application?*

Various energy performance trade-offs exploration: The knobs available in hardware, i.e. core asymmetry, CPU DVFS and memory DVFS, provide an opportunity to execute applications in a wide range of settings and enable exploration of different trade-offs between performance and energy consumption. The trade-off can be assessed using metrics such as Energy Delay Product (EDP) and E^mD^nP that prioritize energy savings and performance differently. Existing prior works are only designed for a specific trade-off metric and lack the flexibility to explore various metrics. Moreover, understanding the impact of tuning knobs for a single task and their interplay effects on both the performance and energy consumption of that specific task, as well as the impact on the overall performance and energy consumption of the entire application, is complicated, especially in the context of a dynamically unfolding DAG during runtime. Furthermore, parallel applications exhibit both high and low task parallelism, which necessitates the adaptive scheduling algorithm for specified targets. Paper IV aims to address the following question:

Question IV: *How to design a scheduler that leverages the available hardware knobs and application characteristics to facilitate the exploration of various energy performance trade-offs when running a task-based application on multi-core platforms?*

1.4 Contributions

This thesis is based on four papers. **Paper I** addresses the first question, and the main contributions are:

- Paper I proposes ERASE: an energy-efficient task scheduler that combines power profiling, performance modeling and core activity tracing for energy-efficient mapping (i.e. choosing the cluster) and resource management (i.e. selecting the number of cores per task). The proposal exploits the insights of task moldability, task-type awareness and instantaneous task

parallelism detection for guiding scheduling decisions to reduce the total energy consumption of parallel applications.

- It describes how to integrate ERASE on top of work stealing runtimes, using the XiTAO [35] runtime for the prototype implementation.
- It compares ERASE to state-of-the-art scheduling techniques on top of the runtime and the evaluation shows that ERASE achieves up to 31% energy savings and outperforms the state-of-the-art by 44% on average.

Paper II addresses the second question, and the main contributions are:

- Paper II shows that considerable energy savings can be achieved by leveraging static asymmetry, dynamic asymmetry and task heterogeneity in conjunction. Accordingly, we propose STEER, a task scheduling framework that exploits these features to predict the best energy-saving configuration for each task.
- It proposes a performance model and a power model to predict the impact of varying the core type, the number of cores and the frequency, that are not limited by the availability of performance counters.
- It develops heuristics to (1) manage DVFS overheads by applying adaptive scheduling techniques for varying task granularities, and (2) mitigate DVFS interference from concurrent task execution on cluster-based multi-core architectures.

Paper III addresses the third question, and the main contributions are:

- Paper III demonstrates that (1) leveraging static asymmetry and dynamic asymmetry, i.e. core type, number of cores, core frequency and memory frequency, together with task characteristics, enables significant reduction in energy consumption; (2) even in the absence of a memory DVFS knob, taking total energy consumption (including CPU and memory) into account for configuration selection results in lower energy consumption compared to only considering CPU energy.
- It proposes the JOSS runtime scheduling framework for task-based parallel applications on multi-core architectures, which provides the ability to explore various energy performance trade-offs.
- It builds a set of models using multivariate polynomial regression capable of accurately predicting the execution time, average CPU power, and average memory power of each task when tuning the four available knobs, individually and simultaneously.

Paper IV addresses the fourth question, and the main contributions are:

- Paper IV proposes SWEEP, a scheduler that leverages architectural knobs (core asymmetry, CPU and memory DVFS) together with application attributes (inter-task parallelism, intra-task parallelism and task characteristics) to facilitate energy performance trade-off exploration. SWEEP

splits execution into phases, classifies phases as high parallelism and low parallelism, and determines the best task schedule and DVFS settings for each phase. SWEEP adapts task schedule and DVFS settings to various energy performance trade-off metrics.

- SWEEP is extensively evaluated by comparing it with several state-of-the-art proposals that target energy-efficient execution. Our evaluation demonstrates the effectiveness of SWEEP to flexibly target different energy performance trade-off metrics, a feature that is not supported by other state-of-the-art proposals. More specifically, SWEEP achieves 19.9%, 36.4% and 9.5% reduction on average in terms of EDP, ED^2P and E^2DP compared to the best performing state-of-the-art.

1.5 Organization

The rest of this thesis is organized as follows. A summary of each paper is provided in Chapter 2. Chapter 3 presents the concluding remarks and the potential future work. Finally, the four papers are appended to the end of this thesis.

Chapter 2

Summary of papers

2.1 Paper-I

2.1.1 Background

Parallel applications often rely on work stealing schedulers in combination with fine-grained tasking to achieve high performance and scalability. Random work stealing is a well-known approach for scheduling task-parallel applications [16, 17] that has been implemented in several production runtimes, such as Cilk [10], TBB [11] and OpenMP's explicit tasks [13]. However, reducing the total energy consumption in the context of work stealing runtimes is still challenging, particularly when using asymmetric architectures with different types of CPU cores. This is because the two principles on which random work stealing is based, namely the work-first principle and random victim selection, are neither aware of task characteristics nor of the cores' performance and energy profiles.

A common approach employed in earlier works to achieve energy savings in work-stealing runtimes involves leveraging DVFS, wherein the throttling is carried out based on factors like task parallelism, stealing relations, task criticality and workload sizes [18, 19, 21, 22]. However, the reliance on DVFS limits their applicability due to several reasons. First, studies have shown that DVFS transition delay is around 100 microseconds, thereby the DVFS switching overheads are non-negligible for fine-grained tasks that execute in the order of microseconds [19, 21, 36, 37]. Second, per-core DVFS control utilized in prior works precludes their techniques applicability on cluster-based architectures [18, 19, 21]. With cluster-level DVFS, multiple tasks attempting frequency changes within the same cluster will result in destructive interference, since the decision taken to reduce energy consumption of a task mapped to a specific core can affect concurrently running tasks on the same cluster. More importantly, it is common that in a multiuser OS, DVFS is most often not under the control of the application, but is externally controlled by the Linux kernel [32], the system administrator [33], or power management frameworks such as GEOPM [34]. Consequently, an energy-efficient runtime designed to be reactive to both given static frequency settings and externally controlled DVFS, instead of relying on actively changing DVFS settings, has the potential to be a more general solution to the problem of energy-aware scheduling.

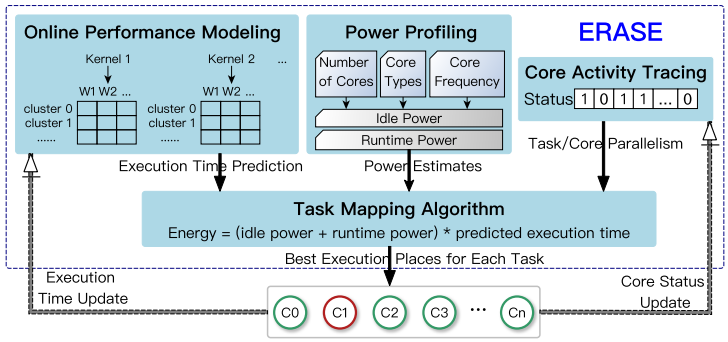

Figure 2.1: An overview of ERASE comprising four modules. C_x denotes the core id, W_y denotes the possible resource widths. In status, "1" denotes that C_x is in active state while "0" denotes the core is in sleep state.

2.1.2 Proposed Approach

In Paper I, we present ERASE - an energy-efficient task scheduler to address the problem of reducing the total CPU energy consumption when running task-based applications on multi-core platforms wherein the frequency throttling is externally controlled.

In a nutshell, ERASE reduces energy consumption of executing a task DAG by attempting to execute each task with the lowest possible energy consumption. To reduce the energy consumed by each task, runtime needs to evaluate the energy consumption of all different resource configurations <cluster, number of cores> and identify the one that consumes the least energy for the task. Hence, it is crucial that the runtime can predict the execution time and the power consumption and thereby the energy consumption to facilitate the configuration selection.

ERASE leverages the insights of task moldability (i.e. intra-task parallelism) and task-type awareness to determine the appropriate resource allocation for each task, including selecting the cluster and determining the number of cores (resource width). Meanwhile, it adaptively puts idle cores to sleep state by applying an exponential back-off sleeping strategy to further reduce the energy waste. Figure 2.1 provides an overview of ERASE, which comprises four essential modules: online performance modeling, power profiling, core activity tracing and a task mapping algorithm.

The online performance modeling module enables ERASE to predict the execution times of a task when mapped on different resource configurations (i.e. number/type of cores). Online performance modeling adopts a history-based model, it continuously monitors task execution during runtime and updates look-up tables. A look-up table is implemented for each kernel. The size of each look-up table equals the product of the number of clusters and the number of available resource widths on a platform. The module can not only provide performance predictions for incoming tasks by referring to corresponding look-up table entries but also instantly detect the external controlled frequency changes by comparing the new execution time to previous table entry records. Once the frequency change is detected, the module will

reset the look-up table entries to zeros and retrain the model.

The power profiling module provides power estimates with respect to different resource configurations for given core frequencies. The module groups tasks into three representative types: compute-bound, memory-bound and cache-intensive. By profiling a set of microbenchmarks, we compute the arithmetic intensity (AI) values and employ the k-NN algorithm to cluster them into the three groups. This is done by computing the euclidean distances of the AIs and then selecting an AI threshold that lies on the frontier region between the clusters. The averaged power values from each group are used as power estimates once a task is classified to a certain task type. During runtime, we compute the AI value of a task and map the task to one of the groups where the average power consumption of the group is utilized as the reference.

Core activity tracing continuously tracks the activities (i.e. work stealing attempts) and status (i.e. active or sleep) of each core and infers the instantaneous task parallelism, which gives the task mapping algorithm a hint for attributing the idle power consumption to concurrently running tasks accurately. This is because the idle power obtained with power profiling module is the total idle power of the entire chip/cluster. It cannot be attributed to the task directly because this is shared between concurrently running tasks. In other words, each concurrently running task shares a portion of idle power at any given moment, and attributing the entire idle power from power profiling to a single task may lead to inaccurate energy estimation. Finally, the task mapping algorithm integrates the aforementioned information from the three modules and guides the scheduling decision for each task based on the energy estimates of running the task on different resource configurations.

In work stealing runtimes, idle cores that continuously attempt stealing without success lead to energy waste. The problem of reducing energy consumption during idle periods requires the runtime to be able to detect the cores' instantaneous utilization and dynamically put idle cores to sleep. The challenge is to determine the sleep duration such that energy consumption during the period is minimized with minimal performance impact. To minimize energy waste from continuous work stealing attempts by idle cores, ERASE adopts an exponential back-off sleep strategy [38]. The core activity tracing module keeps track of the number of unsuccessful steal attempts per core. When the number of attempts exceeds a preset threshold, a core is put into sleep. The core will sleep for an exponentially increasing time (ranging from 1ms to 64ms) if it cannot find any ready task upon wake up.

2.1.3 Evaluation

We evaluate the effectiveness of ERASE by comparing against several state-of-the-art scheduling techniques on an asymmetric platform - NVIDIA Jetson TX2 and a symmetric platform - a dual-socket 16-core per socket Intel Xeon Gold 6130 node.

The results obtained from Jetson TX2 indicate that ERASE achieves 10.1% energy reduction and 43.6% performance improvement on average, compared to the resource-agnostic and non-moldable state-of-the-art across different benchmarks and different degrees of parallelism (dop). In addition, we observe that the energy reduction is significant when the performance gap of two

clusters is large.

The results obtained from the symmetric platform show that ERASE consumes 15% less energy and achieves 18% performance improvement on average when compared to state-of-the-art schedulers across various benchmarks and different dop. We also show that the energy reduction is significant especially when dop is low.

The results of analyzing the adaptability of ERASE show that ERASE models can detect dynamical frequency changes and automatically adapt to the new frequency instantly, which achieves 30% energy savings and 23% performance improvement compared to the scenario in which the models are unaware of the frequency change and keep using the same model settings detected at the beginning of execution.

2.1.4 Conclusion

We propose ERASE - an energy-efficient task scheduler for reducing the total CPU energy consumption of executing fine-grained task-based parallel applications on multi-core platforms. The scheduler works by estimating the energy consumption of different resource configurations at a per-task level and performs task mapping decisions so as to minimize the energy consumption of each task. It also adaptively puts cores that repeatedly execute work stealing loop without success to sleep state by applying an exponential back-off sleeping strategy. Furthermore, ERASE is capable of adapting to both given static frequency settings and externally controlled DVFS.

2.2 Paper-II

2.2.1 Background

In order to improve energy efficiency, hardware vendors have integrated DVFS and core asymmetry on multi-core platforms. DVFS is a well-known technique that represents the dynamic asymmetry feature and offers great promise to significantly reduce power consumption by adapting both voltage and frequency of the system with respect to various workloads [1]. Core asymmetry designs that are being composed of multiple core types with different micro-architectures onto a single die, provide diverse energy-performance capabilities for different workloads, which is a demonstration of static asymmetry due to its fixed feature at design time.

Core-clustering paradigm [39] is adopted for organizing such architectures to reduce the hardware design complexity [2, 3, 5, 40–43]. In such designs, cores of the same type are clustered together to share common resources like last level cache and memory controller. For power management, these cluster-based platforms often support per-cluster DVFS, where cores in the same cluster must operate at the same voltage-frequency level.

Besides static and dynamic asymmetry, energy-efficient scheduling can benefit from considering task characteristics, which motivates us to understand the contributing factors to energy consumption in order to make energy-efficient task schedules on such platforms. These factors include task placement (i.e. task mapping to appropriate resources), and potential intra-task parallelism (task moldability) exploitation by running a single task on multiple resources to reduce resource oversubscription and make use of idle resources, and task granularity (size) in relation to the DVFS timing overheads. Collectively, we refer to these three aspects, namely task characteristics, task moldability and task granularity, as task heterogeneity.

Existing studies [18–25, 44] either explore the energy benefits of levering dynamic asymmetry in isolation or the combination of static asymmetry and dynamic asymmetry without considering the impact of task heterogeneity. Moreover, the proposals that use per-core DVFS [18–21] have limited applicability on cluster-based architectures. In Paper II, we show that leveraging task heterogeneity in conjunction with static asymmetry and dynamic asymmetry can provide significant opportunities for energy reduction.

There are three major challenges that need to be addressed to enable the energy-efficient schedules on platforms with cluster-level DVFS. First, predicting the best execution place and frequency setting for a task involves exploring a three-dimensional search space, i.e. <cluster, number of cores, frequency>. The complexity is exacerbated by the need to estimate energy consumption with low runtime overhead and with reasonable accuracy. Second, it is necessary to design adaptive scheduling techniques for various task granularity, particularly for fine-grained tasks to exploit DVFS and reduce DVFS negative impacts simultaneously. Finally, with concurrent running tasks, it is possible that different frequencies are selected for energy reduction. This fact makes frequency coordination crucial for mitigating the detrimental energy impacts on the concurrent running tasks and reducing the overall energy consumption.

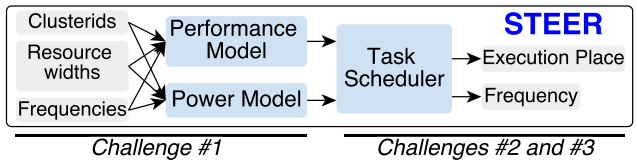

Figure 2.2: An Overview of STEER framework.

2.2.2 Proposed Approach

In this paper, we propose a task scheduling framework STEER. In a nutshell, STEER reduces the energy consumption of the entire DAG by running each task with the lowest energy consumption possible through identifying the best execution place and frequency setting. Figure 2.2 provides an overview of the essential components in STEER.

To address the aforementioned challenges, STEER leverages two predictive models to help the runtime accurately predict the impact of different execution place and frequency settings on the execution time and power consumption. Then a task scheduler is developed to leverage the models to predict the energy consumption, determine the best execution place and frequency setting that consumes the least modeled energy and schedule tasks for execution.

Performance Model: To predict the execution time and reduce the modeling overheads, STEER exploits a hybrid approach by first sampling a limited number of possible settings and utilizing a model to predict performance for the rest of settings. The performance model utilizes memory-boundness (MB) as a measure of the fraction of execution time that does not scale with core frequency [45]. Therefore, the total execution time at a fixed (sampled) frequency (f_0) could be split into a memory bound fraction (MB) and a computation fraction (1-MB):

$$Time_{f_0} = Time_{f_0} \times (MB + (1 - MB)) \tag{2.1}$$

When changing the core frequency, the computation fraction (1-MB) will scale proportionally with frequency, while the memory bound fraction (MB) will remain independent of the frequency. Consequently, the equation for predicting execution time based on MB at a different frequency (f) can be expressed as below:

$$Time_f = Time_{f_0} \times (MB + (1 - MB) \times \frac{f_0}{f}) \tag{2.2}$$

The performance model relies on knowing $Time_{f_0}$ and MB at frequency f_0 to predict the execution time for the task when running at other frequencies. $Time_{f_0}$ can be obtained by timing the task execution during the runtime. We obtain a second sample at a different frequency (f_0) and derive MB as follows:

$$MB = \frac{\frac{Time_f}{Time_{f_0}} - \frac{f_0}{f}}{1 - \frac{f_0}{f}} \tag{2.3}$$

Once MB is estimated by sampling at two different frequencies, the execution time predictions for all other frequencies are obtained using equation 2.2.

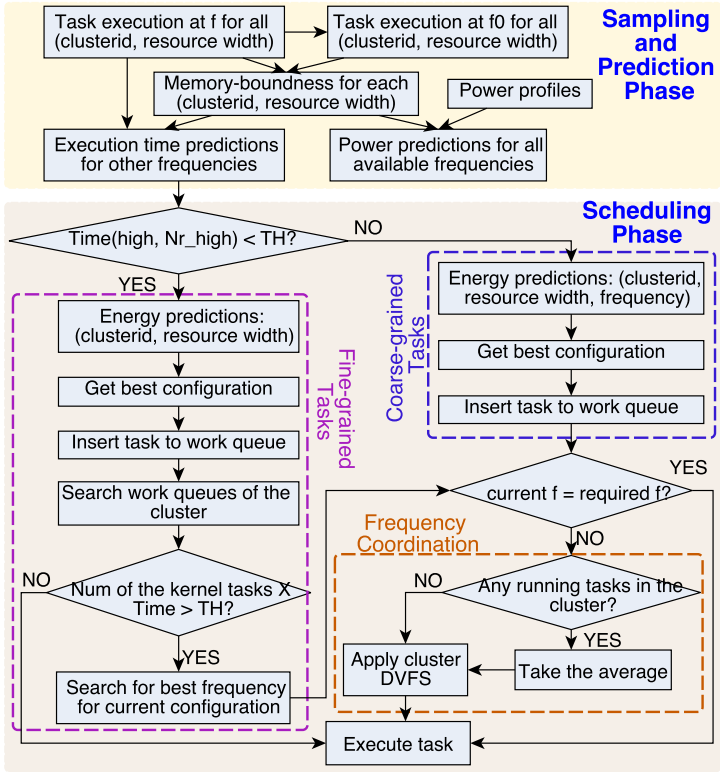

Figure 2.3: The work flow of STEER scheduler.

Power Model: The power model adopts an offline characterization approach, where look-up tables are constructed for power references during runtime. We profile a set of training benchmarks (NAS parallel benchmark suite in this work) to record the power consumption and execution time on different execution place and frequency settings. We then calculate the MB values and cluster these benchmarks into groups according to their MB values. Estimating the power consumption for a task at runtime involves mapping the task into one of the groups given its MB value and accessing the corresponding table entry. The offline power modeling is independent on applications and just needs to be done once for a specific platform (e.g. at install-time or boot-time), and has no impact on execution time.

Task Scheduler: The task scheduler in STEER essentially comprises two phases as shown in Figure 2.3. In sampling and prediction phase, the scheduler performs execution time sampling of tasks to compute MB values and enable performance and power prediction at different execution place and frequency settings. In scheduling phase, the scheduler utilizes the performance and power consumption prediction information to enable energy-efficient task scheduling. For coarse-grained tasks, the scheduler iterates all possible settings and identifies the one that consumes the least modeled energy and then performs the task mapping and frequency throttling accordingly. In the case of fine-grained tasks, DVFS throttling overheads are large enough to offset any benefit.

Therefore, STEER adaptively adjusts scheduling policy once fine-grained tasks are detected. Specifically, fine-grained tasks are composed together with other tasks that are instances of the same kernel and can be viewed as a single coarse-grain task, such that DVFS throttling can be performed for this entire group of tasks for further energy savings. To address the challenge of frequency coordination in cluster-based platforms, STEER develops a heuristic to mitigate the problem by tuning the frequency of the cluster to the arithmetic average predicted frequency of each of the individual tasks running on the cluster.

2.2.3 Evaluation

We evaluate the effectiveness of STEER by comparing it to multiple state-of-the-art schedulers on an asymmetric cluster-based platform NVIDIA Jetson TX2. The evaluation across a range of benchmarks shows that STEER achieves 53% energy reduction on average compared to the baseline scheduler greedy random work stealing (GRWS), and 38% energy reduction on average compared to both the state-of-the-art approach AEQUITAS [22] and our previous work ERASE. Our analysis indicates that STEER can identify the best task schedule and core frequency settings for different task types, therefore leading to the lowest energy consumption.

We also evaluate the prediction accuracy of the proposed performance and power models in STEER and use MAPE (Mean Absolute Percentage Error) as a measure of accuracy. The results show that the proposed performance model and power model achieve 95% (92% in the worst case) and 90% (86% in the worst case) prediction accuracy on average, respectively. Furthermore, our evaluation demonstrates that the fraction of the overall time spent on the sampling and prediction phase is 0.73% on average. The overheads of running the workflow of STEER task scheduler to figure out the best configurations are 0.1% on average on all six cores across all evaluated benchmarks.

2.2.4 Conclusion

We propose STEER to address the problem of reducing the CPU energy consumption of task-based parallel applications on cluster-based multi-core platforms. STEER leverages static asymmetry, dynamic asymmetry and task heterogeneity in conjunction to run each task with the least possible energy. STEER incorporates a performance model and a power model to predict the impact of running tasks with different execution places and frequency settings. It also utilizes heuristics to manage varying task granularity and reduce frequency-tuning conflicts among concurrent running tasks.

2.3 Paper-III

2.3.1 Background

Energy efficiency has emerged as a crucial design constraint in various parallel computing systems ranging from battery-powered mobile devices to high performance computers. There has been extensive research on leveraging static and/or dynamic asymmetry as knobs to reduce CPU energy consumption since it is typically the largest contributor to the total energy consumption of a system [19, 21, 22, 24, 25, 46–48]. In order to meet the memory demand for emerging many-core architectures, main memory bandwidth and capacities have also been steadily increasing, which lead to the memory system also becoming a major contributor to total energy consumption [49]. Accordingly, several emergent architecture designs provide DVFS in memory subsystem for different performance and efficiency requirements.

energy-efficient execution of a task DAG relies on runtime schedulers to map each task in the DAG to hardware resources (i.e. choosing the appropriate core type and number of cores for the task) and throttle the available DVFS knobs simultaneously. Unfortunately, existing works do not leverage static and dynamic asymmetry, especially memory DVFS, in conjunction with task characteristics for scheduling. Moreover, they are designed for a specific target and mostly use a set of heuristics without having the ability to explore trade-offs between performance and energy consumption.

In order to develop a runtime scheduling framework that leverages the aforementioned knobs and provides the ability to target various trade-offs between performance and energy consumption, several challenges need to be addressed. First, the effects of tuning available knobs, individually and in conjunction, on performance and energy consumption need to be understood. Second, the interplay between task scheduling decisions and the exploration of various trade-offs between energy and performance needs to be investigated. Finally, there is a need to accommodate applications/tasks with diverse characteristics while also ensuring low runtime overhead.

2.3.2 Proposed Approach

In Paper III, we propose JOSS, a runtime scheduling framework that can target various energy performance trade-offs through leveraging all the knobs (i.e. core type (T_C), number of cores (N_C), core frequency (f_C) and memory frequency (f_M)). Figure 2.4 provides a high-level overview of the JOSS runtime framework.

JOSS specifically comprises a performance model, a CPU power model and a memory power model to provide the scheduler with the prediction of task execution time and power consumed in CPU and memory domains when executing tasks with different configurations. Energy and performance estimates are used by the task scheduler to guide configuration selection and achieve the desired trade-off goals. The scheduler maps tasks to selected CPU cores and sends the frequency throttling requests to the CPU DVFS controller and the memory DVFS controller. JOSS supports performance constraints specified as speedups relative to the execution time for energy minimization.

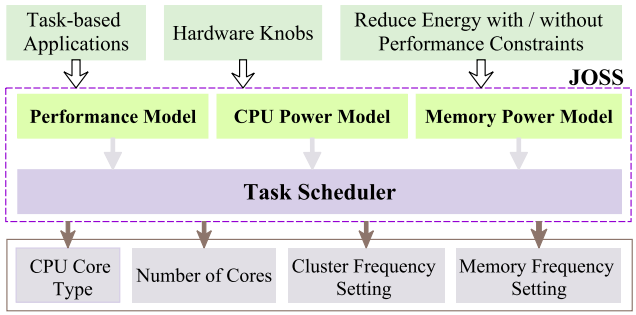

Figure 2.4: High-level overview of JOSS.

Otherwise, JOSS targets total energy reduction if a performance constraint is not specified.

To enable performance and power predictions, we first characterize a platform by running a set of synthetic benchmarks. The basic structure of the synthetic benchmark includes a computation loop and a memory access loop. Through controlling the number of iterations in each loop, we can generate different ratios of computations and memory access. We profile the platform via executing these synthetic benchmarks at all possible configurations for the four knobs and measure the execution time, CPU and memory power consumption.

Performance Model: The performance model aims to predict the execution time of a task under joint CPU and memory frequency scaling. The total execution time is estimated as the sum of computation time and stall time due to memory latency: $Time = Time_{comp} + Time_{stall}$. JOSS uses memory-boundness (MB) as a metric for task characteristic to quantify the fraction of execution time that does not scale with core frequency, similar to prior work [45]. When throttling the core frequency from f_C to f'_C, the computation time will scale proportionally with frequency:

$$Time'_{comp} = Time \times (1 - MB) \times \frac{f_C}{f'_C} \tag{2.4}$$

$\text{Time}'_{\text{stall}}$ is dependent on core and memory frequency scaling in addition to task characteristics (MB). We utilize the statistics from running synthetic benchmarks to build the performance model using multivariate polynomial regression as shown below:

$$Time'_{stall} = Time \times (\sum_{i=0}^{2} \beta_i x_i + \sum_{i=0}^{2} \beta_{ii} x_i^2 + \sum_{i=0}^{1} \sum_{k=i+1}^{2} \beta_{ik} x_i x_k + \varepsilon) \tag{2.5}$$

where $x_i = \{MB, \frac{f_C}{f'_C}, \frac{f_M}{f'_M}\}$, $0 \leq i \leq 2$. Here, $\beta_i, \beta_{ii}, \beta_{ik}$ are the coefficients of the linear component, the quadratic component and the interaction component, respectively, and ε is the intercept. The execution time at $< f'_C, f'_M >$ is $Time' = Time'_{comp} + Time'_{stall}$.

JOSS relies on sampling task execution times at two different core frequencies

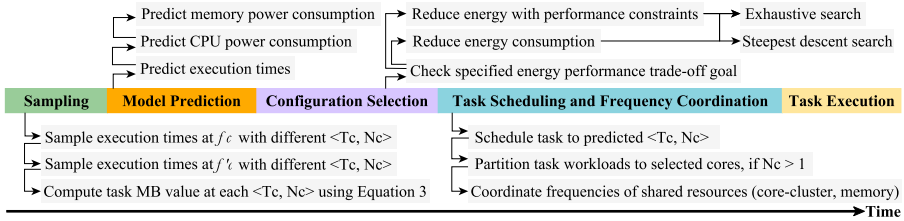

Figure 2.5: JOSS task scheduler timeline.

under the highest memory frequency setting to calculate MB as follows:

$$MB = \frac{\frac{Time_{f'_C}}{Time_{f_C}} - \frac{f_C}{f'_C}}{1 - \frac{f_C}{f'_C}} \tag{2.6}$$

CPU Power Model: The results from running synthetic benchmarks indicate that CPU power consumption is mainly dependent on core frequency and task characteristics (MB). Consequently, we build the CPU power model using multivariate polynomial regression as shown below:

$$Power_C = \sum_{0}^{1} \beta_i x_i + \sum_{0}^{1} \beta_{ii} x_i^2 + \beta_{01} x_0 x_1 + \varepsilon \tag{2.7}$$

where $x_i = \{MB, f_C\}$, 0≤i≤1.

Memory Power Model: Memory power is dependent on all three influential factors, i.e. core frequency scaling, memory frequency scaling and task characteristics (MB). Consequently, we build the memory power model using multivariate polynomial regression as shown below:

$$Power_M = \sum_{0}^{2} \beta_i x_i + \sum_{0}^{2} \beta_{ii} x_i^2 + \sum_{i=0}^{1} \sum_{k=i+1}^{2} \beta_{ik} x_i x_k + \varepsilon \tag{2.8}$$

where $x_i = \{MB, f_C, f_M\}$, 0≤i≤2.

Figure 2.5 provides an overview of the JOSS task scheduler's timeline. JOSS first samples task execution times, for different kernels, when running with different $<T_C, N_C>$ configurations at both f_C and f'_C and then uses the MB values for predicting performance and power at different $<f_C, f_M>$. In configuration selection, JOSS targets reducing the total energy consumption by running each task with the lowest energy possible. JOSS utilizes predictions to determine the configuration that satisfies the desired energy performance trade-off for each kernel. To prune the large search space formed by the four knobs, we also introduce a heuristic search algorithm based on the steepest descent method. In the task scheduling and frequency coordination phase, we schedule tasks according to the scheduling decision and partition the workload, and average the pre-determined frequency setting for the task with the current frequency setting of the shared resources.

2.3.3 Evaluation

We evaluate the effectiveness of JOSS by comparing it to several state-of-the-art schedulers under two scenarios: (1) reducing the total energy consumption; (2) reducing the total energy consumption with user specified performance constraints with respect to (1).

In scenario 1, the results show that JOSS achieves 40.7% energy reduction on average compared to the baseline GRWS, and achieves 21.2% energy reduction compared to STEER (the best among the state-of-the-art). Even in the absence of memory DVFS knob, JOSS_NoMemDVFS (without leveraging the memory DVFS knob and the memory frequency is fixed at max. value) achieves a 24.8% reduction in energy consumption compared to GRWS, which is still an improvement over the state-of-the-art (e.g. 5.2% additional savings than STEER). This emphasizes the importance of taking the total energy consumption into account even when the memory DVFS knob is unavailable.

In scenario 2, the performance and energy results of three performance speedups with respect to JOSS targeting energy reduction solely demonstrate that JOSS can achieve the desired trade-off targets in most of cases. In a few cases, the ability to achieve the desired trade-off targets is impacted by the accuracy of the prediction models, or limited by the processor capabilities, such as peak FLOPS and memory bandwidth.

2.3.4 Conclusion

We propose JOSS, a runtime scheduling framework that leverages both CPU DVFS and memory DVFS in conjunction with core asymmetry and task characteristics to enable energy-efficient execution of task-based applications. JOSS also enables the exploration of energy and performance trade-offs by supporting user-defined performance constraints. It uses a set of models to predict task execution time, CPU and memory power consumption, and then selects the configuration for the tunable knobs to achieve the desired trade-off.

2.4 Paper-IV

2.4.1 Background

Multi-core processors accommodate several hardware features, such as core asymmetry, CPU DVFS and memory DVFS, to improve the energy efficiency of parallel computing systems. The knobs available in hardware provide an opportunity to execute applications in a wide range of settings and enables exploration of different trade-offs between performance and power/energy consumption. The metric used for assessing the trade-offs can be a combined objective, such as Energy Delay Product (EDP) that weighs energy savings and performance equally. Alternatively, its variations E^mD^nP could be used to prioritize either energy savings or performance. In this paper, we target task-based parallel applications, which are modeled as task DAGs. In a task DAG, tasks exhibit different characteristics. Independent tasks can be executed in parallel (inter-task parallelism), and a single task can further support moldable task execution (intra-task parallelism).

Work stealing is a popular task scheduling technique employed in many task-based runtimes. It is effective with sufficient inter-task parallelism available and can scale to large core counts while still ensuring load balancing even on asymmetric architectures. Work stealing is, however, not a good fit for energy performance trade-off (EPTO) exploration since it does not use any DVFS knobs, is unaware of the task characteristics and more importantly, it does not perform well with a low degree of inter-task parallelism. Prior works that focus on energy-efficient runtime scheduling techniques on top of work stealing propose heuristic-based and model-based technique to improve the energy efficiency. Unfortunately, these proposals use a subset of knobs and/or target a specific EPTO metric and lack the flexibility to explore various EPTO metrics.

The goal of the paper is to design a task scheduler that leverages architectural knobs (core asymmetry, CPU DVFS and memory DVFS) and application attributes (inter-task parallelism, intra-task parallelism and task characteristics), to facilitate various EPTO explorations on multi-core architectures. To achieve this goal, several challenges need to be addressed. First, numerous factors influence EPTO, e.g. the choice of core type, the number of cores used to run a task, the amount of tasks to run concurrently and the choice of DVFS settings. Understanding the implications of tuning each knob individually and their interplay is complex. Second, assuming that the impact of the aforementioned knobs can be estimated, the challenge of determining an optimal schedule for a dynamically unfolding DAG on asymmetric architecture still needs to be addressed. Finally, the scheduler should offer flexibility to target different EPTO metrics based on specific requirements.

2.4.2 Proposed Approach

We propose SWEEP that leverages aforementioned knobs and enable the various EPTO explorations for efficient execution of task-based parallel applications. Figure 2.6 provides a high-level overview of SWEEP. The inputs are the task-based parallel application, the supported settings for the different architectural knobs (number of clusters, number of cores in each cluster and

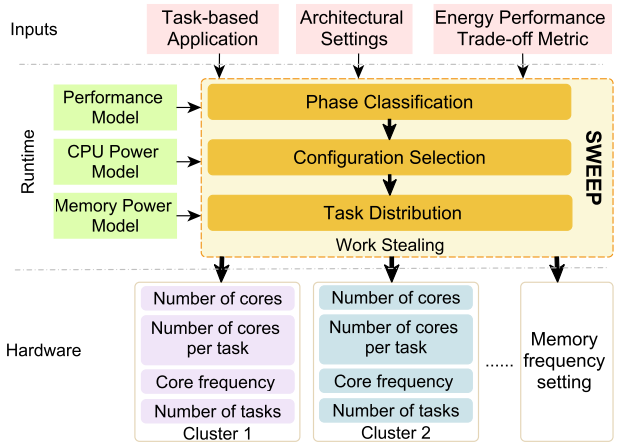

Figure 2.6: A high-level overview of SWEEP.

available frequency settings), and a specific EPTO target metric. SWEEP then determines the task schedule for each cluster and the appropriate CPU and memory DVFS settings.

In a nutshell, SWEEP is a heuristic scheduler that combines model-based prediction with adaptive task distribution algorithms. It works by splitting application execution into phases, classifying each phase into two types - high parallelism (HP) and low parallelism (LP), based on the instantaneous inter-task parallelism and applying different task distribution algorithms for each type of phase. The rationale behind using separate algorithms is that LP phase has less inter-task parallelism, and inadvertently allowing task stealing in LP phase can result in substantial load imbalance that drastically impacts performance.

The task distribution algorithms work by comparing different possible schedules to determine the best schedule, based on the target metric. Due to the computational complexity of comparing all possible schedules, SWEEP uses step-wise heuristic algorithms to narrow the search space and determine the best schedule with low overhead.

High Parallelism: SWEEP implements a four-step algorithm to determine the task schedule and the DVFS settings for HP phases. In the first step, SWEEP determines the number of cores to be used in each cluster for executing an individual task. It chooses to exploit intra-task parallelism only if executing with multiple cores provides superlinear speedup. The second step decides the number of cores to be used in each cluster for concurrent task execution. This is done by looping through the combinations of using different number of cores to determine the best schedule that leads to the lowest predicted trade-off. The third step identifies the best frequency settings for each cluster and memory for executing tasks. SWEEP utilizes another heuristic frequency selection algorithm to prune the large search space. It work by comparing the trade-off value of a data point against all its immediate neighbors (frequency settings) and repeating this search process iteratively until it converges at a point with the lowest predicted trade-off. The final step determines the task distribution

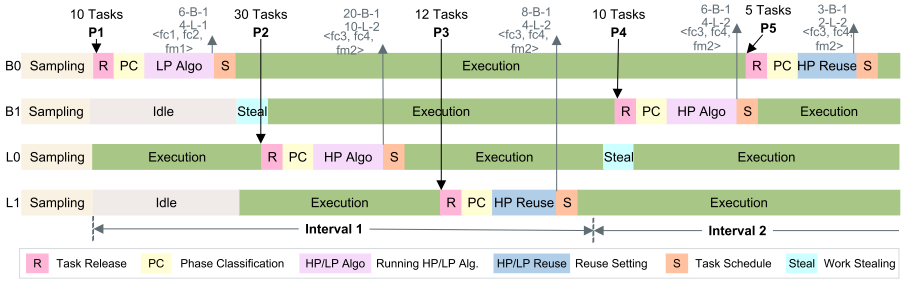

Figure 2.7: SWEEP execution timeline. Note that the timeline is not to scale.

across clusters that achieves load balancing under the settings determined in previous steps.

Low Parallelism: In contrast to an HP phase, it is difficult to achieve load balancing and keep all cores utilized in an LP phase, due to the limited number of ready tasks. SWEEP implements a two-step algorithm to determine the task schedule and DVFS settings in an LP phase. The first step determines the number of cores per task, the number of cores per cluster and the task distribution by looping throught the possible combinations and jointly evaluating their impact on the trade-off target to enable detailed exploration of their interplay. The second step identifies the best frequency setting for CPU and memory that further reduces the trade-off of the phase using the same frequency identification algorithm discussed in HP algorithm.

In addition, SWEEP utilizes an interval-based update approach to further reduce computational overhead. SWEEP runs the algorithms and computes the configurations once per time interval. During subsequent phase invocations within the same time interval, the configurations determined earlier for the HP and LP phases are reused. SWEEP updates the configurations for HP and LP by running the algorithms again in the next time interval. Figure 2.7 provides an example of the SWEEP execution timeline on a four-core system comprising two big and two little cores.

To facilitate task schedule determination and DVFS tuning, the algorithm requires knowledge of per-task execution time and power consumption under different settings. SWEEP employs a set of predictive models, constructed with polynomial regression technique, similar to those proposed in JOSS, to predict performance and CPU and memory power consumption when tuning the hardware knobs for individual tasks (sampling in Figure 2.7).

2.4.3 Evaluation

We evaluate SWEEP by comparing it with several state-of-the-art schedulers under three different EPTO metrics: EDP that weighs energy consumption and performance equally and other two EPTO metrics that prioritizes performance (ED^2P) and prioritizes energy savings (E^2DP) respectively. This allows us to explore the flexibility and effectiveness of different schedulers when targeting different EPTO metrics. In addition, we evaluate two variants of SWEEP to study its adaptability in the absence of DVFS knob (SWEEP_N.F) and in the presence of just the CPU DVFS knob (SWEEP_C.F).

When targeting EDP, SWEEP achieves 19.9% EDP reduction on average compared to the best performing state-of-the-art scheduler and 28.3% reduction than the baseline greedy random work stealing scheduler (GRWS) across all evaluated benchmarks. SWEEP_N.F achieves 15.5% and 23.7% EDP reduction compared to GRWS and ERASE respectively, which do not employ any DVFS knobs. Similarly, SWEEP_C.F outperforms AEQUITAS [22] and STEER by 110% and 108% respectively, which only use CPU DVFS knob.

SWEEP outperforms the best performing state-of-the-art JOSS* (the best performing JOSS scheduler with different performance constraints) by 36.4% and is 40.8% better than the second best GRWS on average in terms of ED^2P. For E^2DP, SWEEP outperforms the best scheduler JOSS* by 9.5% and outperforms the second best ERASE by 38.5%, on average.

2.4.4 Conclusion

We propose SWEEP, a task scheduler that leverages architectural knobs (core asymmetry, CPU DVFS and memory DVFS) and application attributes (inter-task parallelism, intra-task parallelism and task characteristics) to facilitate EPTO exploration. SWEEP uses a combination of models and heuristics and works by splitting application execution into HP and LP phases. It uses an adaptive task distribution algorithm, specific to the phase type, that leverages model-based predictions to determine the best task schedule and the DVFS settings for that phase. Moreover, SWEEP is able to flexibly target various EPTO metrics, a feature that is not supported by other proposals.

Chapter 3

Concluding Remarks and Future Work

The growing impact of energy on operational cost and system robustness becomes a strong motivation for improving energy efficiency in parallel computing systems, in addition to performance. Multi-core platforms are equipped with features to enable energy-efficient computing, such as core asymmetry and DVFS in CPU and memory subsystems. Task-based parallel programming models have been shown to be a powerful approach for developing parallel applications, allowing developers to express parallelism in the form of tasks and their dependencies.

To achieve energy-efficient execution of a task-based application, the runtime scheduler plays a crucial role. It necessitates harnessing both architectural knobs and application characteristics, ensuring each task is mapped to suitable hardware resources while simultaneously managing the available DVFS knobs. Moreover, the scheduler should offer adaptivity to target different energy performance trade-off metrics based on specific requirements. Furthermore, certain knobs may be beyond user control or unavailable, emphasizing the importance of designing a scheduler that adapts to diverse platform capabilities. This leads to the investigation of adaptive task scheduling and resource management techniques for improving energy efficiency of executing task-based applications on multi-core architectures. This thesis proposes four schedulers - ERASE, STEER, JOSS and SWEEP - to address the problem in different optimization targets and contexts.

ERASE targets CPU energy reduction when running fine-grained tasking applications on multi-core platforms wherein the DVFS is externally controlled. The scheduler leverages the insights of task moldability and task-type awareness and utilizes online performance modeling and power profiling to estimate the CPU energy consumption and figure out the appropriate resource allocation for each task. Moreover, it can quickly detect external frequency changes and adaptively schedule tasks with low overheads.

STEER utilizes an integrated scheduling strategy that can effectively manage multiple forms of asymmetry (incl. core asymmetry, CPU DVFS and task heterogeneity) for CPU energy reduction while targeting cluster-based multi-core architectures. STEER leverages two predictive models for predicting the

execution time and CPU power consumption and a task scheduler for determining the execution place and frequency settings that consume the least energy and schedule tasks for execution. Furthermore, it applies adaptive scheduling techniques on various task granularity to manage the DVFS overheads' impact on energy, and coordinates the frequency settings to reduce frequency throttling interference within clusters.

JOSS is a task scheduler that leverages an additional memory DVFS knob, in conjunction with core asymmetry, CPU DVFS and task characteristics to further improve energy efficiency. JOSS introduces a set of models built by multivariate polynomial regression to investigate the impact of tuning all these knobs on both performance and power consumption and facilitate the energy consumption prediction and select the best task schedule and DVFS settings. Moreover, the scheduler supports specified performance constraints with respect to the case that targets energy minimization thereby enabling exploration of energy performance trade-offs.

SWEEP task scheduler is designed to adapt to various energy performance trade-off metrics flexibly. It leverages the architectural knobs (core asymmetry, CPU DVFS and memory DVFS) and the application attributes (inter-task parallelism, intra-task parallelism and task characteristics) to facilitate trade-off exploration. The scheduler first splits application execution into high parallelism and low parallelism phases and then applies different task distribution algorithms for each type of phase to decide the best task schedule and DVFS settings, respectively.

There are several interesting research directions for future research. Firstly, a typical heterogeneous HPC system contains various hardware components, including CPUs and accelerator resources such as GPUs. To improve the energy efficiency of such heterogeneous systems, it is important to propose more advanced scheduling approaches and manage the task allocation to efficiently utilize the available hardware resources. This involves investigating the impact of additional knobs provided by platforms on both performance and energy consumption and building corresponding models. Moreover, it is necessary to enhance the scheduling algorithm to effectively explore configurations in a much larger search space. Furthermore, the scheduler needs to handle multiple versions of kernels targeting CPU and accelerators.

Secondly, a modern system-on-chip typically operates in a thermal-constrained environment, limited by thermal design power. Thus, another interesting direction would be to investigate power budget management techniques to dynamically distribute power budget to different components and adjust the available hardware knobs together with the task characteristics for different energy performance trade-off metrics.

Finally, investigating the benefits of hardware-accelerated task scheduling techniques can be another interesting research direction. For instance, hardware-supported task migration among cores could enhance load balancing, albeit potentially introducing context migration overheads, especially in large-scale machines with numerous number of nodes. In addition, we note that data locality exhibits limited impact on our experimental platform owing to substantial memory bandwidth availability. However, data locality may play a significant role on large-scale systems due to the excessive network traffic and data transmission latency. Therefore, the current work could be improved

with a more data-sensitive allocation strategy supported in hardware, such that tasks sharing substantial amounts of data are scheduled to the same cluster to maximize temporal locality.

Bibliography

[1] T. Simunic, L. Benini, A. Acquaviva, P. Glynn, and G. De Micheli, "Dynamic voltage scaling and power management for portable systems," in *Proceedings of the 38th annual Design Automation Conference*, 2001, pp. 524–529.

[2] E. Rotem, Y. Mandelblat, V. Basin, E. Weissmann, A. Gihon, R. Chabukswar, R. Fenger, and M. Gupta, "Alder lake architecture," in *2021 IEEE Hot Chips 33 Symposium (HCS)*, 2021.

[3] "Nvidia jetson," https://developer.nvidia.com/buy-jetson.

[4] Wikichip, "Nvidia Tegra Xavier," https://en.wikichip.org/wiki/nvidia/tegra/xavier, 2018.

[5] "ODROID XU3," https://developer.arm.com/solutions/graphics-and-gaming/development-platforms/odroid-xu3.

[6] "ODROID XU4," https://magazine.odroid.com/wp-content/uploads/odroid-xu4-user-manual.pdf, 2015.

[7] "Apple A17 Pro Chipset," https://www.gsmarena.com/apple_a17_pro_chipset_appears_on_geekbench_performance_cores_clocked_at_378ghz-news-59897.php, 2023.

[8] J. Doweck, W. Kao, A. K. Lu, J. Mandelblat, A. Rahatekar, L. Rappoport, E. Rotem, A. Yasin, and A. Yoaz, "Inside 6th-generation intel core: New microarchitecture code-named skylake," *IEEE Micro*, vol. 37, no. 2, pp. 52–62, 2017.

[9] T. Singh, S. Rangarajan, D. John, C. Henrion, S. Southard, H. McIntyre, A. Novak, S. Kosonocky, R. Jotwani, A. Schaefer, E. Chang, J. Bell, and M. Co, "3.2 zen: A next-generation high-performance ×86 core," in *2017 IEEE International Solid-State Circuits Conference (ISSCC)*, 2017, pp. 52–53.

[10] M. Frigo, C. E. Leiserson, and K. H. Randall, "The Implementation of the Cilk-5 Multithreaded Language," in *Proceedings of SIGPLAN 1998*, Jun. 1998.

[11] G. Contreras and M. Martonosi, "Characterizing and improving the performance of intel threading building blocks," in *2008 IEEE International Symposium on Workload Characterization*. IEEE, 2008, pp. 57–66.

[12] "Documentation of starpu," https://files.inria.fr/starpu/doc/starpu.pdf, 2014.

[13] OpenMP Architecture Review Board, *OpenMP Application Program Interface. Version 5.0*, OpenMP Architecture Review Board Std., Nov 2018.

[14] P.-É. Polet, R. Fantar, and T. Gautier, "Introducing moldable tasks in openmp," in *OpenMP: Advanced Task-Based, Device and Compiler Programming*, S. McIntosh-Smith, M. Klemm, B. R. de Supinski, T. Deakin, and J. Klinkenberg, Eds. Cham: Springer Nature Switzerland, 2023, pp. 51–65.

[15] M. Pericàs, "Elastic places: An adaptive resource manager for scalable and portable performance," *ACM Trans. Archit. Code Optim.*, vol. 15, no. 2, May 2018. [Online]. Available: https://doi.org/10.1145/3185458

[16] R. D. Blumofe and C. E. Leiserson, "Scheduling multithreaded computations by work stealing," *Journal of the ACM*, vol. 46, no. 5, pp. 720–748, Sep. 1999.

[17] Q. Chen, Y. Chen, Z. Huang, and M. Guo, "Wats: Workload-aware task scheduling in asymmetric multi-core architectures," in *2012 IEEE 26th International Parallel and Distributed Processing Symposium*, 2012, pp. 249–260.

[18] H. Ribic and Y. D. Liu, "Energy-efficient work-stealing language runtimes," in *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '14, 2014.

[19] E. Castillo, M. Moreto, M. Casas, L. Alvarez, E. Vallejo, K. Chronaki, R. Badia, J. L. Bosque, R. Beivide, E. Ayguade, J. Labarta, and M. Valero, "Cata: Criticality aware task acceleration for multicore processors," in *2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 2016.

[20] B. Acun, K. Chandrasekar, and L. V. Kale, "Fine-grained energy efficiency using per-core dvfs with an adaptive runtime system," in *2019 Tenth International Green and Sustainable Computing Conference (IGSC)*, 2019.

[21] C. Torng, M. Wang, and C. Batten, "Asymmetry-aware work-stealing runtimes," in *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, 2016, pp. 40–52.

[22] H. Ribic and Y. Liu, "Aequitas: Coordinated energy management across parallel applications," in *2016 ACM International Conference on Supercomputing*, 06 2016, pp. 1–12.

[23] L. Costero, F. D. Igual, K. Olcoz, and F. Tirado, "Energy efficiency optimization of task-parallel codes on asymmetric architectures," in *2017 International Conference on High Performance Computing Simulation (HPCS)*, July 2017, pp. 402–409.

[24] R. A. Shafik, A. Das, S. Yang, G. Merrett, and B. M. Al-Hashimi, "Adaptive energy minimization of openmp parallel applications on many-core systems," in *Proceedings of the 6th Workshop on Parallel Programming and Run-Time Management Techniques for Many-Core Architectures*, ser. PARMA-DITAM '15, 2015.

[25] M. Han, J. Park, and W. Baek, "Design and implementation of a criticality- and heterogeneity-aware runtime system for task-parallel applications," *IEEE TPDS*, 2021.

[26] Q. Deng, D. Meisner, L. Ramos, T. F. Wenisch, and R. Bianchini, "Memscale: Active low-power modes for main memory," in *Proceedings of the Sixteenth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2011.

[27] H. David, C. Fallin, E. Gorbatov, U. R. Hanebutte, and O. Mutlu, "Memory power management via dynamic voltage/frequency scaling," in *Proceedings of the 8th ACM International Conference on Autonomic Computing*, ser. ICAC '11, 2011, p. 31–40.

[28] V. Sundriyal and M. Sosonkina, "Joint frequency scaling of processor and dram," *J. Supercomput.*, vol. 72, no. 4, p. 1549–1569, apr 2016.

[29] Q. Deng, D. Meisner, A. Bhattacharjee, T. F. Wenisch, and R. Bianchini, "Coscale: Coordinating cpu and memory system dvfs in server systems," in *2012 45th Annual IEEE/ACM International Symposium on Microarchitecture*, 2012.

[30] K. B. Wheeler, R. C. Murphy, and D. Thain, "Qthreads: An api for programming with millions of lightweight threads," in *2008 IEEE International Symposium on Parallel and Distributed Processing*, 2008, pp. 1–8.

[31] S. Kumar, C. J. Hughes, and A. Nguyen, "Carbon: Architectural support for fine-grained parallelism on chip multiprocessors," in *Proceedings of the 34th Annual International Symposium on Computer Architecture*, ser. ISCA '07. New York, NY, USA: Association for Computing Machinery, 2007, p. 162–173. [Online]. Available: https://doi.org/10.1145/1250662.1250683

[32] T. kernel development community, "Energy aware scheduling." [Online]. Available: https://www.kernel.org/doc/html/latest/scheduler/sched-energy.html

[33] D. Brodowski, "Cpu frequency and voltage scaling code in the linux(tm) kernel." [Online]. Available: https://www.kernel.org/doc/Documentation/cpu-freq/governors.txt

[34] J. Eastep, S. Sylvester, C. Cantalupo, B. Geltz, F. Ardanaz, A. Al-Rawi, K. Livingston, F. Keceli, M. Maiterth, and S. Jana, "Global extensible open power manager: A vehicle for hpc community collaboration on co-designed energy management solutions," in *High Performance Computing*, 2017.

[35] "XiTAO runtime," https://github.com/CHART-Team/xitao.git, 2018.

[36] S. Park, J. Park, D. Shin, Y. Wang, Q. Xie, M. Pedram, and N. Chang, "Accurate modeling of the delay and energy overhead of dynamic voltage and frequency scaling in modern microprocessors," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 32, no. 5, pp. 695–708, 2013.

[37] R. Schöne, T. Ilsche, M. Bielert, M. Velten, M. Schmidl, and D. Hackenberg, "Energy efficiency aspects of the AMD zen 2 architecture," *CoRR*, vol. abs/2108.00808, 2021. [Online]. Available: https://arxiv.org/abs/2108.00808

[38] Byung-Jae Kwak, Nah-Oak Song, and L. E. Miller, "Performance analysis of exponential backoff," *IEEE/ACM Transactions on Networking*, vol. 13, no. 2, pp. 343–355, April 2005.

[39] T. Odajima, Y. Kodama, M. Tsuji, M. Matsuda, Y. Maruyama, and M. Sato, "Preliminary performance evaluation of the fujitsu a64fx using hpc applications," in *2020 IEEE International Conference on Cluster Computing (CLUSTER)*, 2020, pp. 523–530.

[40] B. Jeff, "Advances in big.little technology for power and energy savings improving energy efficiency in high-performance mobile platforms," 2012.

[41] "Mediatek x20 development board," https://www.96boards.org/product/mediatek-x20/.

[42] S. Shankland, "iphone xs a12 bionic chip is industry-first 7nm cpu," https://www.cnet.com/news/iphone-xs-a12-bionic-chip-is-industry-first-7nm-cpu/, September 2018.

[43] R. Ritchie, "Apple a14 bionic explained — from ipad air to iphone 12," https://www.imore.com/apple-a14-bionic-explained-ipad-air-iphone-12, September 2020.

[44] Q. Chen, Y. Chen, Z. Huang, and M. Guo, "Wats: Workload-aware task scheduling in asymmetric multi-core architectures," in *2012 IEEE 26th International Parallel and Distributed Processing Symposium*, 2012, pp. 249–260.

[45] B. Goel, *Measurement, Modeling, and Characterization for Energy-efficient Computing*. Chalmers University of Technology, 2016.

[46] M. Endrei, C. Jin, M. N. Dinh, D. Abramson, H. Poxon, L. DeRose, and B. R. de Supinski, "Energy efficiency modeling of parallel applications," in *SC18: International Conference for High Performance Computing, Networking, Storage and Analysis*, 2018.

[47] A. Navarro Muñoz, A. F. Lorenzon, E. Ayguadé Parra, and V. Beltran Querol, "Combining dynamic concurrency throttling with voltage and frequency scaling on task-based programming models," in *50th International Conference on Parallel Processing*, ser. ICPP 2021, 2021.

[48] A. Coutinho Demetrios, D. De Sensi, A. F. Lorenzon, K. Georgiou, J. Nunez-Yanez, K. Eder, and S. Xavier-de Souza, "Performance and energy trade-offs for parallel applications on heterogeneous multi-processing systems," *Energies*, vol. 13, no. 9, p. 2409, 2020.

[49] O. Mutlu, S. Ghose, J. Gómez-Luna, and R. Ausavarungnirun, "A modern primer on processing in memory," in *Emerging Computing: From Devices to Systems*. Springer, 2023, pp. 171–243.