Cache-Aware Energy-Efficient Scheduling on Multicore Real-Time Systems

Thesis Dissertation
Presented by

Saad Zia Sheikh

In Partial Fullfilment
of the Requirements for the Degree of
Doctor of Philosophy in
Electrical Engineering

Supervisor: Dr. Adeel Pasha

Syed Babar Ali School of Science and Engineering
Lahore University of Management Sciences
June 2020

To Zia & Asma (my parents)

Acknowledgements

All praise to Allah who blessed me with the opportunity and ability to do this work.

My parents, no words can describe the sacrifices they have made for my well-being and success. Any achievement I earn is truly a reflection of their dedication and hard work.

Dr. Adeel Pasha is, in every sense, the best supervisor one could hope for. At the start of my PhD journey, I was absolutely clueless about research. However, he took me by the hand and taught me how to recognize existing problems and encouraged me to investigate different research methodologies and solutions. His kindness, patience and excellent character complement his research abilities to make him an outstanding supervisor. On numerous occasions, Dr. Adeel spent sleepless nights so that I could meet my research goals and deadlines. This in itself speaks volumes about his exceptional role and dedication as a supervisor. His optimistic attitude and ability to recognize a silver lining in every obstacle has left an everlasting impression on me. I am very lucky to have had him as my mentor.

I would like to thank my PhD committee members, Dr. Shahid Masud and Dr. Momin Uppal for their valuable feedback and encouragement during my research, and also my dissertation reviewers Dr. Arslan Munir, Dr. Basit Shafiq, Dr. Gogniat Guy, Dr. Hyoseung Kim, Dr. Muhammad Shafique and Dr. Zhishan Guo for evaluating my thesis. Finally, I would like to express my gratitude to the faculty and staff members at LUMS, particularly the electrical engineering department, who were all more than willing to help and guide me whenever I needed it.

A special shout-out to Hasan, Ahmad, Fatima, Balakh, Sikandar, Ahsan, Anas, Aurangzeb, Fahad, Jasim, Sohail, Taimoor and Wahaj, without whom..life would be boring.

Abstract

With the ever-increasing demand for higher performance, the adoption of multicore processors has been a major stepping stone in the evolution of real-time systems. However, despite the increase in computational bandwidth due to parallel processing, scheduling real-time tasks on multicores is not a trivial problem. This scheduling problem is especially aggravated for hard real-time systems where failure to meet task deadlines can be catastrophic. Moreover, the inclusion of shared caches in multicores has increased the unpredictability of the system, and the indispensable interactivity between the hierarchical memory subsystem and multiple cores has further aggravated the already complex Worst Case Execution Time (WCET) analysis of the tasks. Cache partitioning techniques have been proposed as a countermeasure to decouple the shared cache latency from the WCET. However, existing energy-efficient scheduling algorithms are oblivious to the unpredictable nature of shared caches or cache partitioning techniques, thus, diminishing their applicability to real-world systems. Furthermore, a relatively large portion of the processor is occupied by caches contributing to a large percentage of the overall energy consumption. Several general techniques have been proposed to mitigate the energy lost due to caches. However, adopting such techniques into the multicore real-time systems domain has not yet received much attention. This is due to the difficulty of analyzing the impact that core-level energy minimization techniques have on the cache subsystem. Finally, there is now a trend towards heterogeneous multicores where cores on the same processor differ in power, performance, and architectural capabilities. The desired performance and energy consumption is attained by assigning a task to the core that is best suited for it.

In this thesis, we investigate the integration of the cache-partition model into homogeneous and heterogeneous multicore hard real-time systems for system-level energy minimization. We start by investigating a more realistic task model that considers separately the CPU compute cycles and the Memory latency cycles. We then incorporate the impact of caches on independent frame-based tasks running on homogeneous multicores by proposing algorithms for core-level, cache-level and system-level energy optimizations. We then move onto heterogeneous multicores and propose a holistic solution for cache-aware system-level energy minimization while ensuring the schedulability for periodic tasks. Finally, we propose a dynamic cache-partition schedulability analysis for multicore partitioned scheduling.

Contents

List of Figures

List of Tables

List of Acronyms

ICT Information and Communication Technology

CMOS Complementary Metal Oxide Semiconductor

WCET Worst-Case Execution Time

EDF Earliest Deadline First

HPT Higher Priority Tasks

DVFS Dynamic Voltage and Frequency Scaling

IC Integrated Circuits

DPM Dynamic Power Management

CP Cache Partition

VFI Voltage Frequency Island

GA Genetic Algorithm

LP Linear Program

ILP Integer Linear Program

MILP Mixed Integer Linear Program

WFD Worst-Fit Decreasing

MSHR Miss Status Holding Registers

2D Two-Dimensional

3D Three-Dimensional

CDG Cache Dependency Graph

2DSPP Two-Dimensional-Strip-Packing-Problem

THEAM Task-Heterogeneity-Energy Aware Mapping

Chapter 1

Introduction

With the onset of climate change and the steady depletion of energy resources, factors affecting energy consumption have become a global concern. A recent study has shown that contributions to greenhouse emissions by the Information and Communication Technology (ICT) sector alone could be more than 14% by 2040 [10]. Therefore, minimizing energy consumption has become the forefront of research and innovation in many domains, and a cornerstone for sustainability and world-wide economic stability.

Energy efficiency for the computing systems domain, in particular, has received significant interest in recent years due to advancements in Complementary Metal Oxide Semiconductor (CMOS) technology. A computing system is a machine that can be instructed to perform a set of instructions. These instructions are composed in specific sequences to create a wide range of meaningful applications. The instructions, which are stored in a off-chip memory component, are fetched and executed on a processor core.

Since the advent of single-core processors, there has been a constant strive to improve their performance. Continuous reductions in the CMOS feature size has allowed designers to pack an increasing number of transistors into a single chip, thus, enabling circuits to operate at higher frequencies and, therefore, execute instructions at greater speeds. Higher frequencies with an exponential increase in the transistor count, however, had previously forced chip designers into a thermal power-wall, e.g., destroyed Intel's expectations to reach a frequency of 10 GHz for technologies below 90nm [7]. This unexpected complication motivated hardware manufacturers to fabricate chips, with the predicted high number of transistors, as parallel components operating at lower frequencies, thus, entering into the multicore era.

A homogeneous multicore is composed of multiple processor cores of the same type on a single chip. Figure 1-1 is an example architecture of a multicore chip where each core has access to a private Level-1 (L1) cache which is then connected to a Level-2 (L2) shared cache. Caches act as a temporary storage and reduce the memory latency in fetching the instructions/data to and from the off-chip memory component. This multicore setup per-

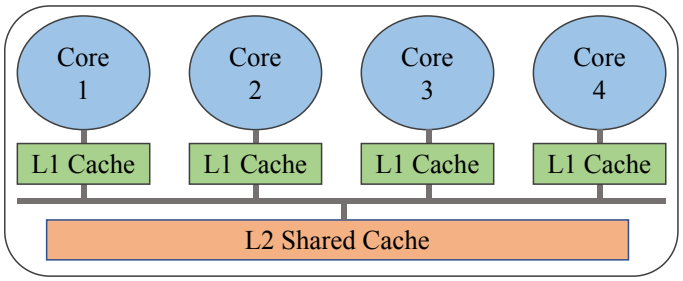


Figure 1-1: Homogeneous multicore architecture

mits multiple applications to be executed on different cores at the same time and, therefore, improves the performance of the system. However, thermal challenges still prevail for multicore systems, and the continuous demand for higher performance must be matched by techniques to reduce the power consumed by these processors.

Another stimulus behind the demand for energy efficiency in computing systems is the desire for portability and increased reliability. Thus, battery-life plays an important role in portable computing devices. However, there is no Moore's law for batteries and the exponential increase in computing power has been shunned by the slow progress of battery technology.

These factors, among many others, have prompted the need for specialized hardware and software solutions to minimize energy consumption. A specific and essential class of computing systems, that significantly relies on software energy-efficient solutions, is Real-Time Systems.

1.1 Real-Time Systems

In contrast to mainstream computing systems, real-time systems are special-purpose systems that are designed to respond to real-world events under strict requirements. Today, real-time systems can be found in virtually any environment, ranging from simple consumer electronics to complex avionics and space exploration applications. Such systems are limited by size, power, and resources. A system is only considered real-time when the correctness of the system depends on both the logical output and the physical instant at which the output is produced.

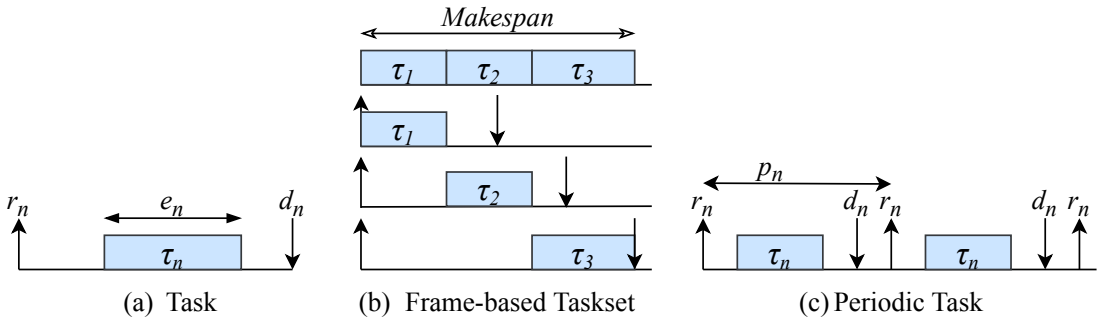


Figure 1-2: Task model in real-time systems

1.1.1 Task-Models

Applications within the real-time systems domain are termed as *tasks*. A real-time system is characterized by a task model. A task τ_n within a task model is defined by various properties as shown in Figure 1-2. Different task models have been used within the domain of real-time systems. Frame-based task models represent single-run tasks defined by a release-time r_n, Worst-Case Execution Time (WCET) e_n, and deadline d_n as shown in Figure 1-2 (a). Figure 1-2 (b) represents a frame-based taskset where the total length of the taskset execution is called a makespan. Periodic task models, on the other hand, define each task with an additional period p_n parameter which describes a task's re-occurrence after its previous release-time as shown in Figure 1-2 (c). Periodic tasks are assumed to execute perpetually where each re-occurrence of a task is defined as a single job. A job's response time is the time from when it was released to when it completes the execution. The response time Res_n of a task is then the longest finish time of all jobs of the task. A job misses its deadline if its response time is greater than its deadline and a task misses its deadline if at least one of the jobs of that task misses its job-deadline. The percentage of total processor time used by a task is defined by its utilization $u_n = e_n/p_n$. The taskset utilization, i.e., total utilization of all the tasks in a taskset, is the sum of utilization individual tasks. The *hyperperiod* of a periodic taskset is the smallest interval of time after which the periodic pattern of all the tasks are repeated and is calculated by taking the least common multiple of the periods of all the tasks. There are other types of tasks aswell, e.g., sporadic tasks where jobs of a task can reoccur at any moment and two successive jobs of a task must be separated by a minimum inter-arrival time, and aperiodic tasks where jobs have irregular arrival time.

1.1.2 Single-Processor Scheduling

Scheduling refers to the process of selecting tasks for execution on processing elements. Scheduling algorithms define the criteria through which specific tasks are selected for scheduling. Scheduling real-time tasks on single-core processors has been a well-investigated subject since the 1960s.

Two primary subject matters emphasized in this domain are: (i) to propose algorithms to schedule different tasksets, and (ii) to derive schedulability tests to ensure that the taskset constraints are met under a specific scheduling algorithm. A schedule can either be *static* or *dynamic*. In static scheduling, all task information is known and scheduling decisions are made at compile time. The predefined schedule is then stored in memory, and at run-time the scheduling algorithm simply reads from the table at appropriate time instants to run a particular task on the processor. This table-driven schedule permits a low run-time overhead but is unsuitable for tasksets with varying workloads where the number of tasks in the system can change at run-time. In dynamic scheduling, all scheduling decisions occur at run-time while the taskset is being executed. Reoccurring tasks, e.g., periodic tasks, are assigned priority numbers. At each scheduling instant (i.e. the time instant when the scheduling algorithm is invoked to select a task to be run by the processor), the scheduling algorithm selects a ready task with the highest priority to be executed. A scheduling algorithms can assign its taskset either static or dynamic priorities. A scheduling algorithm can also be classified as *preemptive* or *non-preemptive*. A scheduling algorithm is preemptive if the execution of a task is suspended due to the arrival of a higher priority task. Similarly, a scheduling algorithm is considered *optimal* if no other schedule can outperform it under a given constraint, e.g., for the energy minimization problem, an optimal schedule will minimize the energy consumption and no other schedule will be able to reduce the energy consumption more than the optimal one. Tasks can also be classified as dependent or independent. The execution of a dependent task relies on the completion of its prerequisite tasks whereas an independent task does not have any such restriction.

There are different types of real-time systems. A soft real-time system permits deadline misses while maintaining a minimum throughput of the system. A hard real-time system, however, has stringent timing requirements where failure to meet any task deadline will result in system failure. Such systems are used in time and safety critical applications.

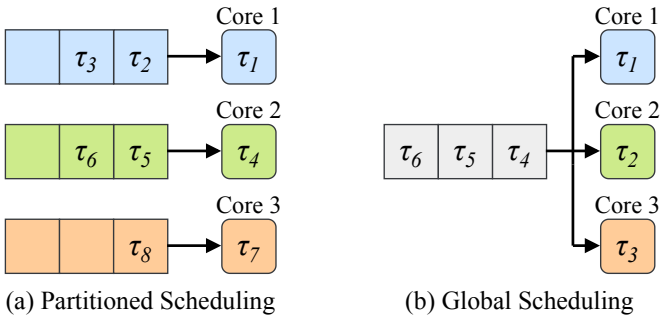


Figure 1-3: Partitioned and global scheduling.

Hard real-time systems will be the focus of this thesis.

Scheduling algorithms can have different utilization schedulability bounds. A taskset is guaranteed to meet all deadline constraints for a particular scheduling algorithm if the utilization of the taskset is less than the schedulability bound of that scheduling algorithm. An example of a preemptive static priority scheduling algorithm is *Rate Monotonic (RM)* where the priority of the task is specified by its period. RM has a schedulability bound of $N(2^{\frac{1}{N}} - 1)$ where N is the number of tasks in the taskset. Preemptive *Earliest Deadline First (EDF)*, on the other hand, is an optimal dynamic scheduling algorithm on single-core processors and has a schedulability bound of 1 [88]. Under EDF, the tasks are dynamically prioritized and scheduled according to their approaching deadlines. Both EDF and RM are work-conserving scheduling algorithms. A work-conserving scheduling algorithm does not allow the processor to be idle if there are tasks ready to be scheduled.

1.1.3 Multicore Processor Scheduling

The continuous demand for higher performance with application variability has led to the adoption of multicore processors in real-time systems. Partitioned scheduling and global scheduling are two primary methods adopted in the literature to tackle this problem [13].

Partitioned scheduling is a static scheduling method for multicore processors. Tasks are assigned to cores at compile time. Each core, then, schedules its assigned tasks independently. Figure 1-3 (a) displays such a scenario where each core has its own ready queue and tasks are partitioned across cores. The partitioned scheduling problem is analogous to the bin-packing problem which is NP-Hard [36]. There are many heuristic solutions, in the literature, that can partition the tasks in polynomial time, e.g., First-Fit (FF),

Best-Fit (BF), First-Fit Decreasing (FFD), etc. The prime advantage of using partitioned scheduling is that it allows each individual core to schedule its assigned taskset via existing, well-established, single-core algorithms. Partitioned scheduling is not work-conserving in nature, however, it is a favorable option for deterministic workloads.

For varying workloads, the best option would be to choose a global scheduler as shown in Figure 1-3 (b). In global scheduling, tasks are dynamically allocated to the cores and are also allowed to migrate among cores at run-time to maintain a workload balance. Global scheduling can theoretically increase the schedulability, i.e., number of tasks that can successfully be scheduled. An example of an optimal global scheduling algorithm for periodic tasks is Pfair [8] where the execution timeline is divided into equal length slots and the scheduler is invoked at the beginning of each slot in order to select a task for execution. In practice, however, such algorithms incur high overheads due to core synchronization constraints and high scheduling overheads. Schedulability tests for global scheduling algorithms are also largely pessimistic since the critical instant of a task is undefined, i.e., it is unknown which specific sequence of its Higher Priority Tasks (HPTs) will lead to its maximum response time. Therefore, an upper-bound on the interference from HPTs must be determined via the *problem window* approach, where a job of a periodic task is assumed to have missed its deadline and the maximum interference from its HPTs, that led to its deadline miss, is deduced and then used to determine the task's schedulability [36]. Furthermore, migration overheads can vary over time and are highly dependent upon the processor architecture. These migrations not only reduce the performance of the scheduler, but also make schedulability tests more difficult to design. Therefore, schedulability tests for global schedulers consider the migration overhead factor as constant or negligible [13].

Semi-partitioned scheduling has also been proposed as a hybrid technique to balance the tradeoff between the under-utilization of partitioned scheduling and the high run-time migration overhead of global scheduling. Semi-partitioned scheduling bounds the number of migrations by either limiting the number of cores that can execute a task or by restricting the migrations to take place only at job boundaries.

1.2 Energy-Efficiency in Real-Time Systems

Along with other domains of computing research, energy-efficiency has become a prime concern for real-time systems [6]. Particularly to hard real-time systems, the goal for energy-efficient real-time scheduling algorithms is to minimize the energy consumed by task executions while ensuring task deadlines are met. Most energy-efficient techniques are based on the power consumption model of the processor which consists of static and dynamic power components [6]. The inherent Dynamic Voltage and Frequency Scaling (DVFS) capabilities of modern processors can be used to decrease the energy consumption. However, decreasing the frequency increases the clock-cycle time which in turn increases execution time of the tasks. Therefore, DVFS techniques carefully utilize the idle processor time between task executions to ensure timely completion of the tasks.

1.2.1 Power Reduction Techniques

Today the majority of Integrated Circuits (ICs) are fabricated using CMOS technology. Its low power consumption and high noise immunity features make it a favorable choice for modern IC design. The power-consumption of a CMOS logic gate is composed of dynamic, short-circuit, and static power consumption components, which can be modeled respectively as:

$$P = \kappa V_{dd}^2 C f + I_{short} V_{dd} + I_{leak} V_{dd} \tag{1.1}$$

where κ is the transistor switching factor, V_{dd} is the supply voltage, C is the gate capacitance, f is the clock frequency, I_{short} is the instantaneous surge current flowing during CMOS switching activity and I_{leak} is the leakage current.

Recognizing the static and dynamic dimensions of the CMOS power consumption model, early research efforts attempted to address these two entities independently to reduce the power consumption. This resulted in two distinct energy saving techniques, i.e., DVFS (for dynamic power) and Dynamic Power Management (DPM) (for static power). The short-circuit current component is negligible compared to the others and is usually ignored.

The maximum operating frequency for a specified voltage is related to the circuit delay t as: $f \propto \frac{1}{t} \propto \frac{(V_{dd}-V_{th})^\gamma}{V_{dd}}$, where V_{th} is the threshold voltage and γ $(1 \leq \gamma \leq 2)$ is the velocity

saturation. The power equation can be simplified if the voltage-frequency operating points are fixed, resulting in an approximated model:

$$P = \kappa f^{\alpha} + k_{s1} f + k_{s2} \tag{1.2}$$

where $\alpha \geq 2$, and k_{s1} and k_{s2} are static power dependent constants. Both Eqs.(1.1) and (1.2) have been used extensively in the literature to model the power consumption of the processor core [34, 45, 113].

1.2.2 Dynamic Voltage and Frequency Scaling (DVFS)

Modern processors are equipped with a clock-gating technique that varies the frequency of the processor core between a minimum and maximum frequency bound. Such a setup requires a programmable voltage regulator and clock generator [96]. The frequency (f) in the dynamic component of Eq. (1.1) has a direct relation with the operating voltage (V_{dd}). Therefore, decrease in voltage and a relevant decrease in frequency often go hand-in-hand resulting in a cubic reduction in dynamic power consumption [2].

However, decreasing the frequency of the processor core increases the execution time of the tasks by elongating the core cycle time while the number of execution cycles remain constant. DVFS techniques use this available processor capability to utilize the core idle time periods between scheduled tasks. The literature usually assumes a linear relationship between the execution time of a task and the core frequency. This linearity can be modeled through the equation $s = f_{max}/f$, where f_{max} is the maximum frequency of the core and s is the task scaling factor, i.e., the resultant WCET of a task τ_i after changing the core frequency to f will be $e_i \times s$.

DVFS can be categorized into static and dynamic techniques. In the static DVFS technique, the *slack*, i.e. the idle time between task executions, is identified at compile time and tasks are then stretched to minimize the slack. The core frequency is usually kept at a constant value during the execution of a single task. This can be seen in Figure 1-4 (a, b) for 2 frame-based tasks with different release times and deadlines, and where the tasks are stretched according to the amount of slack available to each task. There have been considerable achievements in this area ever since one of the earliest studies performed by Yao et al. [124] where the authors proposed an optimal static DVFS algorithm for frame-based

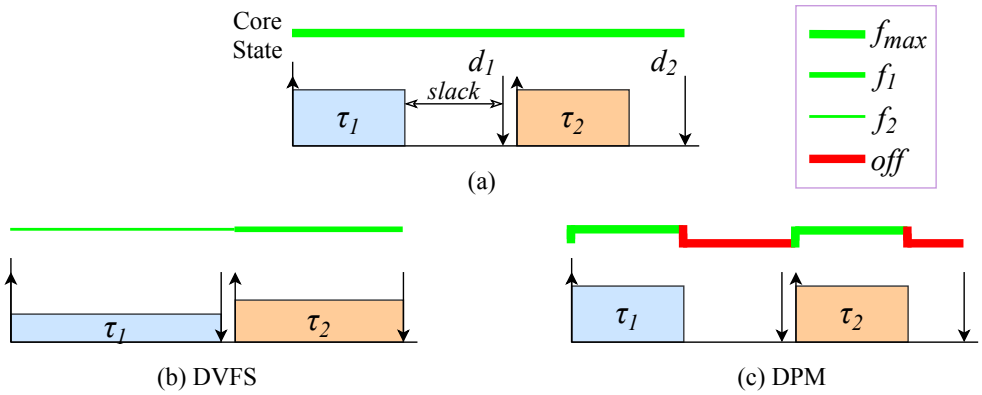


Figure 1-4: Effect of DVFS and DPM on tasks. Core State represents the activity of the core during task execution. (b) More slack for τ_1 permits a lower DVFS setting than that of τ_2. (c) Tasks execute at maximum frequency but the core is in sleep mode during periods of inactivity

tasks executing on a single-core processor. In addition, optimal static DVFS algorithms have also been proposed for periodic tasksets scheduled via the EDF dynamic scheduling algorithm [4]. However, for static priority scheduling algorithms, the energy minimization problem becomes NP-Hard [125].

Furthermore, the frequency is changed at discrete steps. Though impractical, most of the reported literature simplify the problem by considering frequencies to be a continuous function between the minimum and maximum frequency bound. Ishihara et al. [57] showed that, for a system with discrete frequencies, selecting two frequencies adjacent to the optimal desired continuous frequency leads to energy minimization. Frequency transitions also incur a time and energy cost. In practice, this overhead is comparable to the context switching cost in a multitasking environment [96] and is, therefore, usually ignored in most energy minimization works.

Real-time tasks can also experience large variations in their Actual Execution Time (AET). Dynamic DVFS techniques utilize the slack yielded, at run-time, when a task completes before its estimated WCET. There are primarily two ways of utilizing this slack: (i) *Inter-task DVFS* and (ii) *Intra-task DVFS*. Inter-task DVFS redistributes the available slack across other tasks within the system or to a single highest priority ready task [15]. On the other hand, Intra-task DVFS utilizes the slack produced by a task within the same task [107].

However, DVFS techniques only contribute in the reduction of dynamic power consump-

tion, and though dynamic power minimization techniques may have previously been favored, continuous reduction in transistor size has resulted in increased leakage current (I_{leak}) that now makes the static power consumption a significant factor as well. Consequently, lowering the core frequency to its minimum no longer guarantees energy minimization since a lower frequency elongates the computation time that in turn increases the leakage energy [61]. To this effect, *critical frequency* has been defined to limit the amount of task scaling. Setting the frequency of the processor core below its critical frequency will increase the energy consumption of the system. Therefore, for all practical purposes, this factor also needs to be taken into consideration when performing DVFS.

1.2.3 Dynamic Power Management (DPM)

DPM techniques employ power-gating to switch the processor core into sleep mode whenever it is idle as shown in Figure 1-4 (c). DPM can also be categorized into static and dynamic techniques. Static DPM techniques calculate, at compile-time, the exact durations at which the core should be in sleep mode. The more recent processors are equipped with multiple sleep modes. However, switching to sleep mode and restoring back to idle mode incurs a time and energy overhead cost that keeps on increasing further with every deeper sleep mode. Therefore, a particular sleep mode is selected only if the transition reduces the overall energy consumption. This decision can be made using the model proposed by Devadas et al. [40] where the authors defined the *break-even time* as the minimum length of idle period that would justify the core transition to a particular sleep mode.

To further reduce energy consumption, the idle periods can be combined to enable the core into longer and deeper sleep modes. This can primarily be accomplished in two ways. The first method is based on procrastination techniques that maximize the length of a particular idle period by delaying the upcoming tasks while ensuring that the timing constraints are not violated [76]. The second method is to maximize the processor speed during a task's execution to complete it at the earliest; thus, leaving a room for longer idle periods [56].

In Dynamic DPM techniques, the durations of sleep modes are estimated at run-time. One of the challenges in performing dynamic DPM is to determine the exact remaining idle time upon early task completion. Determining this idle time can be classified into predictive and stochastic schemes [14]. An interplay of DVFS and DPM techniques to find the right

mix of static and dynamic energy savings has also been proposed in the literature [32,40,46].

1.2.4 Energy Efficiency in Multicore Real-Time Systems

The decades of research on energy-efficient scheduling techniques, primarily focusing on single-core processors, has been extended into the multicore domain [6]. Energy-efficiency adds another dimension to this multicore scheduling problem. Given the complexity of the problem, the widely-adopted bottom-line approach is to discretize the problem into two phases. The first phase allocates the tasks to cores in a manner to promote maximum energy savings, while the second phase takes advantage of this strategic allocation to effectively minimize the energy consumption via DVFS or DPM.

1.3 Existing Limitations

1.3.1 Linear Task-Models

Most energy-efficient scheduling algorithms reported in the literature are handicapped by the simplistic assumption of linear task scaling. Such algorithms model the execution time of a task as homogeneous clock cycles executed at a particular core-frequency. However, these models fail to capture the memory latency experienced by a task during execution. Since this memory latency does not scale with core-frequency, such linear models can lead to inaccuracies and overestimiation of task execution time when DVFS is applied. The need to capture such non-linearities has prompted researchers to propose improved execution models that separate memory latency from the processor clock cycles [90], resulting in models that exhibit two sets of execution cycles, i.e., computation cycles and memory cycles. In such models, the core-frequency can only scale the computation cycles while the memory cycles are only affected by the operating clock of the bus and memory sub-system [44,126]. However, the benefits of using this model for multicore hard real-time systems is yet to be determined.

1.3.2 The Dilemma of Shared Caches

Shared caches, a consequence of multicore architectures, pose to be a significant challenge for real-time systems scheduling. Earlier on, caches were introduced to keep up with the performance demand by bridging the gap between speeds of the core and main memory.

Caches temporarily store information required by the core to process the executing task. However, this has also resulted in increased unpredictability since cache hits or misses can vary the number of memory latency cycles. Predicting cache behavior is a notable obstacle for WCET analysis tools. The small cache sizes result in inter-task and intra-task cache-line evictions [106], thus, contributing to the erratic nature of caches. Due to this versatile nature, caches have the biggest influence on task execution time and, therefore, have a considerable impact on the precision of the WCET estimation [86].

To ensure predictability on single-core processors, a pessimistic approach is usually adopted by incorporating the upper bound of the cache delay into the WCET of the tasks. Despite the decades of research on single-core WCET analysis, existing techniques cannot be directly extended into the multicore domain due to inter-core conflicts introduced by the inclusion of shared caches. Useful cache lines of a task may be evicted from the shared cache by a task simultaneously running on another core, thus, dynamically increasing the task miss-rate and system-level unpredictability. Inclusion of shared caches may also have adverse effects on the execution time of a task as shown by Kim et al. [66], where the execution time of a task was increased by 40% as opposed to when running alone on the system. Thus, accurate analysis of inter-core conflicts has been termed as an extremely difficult problem by the WCET analysis community [48]. In addition, replacement policies and cache coherence protocols must also be considered.

Because of this unpredictable nature of shared caches, potentially-assumed schedulable tasksets may experience deadline misses. The real-time systems community has attempted to bound this shared cache unpredictability by adopting cache-partitioning techniques [47]. These partitions are then assigned to cores to prevent concurrent tasks from sharing the same cache lines, thus, diminishing inter-core conflicts. Cache Partitions (CPs) can be statically (core-based) or dynamically (task-based) assigned to the cores. However, such techniques must be considered carefully since heedless allocation of CPs can lead to under-utilization of resources. Though shared CP analysis has been tackled rather assiduously in terms of maximizing schedulability [67, 121, 123], its impact on energy-efficiency techniques is yet unknown.

Since predictability is a major concern for real-time systems, energy-efficient scheduling algorithms cannot ignore these caching effects and will otherwise diminish their applicability to practical systems. Existing energy-efficient algorithms cannot be adapted directly to

the static CP-aware scenario since this would require a completely different task-to-core allocation strategy according to the limited CPs assigned to each core. Directly adapting existing algorithms in a dynamic CP scheme, without modeling the dynamic inter-core cache contention, can result in cache violations, i.e., concurrent access of the same CPs by several tasks running in parallel. Therefore, careful analysis needs to be carried out to incorporate the CP scenario into the energy-efficient scheduling domain.

Furthermore, most of the existing energy-efficient scheduling algorithms only focus on the core-level energy minimization problem. A relatively large portion of the processor is occupied by caches contributing to a large percentage of the overall energy consumption. On-chip caches consume 30 % of total power on the StrongARM processor core and nearly 24 % on the Niagara processors [91]. With the continuous increase of cache size and the involvement of multicores and manycores, these numbers are likely to grow. Several general techniques have been proposed to mitigate the energy lost due to caches [91]. However, adopting such techniques into the real-time system's domain has not yet received much attention. There have been some limited attempts to accommodate the cache energy while minimizing the system-wide energy consumption. However, most of them are focused on simpler single-core systems [60, 70, 113, 130].

1.4 Thesis Statement

As real-time systems shift into the multicore domain, there is a need for energy-efficient scheduling algorithms to include the shared last-level-cache energy consumption into their minimization problem while catering for the unpredicatbility caused by the shared-cache.

1.5 Thesis Contribution

Though there has been considerable research on energy-efficient scheduling in multicore hard real-time systems, existing solutions only focus on the core-level and fail to include the shared cache energy and unpredictability into the energy minimization problem. Furthermore, the impact that cache partitioning has on execution cycles of a task in a energy minimization setting has yet to be investigated. These existing limitation are summarized in Figure 1-5, where the Venn diagram represents the complexity of the research problem. Thus, the goal of this thesis is to overcome these limitations and in doing so, pave the way

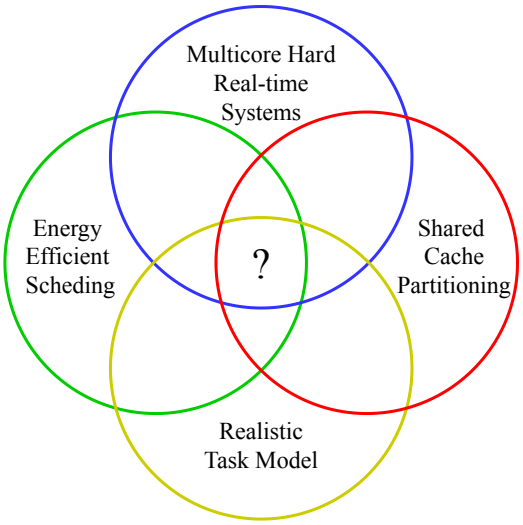


Figure 1-5: Complexity of the research problem

for more practical approaches to minimize the system-level energy consumption in multicore hard real-time systems. In this work we first propose an improved non-linear task model that accounts for the change in memory-latency cycles w.r.t to the number of CPs accessible by a task. This improved task model depicts a more accurate change in execution cycles when DVFS is applied while also catering for unpredictability posed by the shared cache by integrating with the cache partition model. We then apply this improved task model to a homogeneous multicore setting where we propose a novel cache contention model so that well-existing cache-oblivious energy-efficient scheduling algorithms can easily adopt the cache partition model into their problem setting. This allows existing algorithms to not only ensure predictability of the system but also to include the shared cache into their energy minimization problem. We then follow up upon this work by investigating the effects of task execution cycles and cache partitioning in a heterogeneous multicore setting where the multicore processor is composed of cores of different types. Scheduling real-time tasks on heterogeneous multicores is far more challenging since the heterogeneity of the cores adds another dimension to the scheduling problem. However, we propose a novel approach to system-level energy minimization on heterogeneous multicore real-time systems that outperforms state-of-the-art approaches. Finally, since existing cache partitioning algorithms are biased towards a static CP scheme for scheduling of periodic tasks, we propose a schedulability test for the dyanmic CP scheme as the dynamic scheme offers greater schedulability, flexibility and energy-efficiency. Figure 1-6 summarizes the contributions of this thesis.

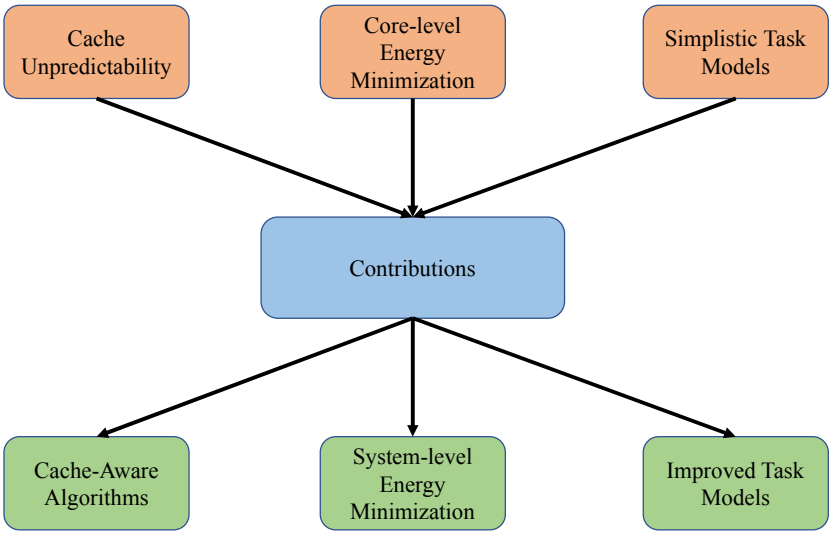


Figure 1-6: Thesis contributions

1.6 Thesis Outline

This thesis is organized in the following manner:

- We present a thorough literature review of existing energy-efficient scheduling techniques for both homogeneous and heterogeneous multicores along with cache-partitioning techniques adopted by the real-time systems community in Chapter 2.
- We propose an improved task model that captures the non-linearity of the memory latency cycles brought about by the shared cache and its impact on the energy consumption of the system in Chapter 3.
- For Homogeneous multicores, we propose a novel approach to model the dynamic CP inter-core interference for frame-based tasks and demonstrate its usability by existing energy-efficient scheduling algorithms in Chapter 4.
- For Heterogeneous multicores, we propose a holistic approach to minimize system-level energy consumption while including the proposed task-model and shared cache model into the energy minimization problem in Chapter 5.
- Finally, we make initial contributions to a dynamic CP schedulability analysis for fixed-priority periodic tasks for future energy-efficient scheduling in Chapter 6.
- We then conclude this thesis with some future directions in Chapter 7.

Chapter 2

Literature Review

This chapter will discuss previous works relevant to this thesis. It is important to note that most of the reviewed energy-efficient scheduling algorithms are specifically designed for multiprocessors. This is because research on energy-efficient real-time scheduling algorithms for single-core processors initially progressed into algorithms for multiprocessor and distributed architectures before entering the multicore domain. Since multiprocessors are usually void of shared caches, problems associated with the unpredictability of shared caches were not considered. Even as multicores became a reality for real-time systems, the majority of energy-efficient scheduling algorithms specific to multicores still failed to accommodate the shared cache unpredictability into their energy minimization problem. Therefore, in the reviewed scheduling algorithms below, we do not make a distinction between multiprocessors and multicores. To simplify terminology, energy-efficient scheduling algorithms for both multiprocessors and multicores are classified as "energy-efficient multicore scheduling algorithms". Techniques proposed in the literature to cater for the shared cache unpredictability are reviewed separately along with a discussion on the limited attempts made to combine such techniques with energy-efficient multicore scheduling algorithms.

2.1 Energy-Efficient Scheduling on Homogeneous Multicores

The energy minimization problem for multicores consists of simultaneous mapping and scheduling the tasks in a manner that minimizes the energy consumption. Thus, the energy minimization is highly dependent on the task allocation strategy as different task allocations will result in different values of energy consumption. The bottom line widely-adopted approach in the literature is to discretize the problem into first allocating the tasks across cores to promote maximum energy savings, and then performing energy-efficient techniques, e.g. DVFS, DPM, etc. effectively to minimize the energy consumption of the tasks.

In the multicore domain, DVFS can be extended into two flavors (i) local DVFS and (ii) global DVFS. These techniques depend on the processor hardware characteristics. In local

DVFS, each core has its own voltage regulator and can operate at a frequency different from the other cores. This enables maximum energy savings as each core can be tuned according to its assigned taskset characteristics. However, providing each core with an independent regulator adds a considerable expense and complexity to the hardware. In global DVFS, all the cores are controlled by a single voltage regulator making it a more practical approach. In order to avoid deadline misses, the global frequency of the chip is selected to match the frequency of the core with the highest execution workload. Recently, Voltage Frequency Island (VFI) techniques have also emerged as a hybrid that leverage from the flexibility of local DVFS and the simplicity of global DVFS. Cores are divided into clusters. Each cluster can operate at a separate frequency while the cores within a cluster are forced to run at the same frequency.

In this section, we discuss the various allocation and scheduling techniques reported in the literature that aim to minimize the energy consumption for tasks executing on homogeneous multicore architectures. Homogeneous multicores consist of cores of the same type. Therefore, a task will consume same energy at a reference frequency irrespective to its core assignment. Though there is a wealth of research articles in this area, the studies specified in this section are selected to give the readers a progressive understanding and general overview of this vast domain. Therefore, we summarize the existing work based on the general classification, proposed by Davis et al. [36], for multicore scheduling algorithm in real-time systems, i.e., Partitioned, Semi-Partitioned and Global scheduling techniques. The same classification is also adopted for its heterogeneous counterpart as shown in Figure 2-1.

2.1.1 Partitioned Scheduling

In partitioned scheduling, tasks are statically allocated to cores at compile time and there are no task migrations. After tasks are allocated, energy-efficient techniques, e.g., DVFS and DPM, are applied to the allocated taskset in order to minimize the energy consumption. One of the first theoretical studies on energy-efficient scheduling on multicores was performed by Chen et al. for independent frame-based tasksets [30]. They certified the problem to be NP-Hard and proved that the energy consumption can be minimized by minimizing the makespan and then setting the frequency of each task according to the utilization of the core it is allocated to. However, they only considered the dynamic component

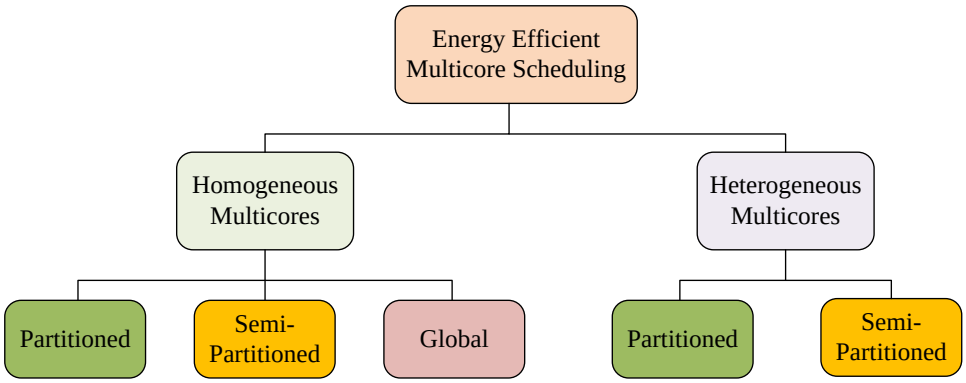


Figure 2-1: Classification of energy-efficient multicore scheduling algorithms for real-time systems.

of the power consumption model. Therefore, for all problems focusing on frame-based independent tasksets, minimizing the makespan of the schedule will be optimal in minimizing the dynamic energy consumption.

However, unlike the independent taskset case, a schedule that minimizes the makespan for dependent tasksets does not minimize the energy consumption [45]. On the contrary, a schedule pertaining greater parallelism with more slack allocated to the parallel portions of the schedule promotes greater energy savings [104]. Dependent tasksets are usually modeled as Directed Acyclic Graphs (DAGs) to be used as input for the energy minimization problem [16]. This problem is also NP-Hard. Due of the dependency relationship between the tasks, slack is accumulated between the task executions and energy is decreased by utilizing this slack to scale the tasks. Therefore, a schedule that results in an optimal slack allocation leads to energy minimization. Attempts have been made to optimally allocate the slack via convex optimization formulations [29,128] or heuristic algorithms [84]. Parallel task models have also been proposed in order to maximize the degree of parallelism [16–18, 49, 101]. In these works, the authors consider a model where each task itself is a parallel task that can utilize multiple computing units at the same time and where each task is represented as a DAG. In doing so, they have proposed energy-efficient real-time scheduling techniques of sporadic parallel tasks for both implicit and constrained deadlines. With the prevailing leakage current with transistor feature size reduction, some authors also include the critical frequency into their analysis [45], while others reduce the energy consumption by reducing the number of active cores [50] or by simultaneously considering DVFS and

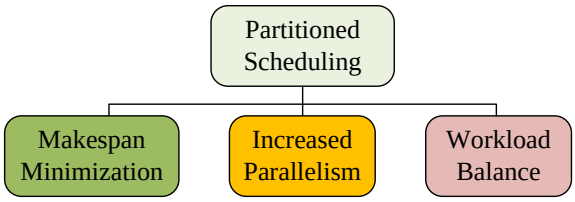


Figure 2-2: Classification of energy-efficient partitioned scheduling on homogeneous multi-core.

DPM techniques [29].

For periodic tasksets, it has been shown that balancing the workload, i.e., taskset utilization, across the cores will minimize the energy consumption. This problem is also NP-Hard in the strong sense when the number of cores are greater than two [5]. However, as with the case for single-core processors, initial studies only focused on the dynamic component of the processor power consumption. More recent approaches also consider leakage current into their scheduling algorithm, e.g., in [31] where the critical frequency is also taken into analysis, and in [127] where the authors strive to balance the tradeoff between attaining a workload balance and decreasing the number of active cores.

Therefore energy-efficient techniques adopted by the authors for partitioned scheduling can be categorized into three groups as shown in Figure 2-2. These categories are based on proofs derived for different types of tasksets, i.e., minimizing the makespan for independent frame-based tasks, increasing parallelism for dependent frame-based tasks and balancing the workload for periodic tasks leads to energy minimization. Most of the articles focus on local DVFS even though commercial processors are usually equipped with the more practical global DVFS setup [53]. Some papers also suggest task-specific voltage frequency scaling where tasks scheduled on the same core can execute at different frequencies w.r.t. each other. Others propose a global scheme where tasks assigned to the same core operate at the same frequency.

Some authors have also considered tasks with different power characteristics, i.e., tasks consume different amounts of power at the same core frequency. Schmitz et al. [102] proposed an algorithm to iteratively elongate pre-allocated tasks with the highest energy-gradients. Energy-gradient is defined as the difference in energy dissipation of a task before and after elongation. Luo et al. [84] followed up on this work by performing execution order optimizations via simulated annealing to further improve the energy consumption.

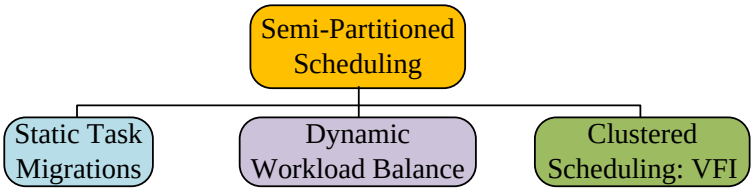


Figure 2-3: Classification of energy-efficient semi-partitioned scheduling on homogeneous multicores.

Similarly, Genetic Algorithms (GAs) have also been proposed [87].

2.1.2 Semi-Partitioned Scheduling

Due to the innate nature of tasks and partitioning heuristics, it is difficult to obtain a perfectly balanced workload across the cores. By permitting controlled migrations across cores, semi-partitioned scheduling techniques attempt to decrease the energy consumption by striving to obtain a better workload balance. Semi-partitioned energy-efficient scheduling algorithms can be further categorized in static task migrations, dynamic workload balancing and clustered scheduling as shown in Figure 2-3. In static task migrations, some works achieve a better workload balance by partitioning the taskset such that some tasks are allocated to only one core while the others are allocated to more than one core [30, 129]. Other works use parallel task models to break up the task into different components in order to reduce the execution time of the tasks [69, 95]. These techniques are classified as static task migrations since task-splitting is performed at compile time.

However, even though statically assigning a task to multiple cores produces a better workload balance, the performance demand on each core can change during application execution. This temporal imbalance can be solved by run-time migrations across the cores. Seo et al. [103] proposed a technique to balance the workload using run-time migrations by migrating the nearest deadline task from the highest utilization core to the lowest utilization core. Such dynamic workload balancing techniques are also effective in a dynamic workload environment where tasks enter and leave the system during run-time [89]. Again, tasks are migrated from the heaviest to the lightest utilization core in order to reduce the imbalance.

A subset of the semi-partitioned scheme can be enveloped into clustered scheduling. The cores on a processor are first grouped into clusters. The taskset is partitioned across the clusters and each cluster, then, schedules its allocated taskset independently. Tasks are

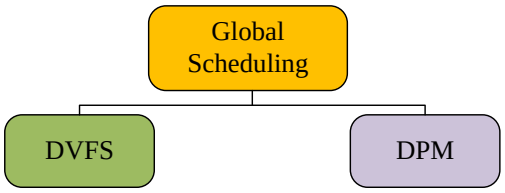


Figure 2-4: Classification of energy-efficient global scheduling on homogeneous multicores.

allowed to migrate across cores within the same cluster. Each cluster can also operate at an independent frequency, e.g., using a VFI scheme [53, 71, 118].

2.1.3 Global Scheduling

Global scheduling has inherent load balancing capabilities as it dynamically assigns tasks at run-time to available cores. However, due to the global nature of such scheduling algorithms, it is impossible to determine the task-to-core allocation or task execution sequence on a particular core at compile time. Without this information static DVFS schemes, that utilize the predefined available slack to scale the tasks, cannot be used. Hence, performing DVFS at run-time may cause future deadline misses. Several authors have attempted to perform DVFS techniques by constructing canonical schedules ahead of the practical schedule [15] or by utilizing the slack produced by early completion of the tasks [132].

Authors have also proposed simplistic DPM techniques that switch-off inactive cores to decrease the energy consumption [62]. Advanced DPM techniques, e.g., procrastination techniques that increase the idle periods between task executions, may also lead to un-schedulability as switching to sleep mode on one core can also affect the schedulability on the other cores [2]. Therefore, it is difficult to predict the outcome of the overall schedule even when a change is performed on a single-core. Others have attempted to increase the idle period durations for DPM by creating a static schedule and, then, using a global scheduling algorithm to schedule tasks within the time frames defined by the static schedule [77].

2.2 Energy-Efficient Scheduling on Heterogeneous Multicores

Homogeneous multicore processors consist of identical cores with identical executing frequencies. Such cores have the same architectural (core type, cache sizes, etc.) and micro-architectural (superscalar, out-of-order execution, etc.) features. Heterogeneous multicore

processors, on the other hand, consist of cores with varying features. This diverse computing potential is desirable because different tasks can have different core-type affinities in terms of performance and energy requirements. The execution time and energy consumption of a task running on a particular core-type depends on the set of instructions it must execute as various instruction types using different parts of the processor [3, 9]. A task may be executed with exceptionally good performance boost when executing on a particular core-type while other tasks may not take any advantage of the additional capabilities available on the same core-type. The heterogeneous multicore processors discussed in the literature are usually limited to a combination of two-type cores. This is because two-type core architectures already present most of the power and performance benefits of heterogeneity and are composed of a set of small, in-order, power-efficient cores with another set of big, out-of-order, high performance cores [64]. Furthermore, such processors are already available in the market.

Though heterogeneity displays significant benefits in power and performance, it brings along a significant complexity in task scheduling. Along with each task's characteristics, the scheduler must also take the different power, performance and architectural capabilities of the cores into consideration while allocating tasks to different cores [85].

Lately, the scheduling problem on heterogeneous architectures has received a lot of attention from the research community, especially for mainstream computers. Similar to the case of homogeneous multicores, heterogeneous multicore scheduling can be either static or dynamic. In the static scheme, tasks are partitioned across core-types at compile time. Since tasks may have different performance and resource usage characteristics over time, this approach has proved to be suboptimal [9]. The dynamic approach relies on analyzing a task's performance on each core-type after specific intervals and selecting the core-type that maximizes performance and/or energy-efficiency.

Heterogeneous multicores are also finding their way into the real-time systems domain. Techniques proposed for soft real-time systems rely on optimizing the Energy Delay Product (EDP) through regression modeling [99] or by solving complex optimization problems [98]. More recently, heterogeneous multicores have caught the attention of the hard real-time systems research community. Techniques proposed in the literature for energy-efficient scheduling are limited to partitioned and semi-partitioned techniques due to the schedulability complexity of global scheduling.

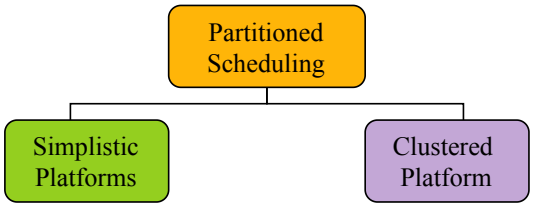


Figure 2-5: Classification of energy-efficient partitioned scheduling on heterogeneous multicores.

2.2.1 Partitioned Scheduling

Colin et al. [34] showed that neither balancing the workload across the cores (which maybe preferable for homogeneous multicores), nor assigning the maximum load to the energy-efficient core-type is optimal for heterogeneous platforms. Therefore, the optimal workload assignment to each core must be determined, which depends on the power characteristics of the core-types and the tasks. Since the problem is NP-Hard, solutions proposed in the literature are limited to Linear Program (LP) formulations and heuristic algorithms. For LP formulation, both task partitioning and energy minimization techniques are performed simultaneously. Heuristic solutions, however, first classify tasks based on their energy-favorable core-types and then allocate them accordingly while ensuring that deadline constraints are met. DVFS or DPM is then applied to the allocated taskset to minimize energy consumption. Furthermore, similar to the homogeneous multicore case, many initial studies assume different tasks to consume the same amount of power when executed on the same core-type, which simplifies the solution.

Energy-efficient partitioned scheduling on heterogeneous multicores can be classified based on the adopted platform, as shown in Figure 2-5. Many initial studies focused on simple hypothetical platforms composed of multiple core-types where the heterogeneity of each core-type is defined by its maximum operating frequency or power-consumption. As practical heterogeneous multicores became a reality, many proposed solutions began to target specific heterogeneous architectures available in the market, e.g., ARM's big.LITTLE clustered heterogeneous architectures where cores of the same type are grouped into a cluster as shown in Figure 2-6. In such solutions, the actual measured power consumption values of each core-type are used to efficiently allocate the tasks across the core-type clusters.

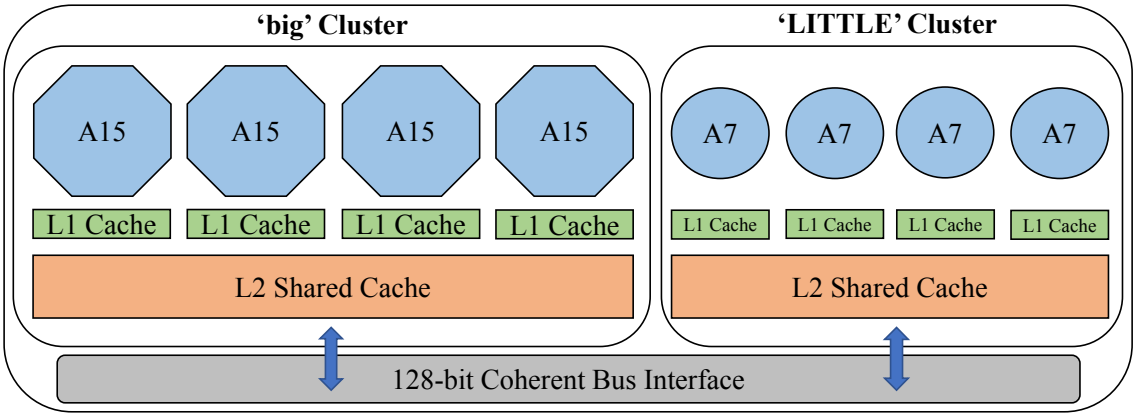


Figure 2-6: Exynos 5422 System-on-Chip (SoC) composed of A15 (big) and A7 (LITTLE) core-type clusters.

Li et al. [78] proposed LP formulations of three DVFS scenarios: (i) Static Global DVFS, (ii) Dynamic Global DVFS, and (iii) Dynamic Local DVFS, on a simplistic heterogeneous platform for frame-based tasksets. The heterogeneity of the system is defined by the execution efficiency of a task on each core-type. Similarly, Elewi et al. [43] proposed different DVFS solutions for periodic tasksets. The heterogeneity of the cores is defined by the maximum frequency of the cores. They, first, create an Integer Linear Program (ILP) for the four cases of DVFS (i) Without DVFS, (ii) Global DVFS, (iii) Local DVFS and (iv) VFI Islands. The ILPs are then complimented with heuristic algorithms to reduce the solution complexity. DPM based approaches have also been proposed for simplistic heterogeneous multicore platforms. Kuo et al. [74] proposed a heuristic that partitions a periodic taskset according to the power consumption characteristics of the tasks. The partitioned taskset is then scheduled according to EDF along with DPM techniques to switch-off the cores when inactive. Awan et al. [3] proposed a DPM approach with multiple sleep states for periodic tasksets. They divide the partitioning problem into two phases. The first phase consists of allocating tasks to cores with the aim to minimize the dynamic energy consumption. The second phase, then, attempts to reduce the static energy consumption by reallocating selected tasks from each core in order to promote deeper DPM sleep states.

Energy-efficient algorithms for clustered heterogeneous multicores, where cores of the same type are grouped into clusters, have also been proposed. Liu et al. [80] modeled real-time streaming applications as hard real-time tasks to efficiently map the workload across clusters while ensuring that the throughput and latency constraints are met. They first

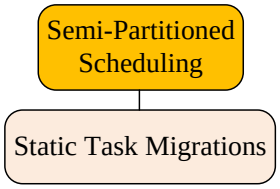


Figure 2-7: Classification of energy-efficient semi-partitioned scheduling on heterogeneous multicores.

classify tasks based on core-type affinities and then use FFD to map the tasks across clusters. More recently, energy-efficient techniques for tasks with different power characteristics have been proposed [94,105]. Both works focus on the ARM big.LITTLE clustered heterogeneous multicore platform and use actual core-type power consumption values to schedule a periodic taskset. Pagani et al. [94] first selected specific frequencies for the core-types in order to reduce the solution space. After sorting the tasks according to their utilization, tasks were allocated across their energy-favorable core-types in a Worst-Fit Decreasing (WFD) manner and DVFS was applied to minimize energy consumption. This was repeated for a number of initial core-type frequency configurations in order to find the energy favorable configuration. Suyyagh et al. [105] first determined frequencies at which it is energy-favorable to allocate tasks to the high-performance core-type since a task may consume less energy running on a high-performance core-type at a lower frequency compared to an energy-efficient core-type at a higher frequency. Initial core-type frequencies were also assigned with the aim to maximize the number of tasks that can fit on the energy-efficient core-type. Tasks were then allocated in a Best-Fit Decreasing (BFD) manner according to their utilization in order to minimize the idle-power consumption. The above mentioned algorithms do not, however, allow intra-cluster task migrations and are, therefore, classified under partitioned scheduling.

2.2.2 Semi-Partitioned Scheduling

The literature on solutions for semi-partitioned scheduling for heterogeneous multicores is rather limited. Furthermore, solutions are restricted to static task migrations.

The authors in [108] presented an LP formulations to determine the percentage of time interval and the frequency at which each task should execute on a core-type. After parti-

tioning, the Hetero-Wrap algorithm [33], was used to schedule the tasks in order to avoid parallel execution of the same task. The formulated LP was, then, extended to reduce the energy consumption via DVFS. Liu et al. [81], however, proposed a heuristic algorithm and adopted a task splitting approach [24] to efficiently utilize the capacity of the cores.

The presented literature on energy-efficient scheduling depicts extensive and diverse solutions for both homogeneous and heterogeneous multicores. However, such solutions have failed to accommodate shared-caches into their energy-model. In the following sections, we describe the steps taken by researchers to model the shared-cache for hard real-time systems. We, then, summarize the attempts made to accommodate such models into the energy-minimization problem.

2.3 Cache-Aware Scheduling on Multicores

WCET unpredictability due to shared caches has been largely tackled by two mechanisms, cache locking and cache partitioning.

Cache locking techniques permit tasks to load specific data into the cache which can then be locked to prevent the contents from being replaced at run-time by other concurrent tasks accessing the shared cache. Cache locking can also be used to improve performance by locking hot and frequently accessed data. However, careful WCET analysis must still be carried out to capture the effects of cache locking [47, 92].

Cache partitioning techniques involve dividing the shared cache into sections to be assigned to specific cores. This prevents concurrent tasks from sharing the same cache lines, thus, diminishing inter-core conflicts. Cache partitioning can be performed at the software-level, e.g., cache-coloring [116] where a mapping between cache entries and physical addresses prevents inter-core conflicts, or hardware-level, e.g., way-based partitioning [122] where the shared cache is divided into sections based on the cache associativity. Cache partitioning adds another dimension to the mapping and scheduling problem, i.e., how to efficiently distribute both tasks and CPs among the cores in order to ensure schedulability of the taskset. The WCET of a task is subject to the number of CPs assigned to it. Depending on the computational characteristics of a task, fewer CPs assigned to a task may result in an increased WCET as shown in the example taskset in Table 2.1. Therefore, such algorithms attempt to optimally perform CP-to-core and task-to-core mappings to increase

Table 2.1: CP-dependent WCET for 3 tasks and a total of 8 CPs

CP	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**Tasks**
WCET	10	9	8	8	8	7	7	7	τ_1
	20	20	20	18	16	10	8	8	τ_2
	11	10	9	8	8	5	4	4	τ_3

schedulability. The CPs can be used in two ways: (1) static scheme (core-based), where CPs assigned to a core remain constant throughout taskset execution, and (2) dynamic scheme (task-based), where CP assignments can be changed at run-time.

2.3.1 Static Cache-Partitioning

Given their correlative nature, the majority of cache-aware partitioned scheduling algorithms focus on a static CP scheme. This greatly simplifies the analysis since tasks allocated to the same core share the same CPs and are not blocked due to the cache-contention from tasks running on other cores. Researchers have proposed various methods to group specific tasks onto cores in order to efficiently utilize the limited CP resources. Berna et al. [11] proposed a two phase algorithm where the first phase involved assigning a *critical* task to each core and then partitioning the remaining tasks based on their period and WCET relationship with the critical tasks. The second phase then readjusted the task partitioning to ensure schedulability of the taskset. A similar approach was proposed by [26] where periodic tasks are grouped together based on their harmonic relationship while taking the CP-dependent variable WCET into consideration. Some works have also considered the impact that preemption has on the WCET, i.e., tasks sharing common CPs undergo Cache-Related Preemption Delay (CRPD) analysis in order to account for the delay due to cache-line invalidation by preempting tasks on the same core [23, 65]. In order to minimize this delay and improve schedulaility, Guo et al. [51] proposed a Mixed Integer Linear Program (MILP) along with approximation algorithms to ensure that strongly interfering tasks are mapped onto distinct cores.

Along with cache-partitioning, some authors have also considered other shared resources that can impact the WCET of tasks. Paolieri et al. showed that the number of tasks executing in parallel can also have an impact on the WCET due to memory controller contention and, therefore, proposed a partitioning algorithm to cater for the extra WCET

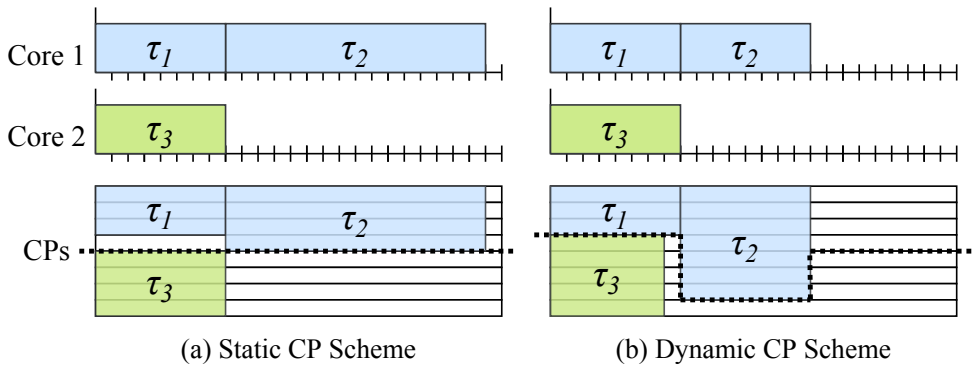


Figure 2-8: Static vs. dynamic CP scheme

variability caused by the memory controller. Similarly, Xu et al. [120] proposed a method to partition both CP and memory bandwidth resources in order to ensure predictability. Valsan et al. [110] showed that the Miss Status Holding Registers (MSHR) can also be a source of unpredictability and they proposed a hardware-software solution to cater for both CPs and MSHR allocation.

2.3.2 Dynamic Cache-Partitioning

The dynamic CP scheme has been exclusively used in global scheduling. This can theoretically increase the schedulability, however, it substantially increases the complexity of the scheduling problem since CPs are also considered a global resource and the scheduler needs to decide on the number of CPs to assign to each core at run-time to ensure schedulability. Furthermore, devising schedulability tests for this scenario is also a difficult problem. Attempts have been made to extend the classical non-CP schedulability tests for global scheduling algorithms into the CP-aware scenario for both preemptive [115, 122] and non-preemptive schemes [48, 119] where both cores and CPs are considered as global resources for a task, and the maximum interference that can be imposed by its HPTs on both the cores and CPs are determined via the problem-window approach [36]. However, these schedulability tests assume a predefined fixed CP allocation to a task and, therefore, a fixed WCET for each task.

For partitioned scheduling, Ward et al. [116] proposed a cache-management technique for preemptive and non-preemptive scheduling of periodic tasks which involves cache-coloring along with way-based cache-locking. However, the proposed schedulability analysis is simpli-

fied by reducing the problem to a uniprocessor scheduling problem resulting in a pessimistic schedulability test for each core. There have been other attempts to adopt a dynamic CP scheme into partitioned scheduling, however, most initiatives are limited to MILP formulations [27]. The dynamic CP scheme can offer greater flexibility and schedulability for partitioned scheduling since tasks are not restricted by the static CP assigned to a core. This can be seen in Figure 2-8 for three independent frame-based tasks partitioned across two cores, based on the CP-variable WCET from Table 2.1, and where the dotted line represents the division of CPs to each core. The resultant finish-time of the taskset for the dynamic CP scheme in Figure 2-8 (b) is less than that of the static CP scheme in Figure 2-8 (a). Furthermore, τ_1 does not benefit from the 4 CPs allocated to it since 3 CPs result in the same WCET. The dynamic CP scheme has also been proven to be beneficial in terms of energy-efficiency [28].

2.4 Cache-Aware Energy-Efficient Scheduling on Multicores

There have also been some attempts to cater for cache unpredictability while minimizing the energy consumption. Wang et al. [112] considered a reconfigurable cache model in their energy minimization problem for periodic tasks on a single-core system. They strived to minimize the energy consumed by both the core and cache subsystems where the energy consumption of a task is defined by its execution frequency and cache configuration. However, the work presented is for a single-core system while our primary focus is on multicore systems, which is far more challenging. For multicores, Fu et al. [44] attempted to minimize the core-energy consumption while considering the cache contention among soft real-time tasks. Tasks were statically partitioned across the cores and CPs were allocated on a core basis (i.e. using the static CP scheme). They modeled the WCET of a task to be dependent on the core frequency and the number of CPs assigned to it. However, uncertainties in the task-model restrict their work to be used for hard real-time systems.

Wang et al. [114] proposed an energy minimization technique which employed both dynamic reconfiguration of private L1 caches and partitioning of shared L2 caches for multicore processors. They adopted a profiling technique with dynamic programming to find L1 and L2 configurations to minimize energy consumption of the cache-subsystem. However, their profiling technique can be computationally expensive for larger tasksets and wider range

of cache configurations. They also adopted a static CP scheme that can result in resource wastage as tasks can require fewer CPs than those allocated to the core. It also results in reduced energy savings as shown in [28]. Chen et al. [28] proposed an MILP to allocate CPs to tasks and generate a time triggered schedule that minimizes the energy consumption, however, their approach can be computationally expensive for large number of tasks and system variables. Furthermore, both approaches presented by Wang et al. [114] and Chen et al. [28] attempt to minimize the energy consumption of the cache subsystem only and therefore, cannot exploit the existing techniques that use DVFS at core-level.

Authors in [126] proposed a method to minimize the complete system energy by performing DVFS on both core and cache subsystem. They proposed a task model where the power characteristics of each task is defined by the ratio of its compute and memory cycles. They assigned different switching activity factors for different modes of operation, i.e., the activity factor during compute cycle execution (active mode) is greater compared to the activity factor during memory stall cycles when the core is idle (idle mode). However, their work is also limited to single-core systems.

2.5 Discussion

There is a proliferation of energy-efficient scheduling algorithms for both homogeneous and heterogeneous multicores. However, there are some practical aspects that have been overlooked and need to be addressed.

Though some algorithms do consider tasks with different power characteristics, e.g., [87, 94, 102, 105], the task model adopted in such works are very simplistic and do not cater for the nonlinear change in execution time brought about by the memory latency cycles. Also, most energy-efficient algorithms only focus on the core-level energy and are oblivious to the unpredictable nature of shared-caches. Since predictability is a major concern for real-time systems, energy-efficient algorithms cannot ignore these caching effects and will otherwise diminish their applicability to real-world systems.

Furthermore, the effects of cache-partitioning on heterogeneous multicores have not yet been addressed. With the advent of heterogeneous multicores with in-built CP technology, e.g., ARM DynamIQ, the need for CP techniques on such architectures becomes increasingly more relevant. There have been attempts to introduce cache-aware energy minimization

techniques for soft-deadline tasks running on heterogeneous multicore systems [42], however, such techniques are still an open issue in the hard deadline-constrained domain. It is also observed that in homogeneous multicore energy-efficient scheduling, many authors have attempted to solve the energy minimization problem via theoretical analyses and proofs. Solutions in the heterogeneous domain, however, lack such theoretical analyses that can lead to optimal solutions for energy minimization.

Finally, the dynamic CP scheme for partitioned scheduling has largely been untackled despite its proficiency in schedulability, flexibility, and energy-efficiency. Most cache-aware partitioned scheduling algorithms focus on a static CP scheme. However, this technique can greatly under-utilize the CP resources. Also, with the increasing number of cores and the transition towards manycore systems, this technique can substantially decrease taskset schedulability. There have been attempts to adopt a dynamic CP scheme into partitioned scheduling where the CPs are dynamically allocated to the cores. However, such initiatives are limited to simpler frame-based tasks and ILP formulations [27].

In the following chapters, we provide solutions to these existing limitation by proposing an improved task model and cache-aware energy minimization techniques for multicore real-time systems.

Chapter 3

Improved Task Model for System-Level Energy Minimization

The real-time research community has already identified the need for new task execution models that can capture non-linearities in DVFS scaling [90]. Therefore, there have been some efforts to separate the memory latency from the computation clock cycles by defining task models with two sets of execution cycles, i.e., computation cycles and memory cycles [44,126]. Dividing the execution length parameter permits a more realistic execution model as the core frequency can only affect the computation cycles. The memory cycles, on the other hand, are affected by the operating frequency of the memory and bus architecture.

Based on the seminal work proposed by Yun et al. [126], we take a step further in improving the execution model by incorporating the change in execution time and energy consumption brought about by the number of CPs assigned to a task. We then use the proposed model for a Three-Dimensional (3D) energy minimization problem, i.e., minimizing the core-, cache-, and system-level energy consumption.

Specifically, in this chapter:

- We make improvements to the existing task model to incorporate the dependency of execution cycles on the cache subsystem.
- We propose techniques to minimize the energy consumption of the core subsystem for tasks with different power characteristics.
- We modify the existing cache energy model and propose a greedy algorithm to minimize energy consumption of the cache subsystem.
- We then propose a method to minimize the complete system-level energy consumption via a Genetic Algorithm (GA).

3.1 Core-Level Energy Optimization

In this section, we first propose optimal frequencies to minimize core-level dynamic energy consumption for tasks with different power characteristics based on a traditional task model. We then make improvements to the existing task model to incorporate the independent memory latency cycles and investigate its impact to the energy minimization problem.

3.1.1 Traditional Task Model

In a traditional task model, each task is defined by a set of homogeneous execution-cycles c_n. These execution-cycles, when executed at a particular frequency f_n, result in the execution time of the task.

$$e_n = \frac{c_n}{f_n}. \tag{3.1}$$

The energy consumed by a single-core during a task's execution depends upon the execution length of the task and the power it consumes while the execution on the core. This power is composed of a dynamic and static component. The dynamic power component represents the circuit-level activity and functional units utilized during task execution, which can be accommodated into the dynamic component of the power equation by including a term of activity factor $\kappa \in \{0, 1\}$ [131]. Thus, the core dynamic power consumption for a task τ_n can be defined as: The energy consumed by a single-core during a task's execution depends upon the task's execution length and the power it consumes while executing on the core. This power is composed of a dynamic and static component. The dynamic power component represents the circuit-level activity and functional units utilized during task execution, which can be accommodated into the dynamic component of the power equation by including a term of activity factor $\kappa \in \{0, 1\}$ [131]. Thus, the core dynamic power consumption for a task τ_n can be defined as:

$$P_n^{core\_dyn} = \kappa_n f_n^{\alpha} \tag{3.2}$$

where κ_n is the activity factor during τ_n's execution and α is a constant ($\alpha \geq 2$). The core

dynamic energy consumption due to τ_n's execution is then defined as:

$$E_n^{core\_dyn} = P_n^{core\_dyn} e_n = \kappa_n f_n^{\alpha-1} c_n \tag{3.3}$$

Thus, the total core dynamic energy consumption of all tasks assigned to a core can be expressed as:

$$E^{core\_dyn} = \sum_{n=1}^{N} E_n^{core\_dyn} \tag{3.4}$$

Assuming a common κ_n in Eq. (3.3) causes all tasks to consume the same amount of power, which simplifies the energy minimization problem. This simplistic assumption, however, opposes the practical nature of the power characteristics of each task which can vary depending upon on the circuit activity and functional units used during task execution [131]. A number of studies have focused on the energy minimization problem for tasks with different power characteristics [34, 84, 102, 131] where different values of κ_n were assigned to each task. However, to the best of our knowledge, closed-form solutions for energy minimization of tasks with different power characteristics is still an open problem. Therefore, we propose a method to find a closed-form solution for the above-mentioned problem.

For this purpose, we model the real-time application as a set of independent frame-based tasks represented as $\Gamma = \{\tau_1, \tau_2, .., \tau_N\}$. In line with previous research studies, we assume a common deadline D for all tasks [45, 63, 79]. The goal, in this case, is to find the optimal frequencies f_n that will minimize the dynamic energy consumption of the core while ensuring the cummulative execution length of tasks does not exceed the deadline D.

$$\begin{aligned} \min_{f_n} \quad & \sum_{n=1}^{N} \kappa_n f_n^{\alpha-1} c_n \\ \text{s.t.} \quad & \sum_{n=1}^{N} \frac{c_n}{f_n} \leq D. \end{aligned} \tag{3.5}$$

Since the tasks are independent, relative sequencing of tasks has no impact on the energy consumption.

Theorem 1. *For tasks with different power characteristics executed on a single-core system with a common deadline D, the dynamic core energy is minimized when the core frequency during the execution of task τ_n equals*

$$f_n = \frac{\sum_{j=1}^{N} c_j \sqrt[\alpha]{k_j}}{D \sqrt[\alpha]{\kappa_n}}$$

Proof. The terms in Eq. (3.3) are strictly convex and since the sum of convex functions is also convex, the function will have at most one global minimum. This allows us to use the Lagrangian multiplier system to find the optimal value for f_n [22]. Thus, the Lagrangian can be defined as:

$$Ł = \sum_{n=1}^{N} \kappa_n f_n^{\alpha-1} c_n - \lambda \left(\sum_{n=1}^{N} \frac{c_n}{f_n} - D \right)$$

Taking the derivative w.r.t. f_n and λ we get:

$$\frac{\delta Ł}{\delta f_n} = (\alpha - 1)\kappa_n c_n f_n^{(\alpha-2)} + \frac{\lambda c_n}{f_n^2} = 0$$

$$\frac{\delta Ł}{\delta \lambda} = - \left(\sum_{n=1}^{N} \frac{c_n}{f_n} - D \right) = 0$$

that is,

$$f_n = \frac{\sum_{n=1}^{N} c_n \sqrt[\alpha]{(\alpha - 1)\kappa_n}}{D \sqrt[\alpha]{(\alpha - 1)\kappa_n}}$$

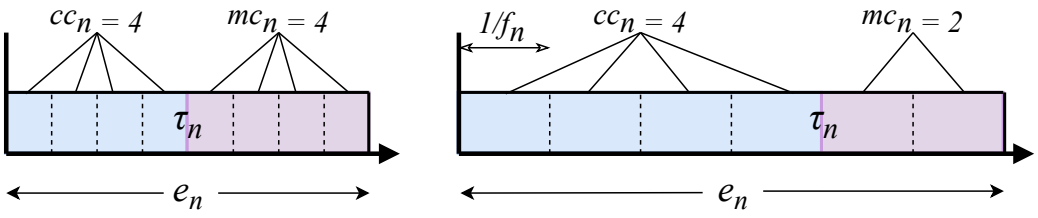
Since the numerator includes a summation of parameters of all the tasks in the taskset, the resultant equation can be simplified as:

$$f_n = \frac{\sum_{j=1}^{N} c_j \sqrt[\alpha]{k_j}}{D \sqrt[\alpha]{\kappa_n}}$$

□

3.1.2 Improved Task Model

In the previous section, we assumed that τ_n's execution-cycles scale linearly with the operation frequency of the core during τ_n's execution. However, this assumption does not take the nonlinear memory cycle latency into consideration. In order to incorporate this nonlinearity into the task model, we adopt an improved task model where each task is defined by the number of compute-cycles cc_n to be executed by the CPU along with number of



(a) Without DVFS (b) With DVFS

Figure 3-1: Improved task model reflects the change in compute cc_n and memory cycles mc_n when DVFS is applied on a task τ_n assigned to core. Number of cc_n remain constant while the number of mc_n decrease in order to compensate for the change in clock-cycle length of the core

memory cycles mc_n due to the memory subsystem [126].

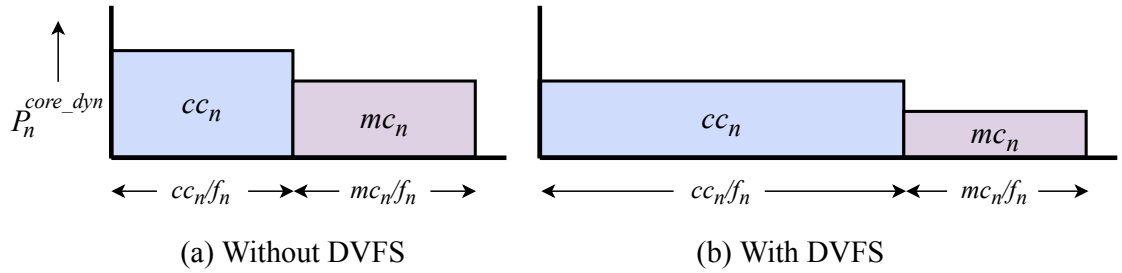
$$c_n = cc_n + mc_n \tag{3.6}$$

In such models, when DVFS is performed on a task, the number of compute-cycles cc_n remains constant while the clock-cycle length of the core increases. However, since DVFS – applied to the core subsystem – does not affect the frequency of the memory subsystem, the execution time contributed by the memory-latency cycles remains constant. This is because the memory latency cycles depend on the memory hierarcy and frequency of the memory subsystem. Therefore, the number of memory-cycles mc_n will decrease in order to compensate for the increase in clock-cycle length as shown in Figure 3-1. Thus, e_n can be defined as:

$$e_n = \frac{cc_n}{f_n} + \frac{mc_n}{f_{mem}} \tag{3.7}$$

where f_{mem} is the constant frequency of the memory subsystem.

Yun et al. [126] further elaborated the model by defining the power characteristics of each task by the ratio of its compute and memory cycles. They do this by assigning different activity factors to different modes of core operation where the activity factor during memory cycles (sleep mode) is smaller compared to the activity factor during execution of compute cycles (active mode). Thus, tasks with a larger compute-to-memory cycle ratio are assigned higher activity factors. This is a reasonable assumption since clock gating technology can be used to make the core consume less power during memory access/stall cycles by cutting off the clock supply to various components of the core [93]. The authors verified their proposed



(a) Without DVFS (b) With DVFS

Figure 3-2: Change in core-level dynamic power consumption of cc_n and mc_n when DVFS is applied

model by measuring the activity factors of numerous applications on actual hardware to achieve a maximum error of less than 2%. Thus, based on their proposed model, the dynamic energy consumed by the core during task τ_n's execution can be defined as:

$$E_n^{core\_dyn} = \kappa_a f_n^{\alpha-1} cc_n + \kappa_s f_n^{\alpha} \frac{mc_n}{f_{mem}} \tag{3.8}$$

where κ_a and κ_s are core's activity factors during compute- and memory-cycles with $\kappa_a > \kappa_s$. The change in $E_n^{core\_dyn}$ when f_n is reduced is depicted in Figure 3-2. Since $\kappa_a > \kappa_s$, the power consumed by cc_n is greater than that of mc_n. When f_n is reduced, the execution length contributed by cc_n is increased while that of mc_n remains constant. However, the dynamic power consumed by mc_n is also reduced since f_n contributes to the energy consumed by the mc_n component as expressed in Eq. (3.8).

The heterogeneous nature of the execution-cycles, however, increases the intractability of finding the optimal clock frequencies to minimize the energy consumption of the core. This is explained in the following corollary.

Corollary 1. *Considering the improved task model, the activity factor for each task can be defined as*

$$\kappa_n = \frac{\kappa_a cc_n + \kappa_s mc_n \delta_n}{cc_n + mc_n \delta_n} \tag{3.9}$$

where $f_n = \delta_n f_{mem}$

Proof. Given that f_{mem} is a constant, f_n can be related to f_{mem} by a task specific factor δ_n, causing the optimal frequencies of the tasks to be different from each other.

$$\kappa_n f_n^{\alpha} \left(\frac{cc_n}{f_n} + \frac{mc_n}{f_{mem}}\right) = \kappa_a f_n^{\alpha-1} cc_n + \kappa_s f_n^{\alpha} \frac{mc_n \delta_n}{f_{mem}}$$

Since $f_n = \delta_n f_{mem}$,

$$\kappa_n = \frac{\kappa_a cc_n + \kappa_s mc_n \delta_n}{cc_n + mc_n \delta_n}$$

The activity factor of each task depends on the ratio of f_n to f_{mem}. A closed-form solution is difficult to achieve in this case since κ_n is now a function of the optimization variable f_n. □

As a consequence to Corollary 1, we propose a convex optimization formulation to find the optimal frequencies f_n. Moreover, we include the static power consumption of the core into the energy equation to get the total energy consumption of the core subsystem. The static power and energy components are independent of the activity factor and can be defined as,

$P^{core\_sta} = k_{s1} f + k_{s2}$

$E^{core\_sta} = (k_{s1} f + k_{s2}) e_n$

where k_{s1}, k_{s2} are non-negative constants. Thus, core-level energy consumption of τ_n is defined as:

$$E_n^{core} = \kappa_a f_n^{\alpha-1} cc_n + \kappa_s f_n^{\alpha} \frac{mc_n}{f_{mem}} + (k_{s1} f + k_{s2}) e_n \tag{3.10}$$

3.1.3 Optimization Problem 1: Core-Level

The goal in our optimization problem is to find the optimal frequencies that will minimize the energy consumption of the core while ensuring that the total execution length does not exceed D.

$$\begin{aligned} \min_{f_n} \quad & \sum_{n=1}^{N} \kappa_a f_n^{\alpha-1} cc_n + \kappa_s f_n^{\alpha} \frac{mc_n}{f_{mem}} + (k_{s1} f + k_{s2}) e_n \\ \text{s.t.} \quad & \sum_{n=1}^{N} \frac{cc_n}{f_n} + \frac{mc_n}{f_{mem}} \leq D; \quad f_{min} \leq f_n \leq f_{max} \end{aligned} \tag{3.11}$$

Optimization solvers [41, 83] can be used to determine the optimal frequencies for each task.

3.2 Cache-Level Energy Optimization

Caches occupy a large area of the die and contribute significantly to the energy consumption of the system. However, energy minimization of the cache subsystem has received little

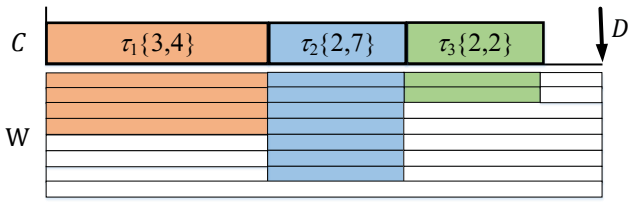


Figure 3-3: Cache partitioned task model

attention by the real-time community. The majority of works on energy-minimization for real-time systems have focused only on the core subsystem. In this section, we investigate techniques to minimize the energy consumed by the cache subsystem during a task's execution. We adopt the cache partitioning model and incorporate the CP attribute into the improved task model proposed in the previous section.

Due to the interactivity between the core and memory subsystems, tasks running on the core will require access off-chip cache if the requested data is not found in the on-chip caches. We adopt a way-based cache partitioning scheme proposed in [28, 122] where the of-chip cache is divided into A number of CPs represented as $W = \{w_1, w_2, .., w_A\}$. Furthermore, the number of memory cycles mc_n can vary based on the number of CPs a_n allocated to the task τ_n where $1 \leq a_n \leq A$ [26]. Thus, mc_n of each τ_n can be defined by a table with A columns indexed by a_n, i.e., $mc_n^{a_n} = \{mc_n^1, mc_n^2..mc_n^A\}$. A smaller a_n will increase mc_n due to an increase in cache miss-rate. Such a model is depicted in Figure 3-3 where W represents the CPs and C represents the executing core. Cache ways that have not been assigned to τ_n can be switched-off to reduce energy consumption via the selective-way energy saving approach [91].

Similar to the core-level energy consumption, the cache-level energy consumption is composed of both static and dynamic components [112]. Since we assume the cache to be switched-off during periods of inactivity, the total energy consumption of the cache subsystem can be defined as the sum of cache energy consumed by each individual task τ_n:

$$\begin{aligned} E^{cache} &= \sum_{n=1}^{N} E_n^{cache} \\ E_n^{cache} &= E_n^{cache\_dyn} + E_n^{cache\_sta} \\ E_n^{cache\_dyn} &= N_n^{ac} E^{ac} + N_n^{ms} E^{ms} \\ E_n^{cache\_sta} &= P^{cache\_sta} \frac{a_n}{A} e_n \end{aligned} \tag{3.12}$$

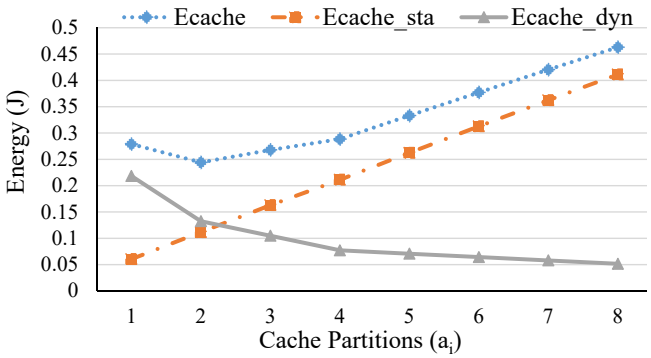


Figure 3-4: Cache energy consumption vs. cache ways assignment

The dynamic energy depends on the number of cache hits and misses experienced by a task. The number of cache accesses N_n^{ac} and cache misses N_n^{ms} can be found by static profiling techniques and depend on mc_n, a_n and the instruction characteristics of the task. Values for cache access energy E^{ac}, cache miss energy E^{ms} and cache static power P^{cache_sta} can be derived via CACTI [109]. The cache energy consumption can vary based on the value of a_n assigned to τ_n. Decreasing a_n will decrease the static energy consumption E^{cache_sta} since fewer CPs will be on, but at the same time may increase the cache miss-rate resulting in an increased cache dynamic energy consumption E^{cache_dyn} as shown in Figure 3-4.

We propose a greedy algorithm that finds the values of a_n for each task τ_n in order to reduces the energy consumption of the cache. Since our aim is to minimize the cache-level energy consumption, we keep the core-level power parameters constant. Thus, the optimization problem is defined as,

Optimization Problem 2: Cache-Level

$$\begin{aligned} \min_{f_n} \quad & \sum_{n=1}^{N} N_n^{ac} E^{ac} + N_n^{ms} E^{ms} + P^{cache\_sta} \frac{a_n}{A} e_n \\ \text{s.t.} \quad & \sum_{n=1}^{N} \frac{cc_n}{f_n} + \frac{mc_n^{a_n}}{f_{mem}} \leq D; \quad 1 \leq a_n \leq A \end{aligned} \tag{3.13}$$

Algorithm 1 attempts to minimize cache energy consumption by iteratively selecting tasks that have a minimum impact on the total execution length. Tasks are first assigned maximum CPs (line: 2) and are then selected based on their impact on the total execution length (lines: 4-5). Doing so will permit greater number of tasks to reduce their CPs.

ALGORITHM 1: Cache Energy Optimization

1: **Output:** a_n;
2: $a_n = A \quad \forall i$;
3: **while** Tasks are extendable **do**
4: $change = mc_n[a_n] - mc_n[a_n - 1]$;
5: Select task with minimum *change*;
6: $a_n = a_n - 1$;
7: Update e_n and E_n^{cache} of τ_n accordingly;
8: **if** $(\sum_{n=1}^{N} e_n > D || E_n^{cache}(a_n + 1) < E_n^{cache}(a_n))$ **then**
9: $a_n = a_n + 1$;
10: Update e_n and E_n^{cache} of τ_n;
11: Mark τ_n as non extendable;
12: **end if**
13: **end while**

This continues until the deadline is met. Since we are considering both dynamic and static energy, the energy curve for each task is strictly convex. We must, therefore, ensure that reduction in CPs does not increase the total energy consumption due to increase in dynamic energy from increased miss-rate (lines: 8-12).

3.3 System-Level Energy Optimization

In Section 3.1 and 3.2, we proposed solutions to minimize the energy for the core and cache subsystem. In this section, we propose a method to minimize complete system-level energy. Hence, the goal is to find optimal values for f_n and a_n that will minimize the overall energy consumption which is defined as:

Optimization Problem 3: System-Level

$$\begin{aligned} \min_{f_n} \sum_{n=1}^{N} [&[\kappa_a f_n^{\alpha} \frac{c_n}{f_n} + \kappa_s f_n^{\alpha} \frac{mc_n^{a_n}}{f_{mem}} + (k_{s1} f + k_{s2}) e_n] \\ &+ [N_n^{ac} E^{ac} + N_n^{ms} E^{ms} + P^{cache\_sta} \frac{a_n}{A} e_n]] \\ \text{s.t.} \sum_{n=1}^{N} &\frac{c_n}{f_n} + \frac{mc_n^{a_n}}{f_{mem}} \leq D;\ f_{min} \leq f_n \leq f_{max};\ 1 \leq a_n \leq A \end{aligned} \tag{3.14}$$

Optimizing both subsystems together increases the complexity of the energy minimization problem. We, thus, propose a Genetic Algorithm (Algorithm 2) to utilize the numerous degrees of freedom in the optimization variables. GAs have been widely used in numerous domains and are considered to be one of the most powerful techniques in solving optimiza-

Table 3.1: Genetic algorithm chromosome

τ_n	1	2	3	4	5	6	7	8
f_n	150	180	60	50	200	110	75	63
a_n	4	7	2	8	5	3	4	2

tion problems [87].

Within the context of our proposed optimization problem, the chromosome is represented by the frequencies f_n and CPs a_n of each task in the taskset. Thus, the chromosome can be viewed as a Two-Dimensional (2D) array with n columns and 3 rows as shown in Table 3.1.

ALGORITHM 2: System Energy Minimization

```
1: Output: a_n^*, f_n^*
2: Input: a_n, f_n
3: Generate initial Population
4: Assign fitness of the initial population
5: while termination condition not met do
6:     Select parents
7:     Perform crossover
8:         Randomly select genes from each parent
9:         Swap selected genes to create children
10:    For each child check feasibility
11:        Discard child if not feasible
12:    Perform Mutation
13:        For each child
14:            Increment/decrement a_n or f_n to decrease E_n
15: end while
16: Evaluate fitness of children
17: Replace population with new generation
```

Algorithm 2 gives a pseudo-code of the proposed approach. The initial population is generated by randomly selecting values for f_n and a_n (line: 2-3). Parents are selected based on a roulette wheel approach (line: 5). For the crossover operation, genes (tasks) of the parent chromosome are randomly selected and swapped to create child chromosomes. Task execution time e_n and energy E_n are updated accordingly (lines: 6-8). Feasibility condition is maintained by ensuring the total execution time does not exceed the deadline, D (lines: 9-10). Feasible child chromosomes then undergo a mutation operation where either f_n or a_n is changed in order to decrease the total energy consumption (lines: 11-13). The crossover and mutation operations are performed with probabilities of 0.8 and 0.6 respectively. The

energy consumption curves for core and cache are both non-monotonic based on their optimization variables f_n and a_n, respectively. Consequently, both increment and decrement of optimization variables must be evaluated to ensure convergence towards a minima. After ensuring the feasibility of the mutated chromosomes, the current population is replaced with the new generation (lines: 15-16). This continues until the energy consumption fails to decrease any further, resulting in energy favorable task parameters a_n^* and f_n^*.

3.4 Simulation Results and Discussion

In this section, we present simulation results for the proposed optimization problems and perform experiments to show how the cache can play a significant role in influencing the energy consumption of the system.

3.4.1 Experimental Setup

We modeled our taskset based on real-world application benchmarks, SPEC-CPU2000 [55]. Instruction count, miss-rates and load/store ratios of each benchmark for various cache sizes have previously been collected by authors in [25]. This available data was used to create a synthetic benchmark. The miss-rate and load-store ratio of a randomly selected SPEC-CPU2000 benchmark were used to model the cc_n and mc_n of a task τ_n for a specified cache size. The instruction count of the selected benchmark was randomly generated to introduce variability in the execution length of the task τ_n. We have used cache sizes of 8 kB, 16 kB, 32 kB, 64 kB and 128 kB with 8 ways as shown in Table 3.2. We use the technique proposed in [63] to define the deadline of the taskset as: $D = (1 + d_f) \times f_t$, where d_f is the deadline factor which can be set to 0.05, 0.1, etc., and f_t is the finish time of the last task in the schedule.

Parameters for the power model are taken from [126], i.e., $\kappa_a = 0.51, \kappa_s = 0.22$ and $f_n = [20, 200]$ MHz while f_{mem} is fixed at 200 MHz. The cache energy consumption parameters are retrieved from CACTI for 68 nm technology [109].

3.4.2 Energy Savings for Different Optimization Techniques

Figure 3-5 shows the energy saving percentages for the three optimization techniques proposed in the framework for $n = 10$ tasks with $d_f = 0.15$ and a cache size of 16 kB. OPT-1,

Table 3.2: Memory cycles (mc_n) vs. CP (a_n) for SPEC-CPU2000 for task length of 100 cycles

a_n	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**Benchmarks**
mc_n	38	19	12	5	5	4	3	2	164.gzip
	66	49	49	49	49	49	49	49	173.aplu
	74	29	25	20	19	18	16	15	178.galgel
	66	18	12	5	5	4	3	2	189.lucas

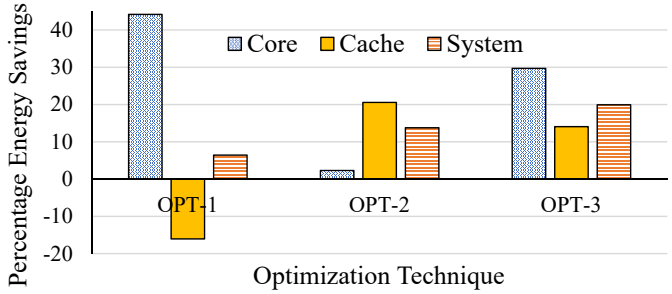


Figure 3-5: Energy savings for different proposed optimization techniques

OPT-2 and OPT-3 refers to the optimization algorithms proposed for core-, cache- and system-level energy minimization, respectively. For each optimization problem, the impact on energy saving of the core subsystem, cache subsystem and complete system (core+cache) are shown.

For the core-level energy optimization problem (OPT-1), the MATLAB $fmincon$ function was used to obtain to optimal frequencies f_n in Eq. (3.11). As expected, results show considerable energy savings for the core subsystem with a negative impact on the cache subsystem. The decrease in core frequency increases the tasks' execution length and thus elongates the length of time for which the assigned CPs are active. Since the CPs for each task are fixed, the cache energy is increased due to an increase the static energy component.

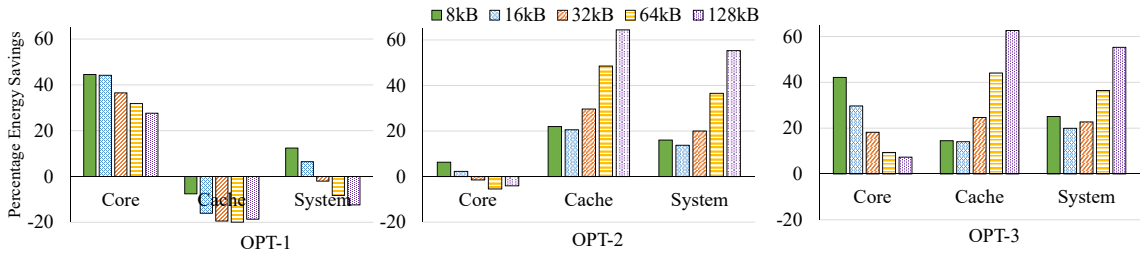


Figure 3-6: Energy savings for the proposed energy optimization techniques against different cache sizes

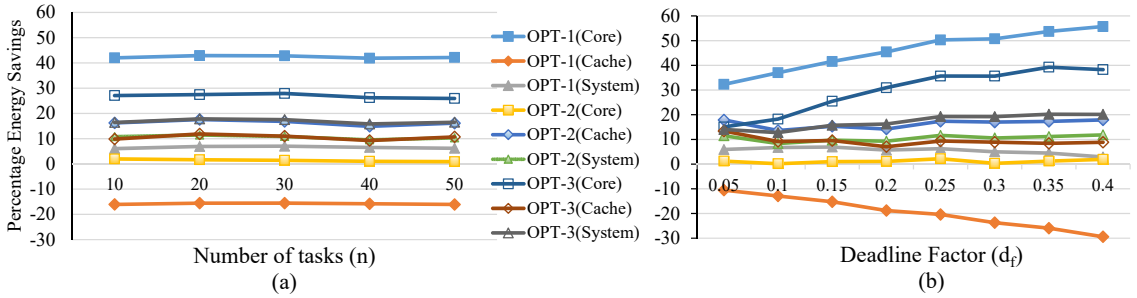


Figure 3-7: Energy savings for (a) different number of tasks, n and (b) different values of deadline factor d_f

Initializing the taskset with fewer CPs can reduce the cache static energy consumption but may also increase the cache dynamic energy consumption along with reducing the slack available for core DVFS.

For the cache-level energy optimization problem (OPT-2), Algorithm 1 attempts to reduce the value of a_n for each task which reduces the static cache energy consumption resulting considerable energy saving for the cache subsystem. The slight increase in energy savings for the core is explained below in Section 3.4.3.

For system-level energy optimization via GA (OPT-3), the energy savings for each individual subsystem (either cache or core) are less than those achieved through the subsystem-specific optimization techniques. However, the system-level savings are better because the GA simultaneously attempts to optimize both subsystems.

3.4.3 Energy Savings vs. Cache Size

Figure 3-6 shows how energy savings change as the cache size is varied from 8kB to 128kB. For OPT-1, core energy savings decrease as the cache size is increased. This is because mc_n dominates the execution cycles for smaller cache sizes resulting in greater slack per cc_n available for DVFS. Cache energy savings for OPT-1 initially decrease as the cache size is increased due to an increase in static cache energy consumption. However, for very large cache sizes, e.g., 128kB, energy savings tend to increase again because the reduction in dynamic energy consumption, due to a reduced miss-rate, dominates the cache energy profile.

For OPT-2, core energy savings decrease as the cache size increases because Algorithm 1 starts with a minimum core energy consumption since original values of a_n are initially replaced with full CPs A for all tasks. After the algorithm completes, some tasks may still

have a_n values greater than the original a_n assignment, thus, resulting in increased energy savings. However, as the cache-size increases, the change in miss-rate at higher values of a_n are less significant, permitting greater reductions in a_n, thus causing a_n to become smaller than the original assignment.

This reason also holds true for the cache subsystem where the energy savings increase significantly with an increase in cache size due to the reduced static energy consumption component. Similar trends are also observed for OPT-3.

3.4.4 Energy Savings vs. Number of Tasks

Figure 3-7 (a) shows the energy savings as n is increased from 10 to 50 for $d_f = 0.15$ and for a cache-size of 16kB. Energy savings remain the same across number of tasks and optimization techniques. This is understandable since the amount of slack available per task remains the same for the same value of d_f.

3.4.5 Energy Savings vs. Deadline Factor

Energy savings as d_f is varied from 0.05 to 0.4 for 16kB cache-size is shown in Figure 3-7 (b). Core energy savings for OPT-1 and OPT-3 increase with d_f because more slack is available for DVFS as d_f is increased. Increased slack utilization consequently has a negative impact on OPT-1 cache energy consumption. Energy savings for OPT-2 remain constant because a small value of d_f is sufficient for Algorithm 1 to find values for a_n that minimize the energy consumption.

3.5 Discussion

Cache energy minimization is usually neglected in the real-time systems domain even though it significantly contributes to the overall system energy consumption. For this reason, we improved existing execution task models by incorporating the change in execution time and energy consumption brought about by the number of CPs assigned to a task. We then used the proposed model for a 3D energy optimization problem, i.e., minimizing the core-, cache-, and system-level energy consumption, in order to demonstrate the effects that caches have on the energy consumption of various components of the system. Results demonstrated that the cache energy optimization plays a major role in overall energy minimization.

Chapter 4

Cache-Aware Energy-Efficient Scheduling on Homogeneous Multicores

Despite the overwhelming research on energy-efficient scheduling algorithms for multicore real-time systems, existing algorithms are oblivious to the unpredictable nature of shared caches or cache partitioning techniques, as discussed in Chapter 2. Since predictability is a major concern for real-time systems, such algorithms cannot ignore these caching effect and will otherwise diminish their applicability to practical scenarios.

Therefore, in this chapter, we extend the initial investigations on an improved task and system model for energy minimization, performed in the previous chapter, for the homogeneous multicore setup. In particular, the problem of scheduling cache-aware independent frame-based tasks to minimize system-level energy consumption is considered. In this work, we make use of the dynamic CP (task-based) scheme to take advantage of its increased flexibility, schedulability and energy-efficiency against the static CP (core-based) scheme. Since the simple task-to-core allocation problem in itself is NP-Hard, the overall objective is broken down into sub-problems to develop efficient and tractable solutions. The problem is divided into three non-trivial sub-problems: (i) cache-aware task-to-core mapping to minimize schedule length, (ii) modeling the inter-core dynamic CP interference, and (iii) utilizing the model to minimize the system-level energy consumption. We make assumptions that aim to build upon recently proposed and improved frameworks. Based on these assumptions, we then present proofs to further corroborate the cache-aware scheduling techniques proposed in this chapter. Specifically, this chapter makes the following contributions:

- We start with formulating an algorithm for makespan minimization of cache-aware independent frame-based tasksets.
- We then show how directly applying existing cache-oblivious energy minimization schemes on task-models accompanied with CPs can result in cache violations and then explore ways to decrease the energy consumption of tasks without any inter-core

dynamic CP interference model.

- We then continue to propose a novel approach to model the inter-core dynamic CP interference as a dependency graph, called *Cache Dependency Graph* (CDG).
- Finally, we present experimental results to demonstrate how existing algorithms can take advantage of our proposed approach to minimize the core-, cache-, and system-level energy consumption of tasksets modeled with CP requirements.

It is important to note that the proposed approach focuses on facilitating existing energy-efficient scheduling algorithms to accommodate the shared cache unpredictability and energy-consumption in their energy minimization problem. Therefore, to exhibit the effectiveness of energy minimization using our proposed CDG, we present the results using a well-established core-level energy-efficient scheduling algorithm [102] to reduce system-level energy consumption. However, since such existing algorithms are designed to minimize only the core-level energy consumption, utilizing the proposed CDG model does not guarantee system-level energy minimization. We, therefore, propose a CP-gradient approach to reduce only the cache-level energy consumption. We then combine both core- and cache-level techniques to propose an algorithm designed for system-level energy minimization.

4.1 System and Task Model

In this section, we adapt the improved task model, proposed in the previous chapter, to a homogeneous multicore setting. We also employ another commonly used power-model to indicate the universal applicability of the improved task model. With the presented models, we then show how cache-oblivious scheduling can disrupt the correctness of the schedule and argue upon the need for new CP-aware scheduling techniques for system-level energy minimization. In this section, we adapt the improved task model, proposed in the previous chapter, to a homogeneous multicore setting. We also employ another commonly used power-model to indicate the universal applicability of the improved task model. With the presented models, we then show how cache-oblivious scheduling can disrupt the correctness of the schedule and argue upon the need for new CP-aware scheduling techniques for system-level energy minimization.

4.1.1 Task Model

We consider a multicore platform with M identical cores represented as $O = \{o_1, o_2, .., o_M\}$ and adopt the way based cache partitioning scheme used in the previous chapter, where the shared cache is divided into A number of partitions represented as $W = \{w_1, w_2, .., w_A\}$. We consider a set of N independent frame-based tasks represented as $\Gamma = \{\tau_1, \tau_2, .., \tau_N\}$. Each task $\tau_n < cc_n, mc_n, a_n, f_n, e_n, d_n, r_n, R_n >$ can be defined by computation cycles cc_n, memory cycles mc_n, CPs a_n, core frequency f_n, execution time e_n, deadline d_n, start time r_n and finish time R_n.

Each task is composed of two sets of cycles; (i) constant compute cycles cc_n that are executed by the CPU , and (ii) variable memory cycles $mc_n^{a_n} = \{mc_n^1, mc_n^2..mc_n^A\}$ due to shared cache latency. The number of compute cycles remain constant while the memory cycles can change according to the CPs assigned to τ_n. Furthermore, since DVFS is only applied to the cores, the frequency of the core f_n only affects the cycle-length of cc_n, thus, increasing the execution time contributed by the compute cycles, while the execution time due to memory cycles remains constant. The execution time e_n can, therefore, be defined as:

$$e_n = \frac{cc_n}{f_n} + \frac{mc_n^{a_n}}{f_{mem}} \tag{4.1}$$

where f_n is the core-frequency when τ_n is executing and f_{mem} is the constant memory frequency. The e_n can change based on the variables f_n and a_n, i.e., a smaller a_n assignment may increase the cache miss-rate, increasing the number of memory cycles due to off-chip memory access. Similarly, decreasing f_n will increase the clock cycle length of the computation cycles, increasing e_n. A dynamic CP scheme is adopted where the CPs are considered as a global resource and scheduled tasks must acquire the required number of CPs before they can execute. Depending upon the availability, ready tasks can use any of the unengaged CPs. Similar to other works, e.g., [122], we assume that the overhead due to intrinsic cache misses is already included in the WCET of the tasks. In line with previous studies, we assume a common deadline of $d_n = D$ for all the tasks [45, 63, 79].

4.1.2 Power Model

We adopt another classical power model consisting of dynamic and static power components which is defined by the operating voltage and frequency of the core. We assume the core to

be switched off when it is idle, i.e, when no task is executing. Switching-off the core has an associated time and energy cost. Solutions have been proposed to account for this cost by defining break-even parameters to justify a transition in terms of energy minimization [6]. However, to simplify the analysis, we assume that these costs are already factored into the WCET and energy of the task. Thus, the total power can be expressed as the sum of power consumed by individual tasks.

$$P_n^{core} = P_n^{core\_dyn} + P_n^{core\_sta} = \kappa_n V_n^2 f_n + V_n I_n \tag{4.2}$$

where κ_n is the capacitive switching activity factor, V_n is the supply voltage, f_n is the core clock frequency, and I_n is the leakage current during τ_n's execution. I_n can be calculated with the following [126]:

$$I_n = \hat{k}_1 e^{V_n \hat{k}_2} \tag{4.3}$$

where $\hat{k}_1$ and $\hat{k}_2$ are constants. The clock cycle time t_n is related to the core-frequency as follows:

$$t_n = \frac{1}{f_n} = \frac{\hat{k}_3 [V_n - V_t]^2}{V_n} \tag{4.4}$$

where V_t is the voltage threshold. With the improved task model consisting of compute- cc_n and memory- mc_n cycles, the energy consumption of the core during τ_n's execution can be defined as:

$$E_n^{core} = \kappa_a V_n^2 cc_n + \kappa_s V_n^2 f_n \frac{mc_n^{a_n}}{f_{mem}} + V_n I_n e_n \tag{4.5}$$

where κ_a and κ_s are the core switching activity factors for active mode and sleep mode, respectively.

The cache energy model presented in Chapter 3 is used to define the total energy consumption of the system due to τ_n's execution as:

$$E_n = E_n^{core} + E_n^{cache} \tag{4.6}$$

4.1.3 Problem Formulation

Existing multicore energy-efficient algorithms are oblivious to the unpredictable nature of shared caches or recent cache partitioning models that isolate the shared cache unpredictability from the WCET of a task. Directly adapting existing algorithms to a dynamic

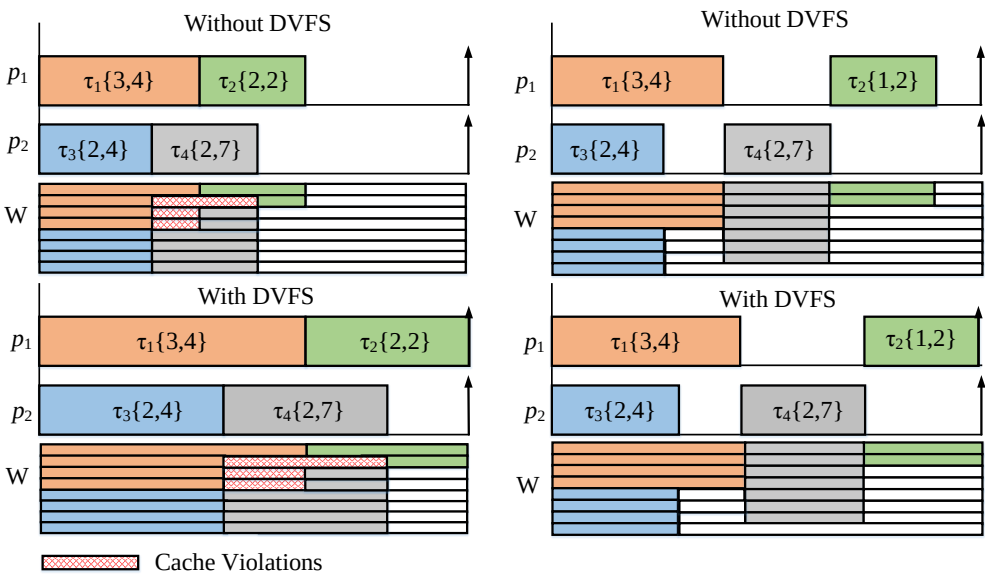
CP scenario can result in cache-violations, i.e., multiple tasks using the same CPs at the same time instant. These violations may increase the cache miss-rate of the tasks due to cache-line evictions, resulting in longer and unpredictable execution times, and ultimately deadline misses. The predicted energy savings may also be exaggerated from the actual scenario. This is because in a cache-aware schedule, which safeguards against cache-violations, a task may be delayed due to insufficient CPs even if an idle core is available, thus, resulting in comparatively less slack for DVFS to utilize.

We, thus, define a *valid schedule* as one that ensures timely completion of tasks while preventing any cache violations. Making use of the model proposed in the previous section, Figure 4-1 displays a scenario where DVFS is performed on a system with $M = 2$ cores and $W = 8$ CPs for $N = 4$ tasks with e_n and a_n values for each task as $\Gamma = \{\tau_1 < 3, 4 >, \tau_2 < 2, 2 >, \tau_3 < 2, 4 >, \tau_4 < 2, 7 >\}$. In Figure 4-1 (a), tasks are partitioned without considering the task CP assignments. After DVFS, the tasks are elongated uniformly till the deadline to achieve considerable energy savings, however, the cache violations are evident. In Figure 4-1 (b), tasks are partitioned while taking the CP assignments into consideration. The inter-core dynamic CP interference, i.e., blocking imposed by tasks due to their CP assignments, prevents some tasks from execution. This unavailability of CPs increases the schedule length compared to the one in Figure 4-1 (a). Therefore, there is less slack available for the DVFS resulting in reduced energy savings, but with the benefit of no cache violations.

In the following sections, we will present techniques to tackle this dynamic CP-aware energy minimization problem for system-level energy consumption while ensuring valid schedules.

4.2 Makespan Minimization

Minimizing the makespan of the schedule is a favorable first step to minimize the energy-consumption for independent frame-based tasksets [30]. Makespan minimization is analogous to the bin-packing problem which is NP-Hard. To keep up with the literature, we also attempt to minimize the makespan for our problem setup. However, considering the CPs of the tasks during makespan minimization increases the intractability of the problem. This is because, along with reducing the schedule length, the cache constraints must also



(a) Without Cache Considerations (b) With Cache Considerations

Figure 4-1: Energy minimization with cache contention

be ensured throughout the schedule, i.e., at any point in time, the sum of CPs required by the executing tasks cannot exceed the total partitions available in the cache.

As stated in Section 4.1, tasks executions are statically assigned to specific cores while CPs are mapped according to the availability of cache ways. Thus, the goal is to find a task-to-core mapping Π, i.e., $\tau_n \rightarrow o_m \quad \forall \tau_n \in \Gamma \wedge \forall o_m \in M$, such that the schedule length is minimized. During the mapping process, the execution time e_n can only be mapped to an idle core. Furthermore, e_n must be pushed forward on the core's execution path if there are insufficient idle CPs to accommodate τ_n's CPs a_n even if the core is idle. Therefore, the following constraints must be ensured i) Core constraint: the number of tasks placed in parallel does not exceed the number of cores and ii) Cache constraint: the CPs utilized by the tasks in parallel do not exceed the total number of CPs.

However, for homogeneous cores, i.e., identical cores, if we can ensure that the number of parallel tasks do not exceed the core count, it is not necessary to determine the specific core to which a task should be mapped. Rather, the task can be mapped to any of the idle cores. This reduces the complexity of the problem and allows it to be simplified to the Two-Dimensional-Strip-Packing-Problem (2DSPP) [82].

The goal in the 2DSPP is to minimize the height required to pack a set of rectangular objects onto a vertical strip of fixed width. A task's CPs a_n, execution-time e_n and schedule-

length of the taskset can be considered as the rectangle width, rectangle height and strip height, respectively in the 2DSPP. Ensuring that the number of rectangles placed in parallel do not exceed the number of cores, if a rectangle is placed at a position in the strip, there must be a core available at that point in the execution schedule to execute the task. The result of the 2DSPP problem is the Cartesian position of each rectangle in the strip which can be transformed into the placement of each task's CP requirement in the original problem as shown in Figure 4-2 (a) and (b).

Since this problem is also NP-Hard, we propose a heuristic solution for the 2DSPP along with the extra restriction on the number of tasks that can be placed in parallel.

Algorithm 3 gives the pseudo-code of our proposed heuristic solution. The algorithm iteratively searches for empty spaces in the strip to best-fit a task. If no task can be placed in the selected gap, the gap is vertically filled so that a new gap can be found in the next iteration (lines: 7-9). Since the tasks are placed bottom to top, if a task placement results in the number of parallel tasks to exceed the number of cores, the task is removed and the gap is vertically filled at the bottom by 1 unit before making another attempt in the next iteration (lines: 10-12). After a task is successfully placed, its corresponding core allocation in the original problem can be made by selecting an idle core with the minimum finish time (lines: 14-16).

ALGORITHM 3: 2DSPP Makespan Minimization

```
1: Output: r_n, R_n, Π;
2: Input: Γ, M;
3: Sort τ_n ∈ Γ in decreasing order of a_n;
4: while ∃τ_n ∉ Π do
5:    Find lowest free gap in the strip;
6:    Find τ_n that can best-fit into the gap;
7:    if τ_n cannot fit into gap completely then
8:       Fill the gap;
9:       Continue;
10:   else if τ_n placement will violate core constraint then
11:      Fill the gap vertically by 1 unit;
12:      Continue;
13:   else
14:      Place τ_n in gap;
15:      Assign τ_n to idle core with minimum finish time;
16:   end if
17: end while
```

With a method to obtain a valid task-to-core mapping that minimizes the schedule length, the following section investigates different DVFS techniques that will maintain a

valid schedule.

4.3 Valid Schedules

Assuming a cache-aware pre-mapped taskset Π, it remains to minimize the energy consumption of the tasks while ensuring the schedule is valid. Without any method to model the inter-core dynamic CP interference, one way to apply DVFS without causing cache violations is to stretch the exact schedule uniformly until the deadline, i.e. the elongated schedule is a stretched version of the original schedule as shown in Figure 4-1 (b). This is proved in the following lemma.

Lemma 1. *Without any method to model the inter-core dynamic CP interference, stretching the schedule uniformly until the deadline will always result in a valid schedule.*

Proof. Consider a cache-aware pre-mapped valid schedule Π. Suppose that the original mapping is stretched by a factor s resulting in a schedule Π_s where $s = D/R_l$ and R_l is the finish time of the last task in the schedule. The stretched version of the original schedule ensures that any tasks τ_j that start after τ_i, i.e., $r_j = R_i + \Delta$ in Π will still continue to do so by the factor s, i.e., $r_j s = R_i s + \Delta s$ in Π_s since both tasks τ_i, τ_j and slack between the tasks Δ are elongated by the same factor s. Since all the tasks and their related slacks are elongated by the same factor, there will be no extra overlap of tasks executions or task CPs. Thus, ensuring Π_s to be a valid schedule. □

Following Lemma 1, we must now determine a DVFS scheme that constructs a valid schedule.

Corollary 2. *For a simplistic task model where all tasks are assumed to have the same power characteristics, static global DVFS can be used to construct a valid schedule.*

Proof. Under static global DVFS, all tasks will execute at a common global frequency f_g. As the tasks have homogeneous power characteristics, the activity factor κ_n (defined in Chapter 3: Eq: 3.9) for all tasks will be the same. That is,

$$\frac{\kappa_a cc_i + \kappa_s mc_i \delta_i}{cc_i + mc_i \delta_i} = \frac{\kappa_a cc_j + \kappa_s mc_j \delta_j}{cc_j + mc_j \delta_j} \quad \forall i \forall j$$

Since applying the global frequency implies that the ratio of compute to memory cycles for all task must be same,

$$\frac{cc_i}{mc_i} = \frac{cc_j}{mc_j} \quad \forall i \forall j$$

the resultant scaling factor will also be the same for all tasks

$$s_i = s_j \quad \forall i \forall j$$

where $s_i = D/f_g$ □

For tasks with different power characteristics, global DVFS will no longer guarantee a valid schedule.

Corollary 3. *For tasks with different power characteristics, each task τ_n will run at a frequency $\hat{f_n}$ in order to obtain an valid schedule*

Proof. Given that each task is scaled by the global scaling factor s, we must find find the frequencies of each task that will cause the tasks to scale by the same factor s.

$$\hat{e_n} = e_n s$$

$$\frac{cc_n}{\hat{f_n}} + \frac{mc_n^{a_n}}{f_{mem}} = s(\frac{cc_n}{f_n} + \frac{mc_n^{a_n}}{f_{mem}})$$

$$\hat{f_n} = \frac{cc_n f_n}{cc_n s + \frac{mc_n^{a_n}}{f_{mem}} f_n s - \frac{mc_n^{a_n}}{f_{mem}} f_n}$$

$$\hat{f_n} = \frac{cc_n f_n}{cc_n s + \frac{mc_n^{a_n}}{f_{mem}} f_n (s-1)}$$

□

Though this approach results in a valid schedule, it restricts the energy-saving capabilities of the DVFS method since the schedule pattern remains the same and slack may still be available between the task executions for DVFS to utilize. However, since there is no method to model the inter-core dynamic CP interference, elongating any task with hopes to utilize the unused slack may result in cache-violations.

Lemma 2. *Without any method to model the inter-core dynamic cache contention, techniques other than that proposed in Lemma 1 may result in an invalid schedule*

Proof. By contradiction, we assume that without any method to model the inter-core dynamic CP interference, other ways to elongate the tasks will always result in a valid schedule Π_c. We consider two tasks τ_i and τ_j mapped onto different cores where $r_j = R_i + \Delta$ and $a_i + a_j > A$. Contrary to the schedule proposed in Lemma 1, Π_c will elongate the tasks τ_i and τ_j and slack between them Δ by different scaling factors, i.e., τ_i is scaled by a factor s_i and Δ is scaled by a factor s_Δ. If $s_i >> s_\Delta$ such that $r_j s_j < R_i s_i + \Delta s_\Delta$, then the resultant elongations will result in a CP overlap and, therefore, a cache-violation, which contradicts the assumption that Π_c is a valid schedule. □

Based on the analysis presented above, uniformly stretching the schedule will always guarantees a valid schedule. However, this method nullifies the decades of research put into trying to utilize the available slack most effectively. In the following section, we propose our novel approach to model the inter-core dynamic CP interference to achieve better energy savings.

4.4 Cache Dependency Graph (CDG)

This section presents our novel technique to model the inter-core dynamic cache contention. At this point, we assume that the taskset is mapped across the cores using the heuristic algorithm presented in Section 4.2. We now model the CP interference of the allocated tasks by creating dependencies among tasks that are blocked due to insufficient CPs.

Our proposed technique converts the independent mapped taskset into a dependent taskset based on the inter-core dynamic cache-contention as shown in Figure 4-2 (b) and (c).

4.4.1 Rules For Creating Dependencies

Given a pre-mapped taskest Π, we introduce the concepts of *single-blocking* and *poly-blocking* on a DAG representation of Π.

Definition 1: τ_j is said to be *single-blocked* by τ_i if there is a single edge from node τ_i to node τ_j, shown as dotted edges in Figure 4-2 (c).

The rules for creating single-blocking dependencies among the tasks are as follows:

(a) *Same-core Blocking:* Given two tasks τ_i and τ_j executed in succession on the same core, there will be a single-blocking edge from τ_i to τ_j

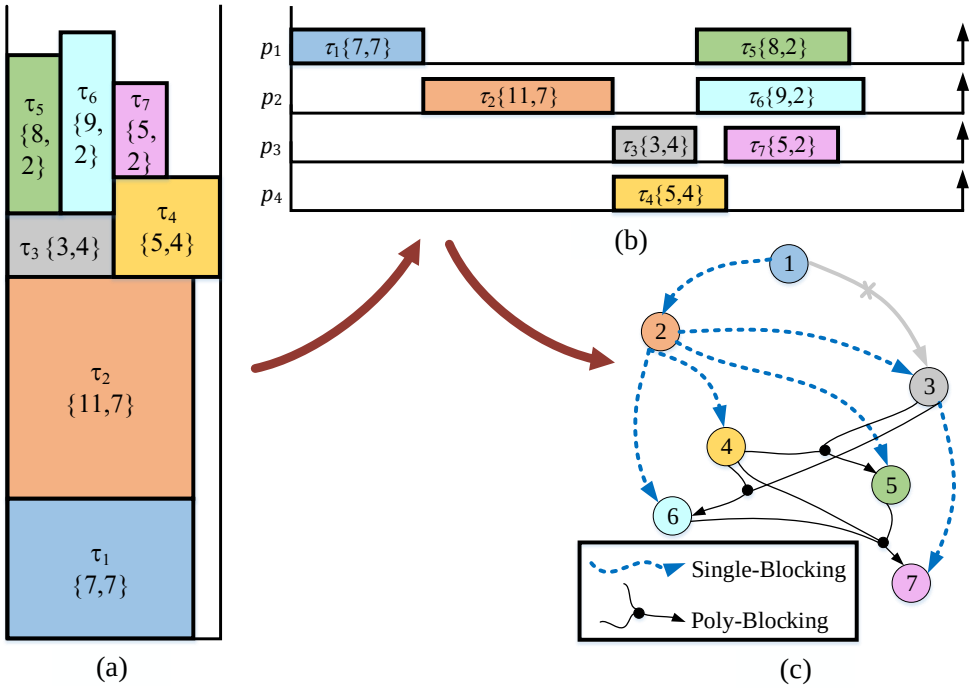


Figure 4-2: Makespan minimization for a cache-aware independent frame-based taskset $\Gamma = \{\tau_1\{7,7\}, \tau_2\{11,7\}, \tau_3\{3,4\}, \tau_4\{5,4\}, \tau_5\{8,2\}, \tau_6\{9,2\}, \tau_7\{5,2\}\}$ with M=4 and W=8; (a) Illustration of 2DSPP, (b) Corresponding task-to-core mapping and (c) Resultant CDG.

(b) *Cache Blocking:* Given two tasks τ_i and τ_j to be executed on different cores where $R_i \leq r_j$, there will be single-blocking from τ_i to τ_j if $a_i > A - a_j$

(c) *Chain Blocking:* If τ_i single-blocks both τ_j and τ_k, and τ_j single-blocks τ_k. We can then ignore the dependency between τ_i and τ_k due to the chain blocking, therefore, simplifying the DAG.

Definition 2: τ_j is said to be *poly-blocked* by a combination C of two or more tasks if there is a combinatorial edge from nodes $\tau_i \in C$, shown as solid edges in Figure 4-2(c). The rule for creating poly-blocking dependencies among the tasks is a follows:

(a) For tasks τ_i which start with or before τ_j and cannot separately single-block τ_j, there will be poly-blocking among different combinations of these tasks. Such tasks can be grouped into a set $PB_j \leftarrow \tau_i : r_i \leq r_j \wedge a_i \leq A - a_j$. We define a poly-blocking combination $C \subset PB_j : \sum_{\tau_i \in C} a_i > A - a_j$.

4.4.2 Algorithm for the Creation of CDGs

In Algorithm 4, tasks are first sorted in ascending order of their start times (line: 3). A poly-dependency queue *PolyQ* is created to collect candidates for poly-blocked tasks

(line:4). The outer loop (lines: 5-22) iteratively selects each task τ_i from the taskset. Counter *count* is used to prevent a task from creating further dependencies once successive tasks on each core are single-blocked. The inner loop (lines: 7-21) then selects the next task τ_j from the taskset. If Same-Core or Cache single-blocking dependencies exist, single-blocking dependencies are created accordingly (lines: 8-14). If none of the conditions are satisfied, then the task is considered a candidate for poly-blocking (lines: 15-20). τ_j is added to *polyQ* and τ_i is added to PB_j accordingly. Once all the single-blocking dependencies are recognized, another loop (lines: 23-26) creates poly-blocking dependencies. Tasks are iteratively extracted from *polyQ*, and all combinations of tasks in the PB_j that can poly-block τ_j under the poly-blocking dependency rule are added to the DAG. This DAG then becomes a CDG and can be used by existing energy-efficient algorithms to scale the tasks via DVFS to minimize energy consumption.

ALGORITHM 4: Creating Cache Dependency Graph

```
1: Output: CDG
2: Input: Γ, Π
3: Sort τ_i ∈ Γ in ascending order of their r_i;
4: Create PolyQ;
5: for τ_i ∈ Γ do
6:    Initialize count to equal M;
7:    for τ_j ∈ Γ : r_j ≥ r_i do
8:       if Same-core || Cache (Blocking) then
9:          if Chain Blocking then
10:            Ignore single-blocking τ_i ! → τ_j;
11:         else
12:            Create single-blocking τ_i → τ_j;
13:         end if
14:         Decrement count and exit loop if equals 0;
15:      else
16:         if τ_j ∉ polyQ then
17:            Add τ_j to polyQ;
18:         end if
19:         Add τ_i to PB_j ;
20:      end if
21:   end for
22: end for
23: for τ_j ∈ polyQ do
24:   Find all combinations of τ_i ∈ PB_j that will block τ_j;
25:   Remove τ_j from polyQ;
26: end for
```

Note that the makespan minimization technique presented in the Section 4.2 is specific for independent frame-based tasksets with a common deadline. However, frame-based

tasksets can also have arbitrary deadlines or precedence constraints, in which case other techniques are required to map the tasks successfully onto the cores. Nonetheless, the CDG approach presented in this section can be used for tasksets with arbitrary deadlines and precedence constraints assuming that they are already mapped across the cores. This is because the CDG mechanism will simply add dependencies to the existing dependencies of the precedence constrained taskset and the arbitrary deadlines will be catered for via the energy-efficient algorithm selected for energy minimization.

4.5 Energy Minimization

In Section 4.4, we presented a technique to model the inter-core dynamic CP interference in the form of a CDG. Thus, the presented model can be adopted by existing, well-established, energy-efficient algorithms to avoid cache-violations while minimizing the energy consumption for a cache-aware scenario.

Energy minimization of the core subsystem has been the focal point for most multicore energy-efficient algorithms in the literature. However, a relatively large portion of the processor is occupied by caches contributing to a large percentage of the overall energy consumption. With the continuous increase in cache sizes and the involvement of multicores and many-cores, this percentage is likely to grow. Consequently, the contribution made by caches to energy consumption can no longer be ignored. Thus, with the proposed interference model, the cache energy consumption can be easily accommodated in the energy minimization problem.

Using the 68 nm technology processor parameters [29, 60, 112] for the system model proposed in Section 4.1, we show how the core subsystem, cache subsystem and system-wide (core+cache) energy consumption can change with respect to the core-voltage V_n during τ_n's execution and the CPs a_n assigned to τ_n in Figure 4-3. The dynamic energy consumption of the core subsystem for a task decreases when V_n is decreased, however, it increases when a_n is increased (Figure 4-3 (a)). This is because a smaller a_n increases the memory latency which increases the overall execution length of the task. A longer execution length for the same V_n results in higher energy consumption. Contrary to what appears in Figure 4-3 (c), the static energy consumption of the core should increase with V_n. However, the decrease, in this case, is due to the task model adopted in this study. Since the

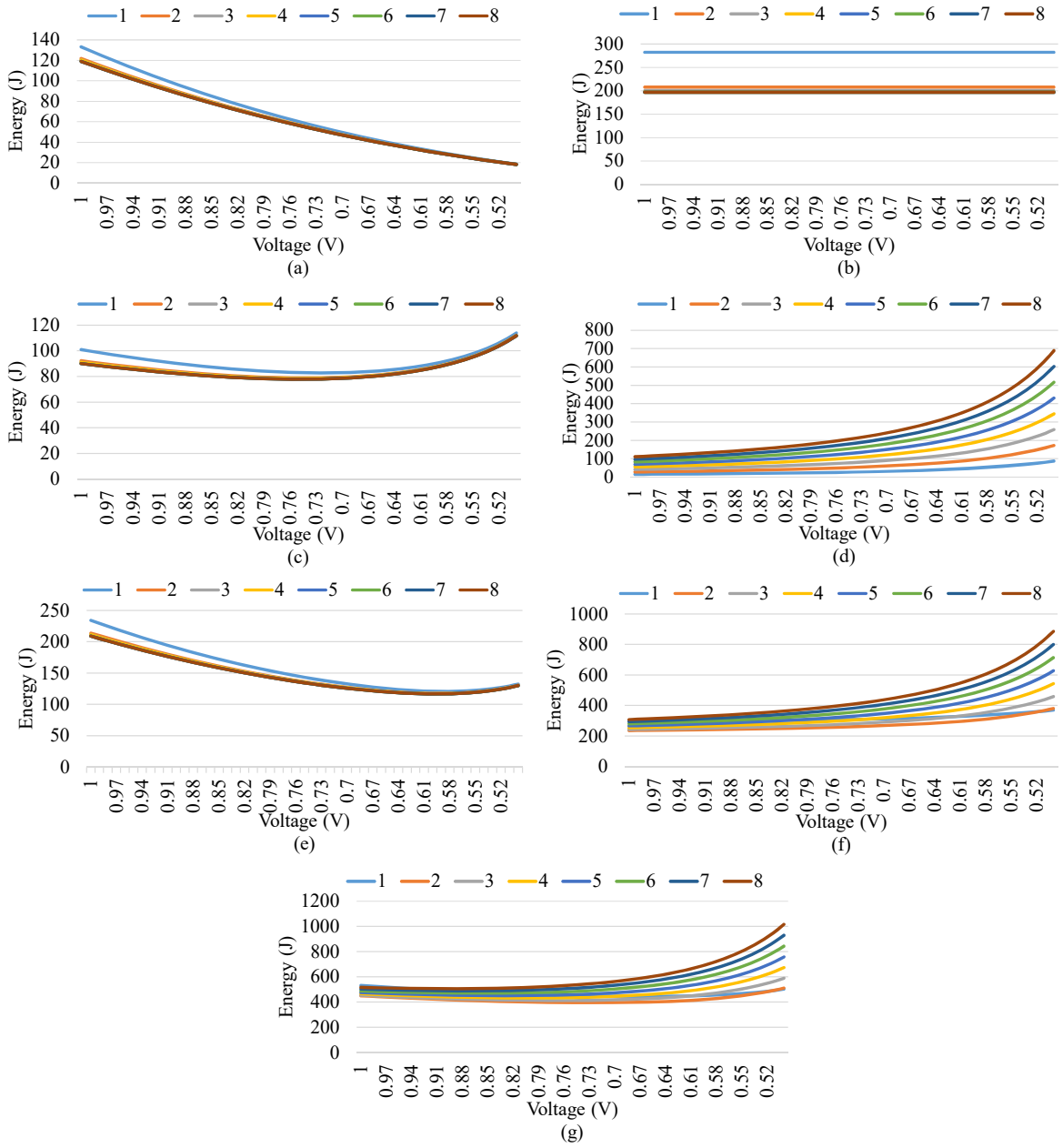


Figure 4-3: Impact on energy consumption of a τ_n $fluidanimate(PARSEC)$ by varying the voltage V_n and CPs a_n. (a) Dynamic energy consumption of core. (b) Dynamic energy consumption of cache. (c) Static energy consumption of core. (d) Static energy consumption of cache. (e) Total energy consumption of core. (f) Total energy consumption of cache. (g) Total energy consumption of complete system.

change in voltage only affects the cc_n portion of the task rather than the complete execution length, the increase in execution length is not as significant as the decrease in static power consumption resulting in an overall decrease in energy consumption. This pattern is more apparent for a smaller a_n, e.g., $a_n = 1$, since a smaller CP assignment decreases the cc_n to mc_n ratio. However, for a larger a_n where the cc_n overwhelms the mc_n, the conventional trend is still noticeable. The overall core energy consumption is seen to decrease as V_n is decreased. However, the energy curves are non-monotonic and the critical voltage, i.e., the voltage at which the energy consumption is minimum, of the task is still apparent, e.g., critical voltage is 0.55 V for $a_n = 8$ (Figure 4-3 (e)).

V_n has no impact on the dynamic energy consumption of the cache subsystem (Figure 4-3 (b)). However, reducing the number of CPs increases the cache miss-rate which increases the dynamic cache energy consumption in accordance with cache energy model in Chapter 3. For smaller values of V_n, the static energy consumption of cache is comparable to its dynamic counterpart (Figure 4-3 (d)), however, it increases significantly as the V_n is decreased since the CPs are active for a longer period of time. Furthermore, reducing a_n reduces the static cache energy consumption. The overall cache energy consumption can be seen in Figure 4-3 (f). Finally, Figure 4-3 (g) sums up the energy consumptions of both core and cache subsystems and it is apparent that both core-voltage and CP variations have a significant impact on the system-level energy consumption.

In this section, we investigate this fact further by proposing techniques for a 3D cache-aware energy minimization problem, i.e., minimizing the core-, cache-, and system-level energy consumption. For each sub-problem, we propose techniques that aim to minimize the energy consumption of the specified component irrespective of its impact on the other components of the system. Experimental results for the proposed techniques are also presented in this section. The performance metrics are based on the energy savings obtained by adopting the proposed techniques against the energy consumed by the taskset in its original state, i.e., without DVFS.

Since this study is the first of its kind in proposing a 3D cache-aware energy minimization problem for homogeneous multicore real-time systems, the performance metrics cannot be compared to those present in the literature. Thus, for baseline comparison, the energy savings obtained by the proposed approach are compared against the energy savings obtained by stretching the schedule uniformly till the deadline, which, to the best of our

knowledge, is the only way to guarantee a valid schedule while performing DVFS when there is no method to model the inter-core dynamic CP interference. We have mentioned this fact previously in Section 4.3. Along with the energy savings for the specific optimization component for each sub-problem, we also demonstrate the impact of the specified technique on the energy consumption of the other components of the system. For instance, for the core-level energy minimization problem, we present the energy savings on the core along with its impact on the cache and complete system energy consumption.

For the core-level energy minimization problem, we adapt existing algorithms to the CP interference model. For the cache-level energy minimization problem, we propose a heuristic that reduces the CPs for each task based on a CP-gradient metric. We then combine both core- and cache-level techniques to minimize system-level energy consumption.

4.5.1 Experimental Setup

To evaluate the effectiveness of our approach, we conducted a series of simulations using real-time workloads for the proposed system and task model.

Workload

We used the PARSEC [19] benchmark to represent real-time workloads that have been extensively used by the real-time community [67, 121]. We collected the instruction count, load/store ratio and miss-rates for each benchmark with various cache sizes and levels of associativity using the Gem5 Simulator [20]. The collected benchmark statistics were then used to create a synthetic benchmark, similar to the approach presented [23, 67]. Specifically, the cache hit and miss delay, taken from [67], along with instruction count, the load/store ratio and miss-rates, collected from Gem5, were used to model the cc_n and mc_n [54]. Table 4.1 gives an example of how mc_n changes for different values of a_n.

A taskset is generated by randomly selecting one of the benchmarks from within the suite for different taskset sizes [67]. We adopt the method proposed in [63] to set the deadline of the taskset as: $D = (1 + d_f) \times R_l$, where d_f is the deadline factor which can be set to $0.0 (D = R_l), 0.05, 0.1$, etc., and R_l is the finish time of the last task in the schedule.

Table 4.1: Example of scaled mc_n vs. a_n for PARSEC benchmarks

a_n	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**Benchmarks**
mc_n	70	45	42	41	41	41	41	41	blackscholes
	93	73	67	65	64	62	62	62	canneal
	70	66	64	62	56	43	41	41	streamcluster
	29	25	23	23	22	22	22	22	swaptions

System model

The system model parameters are taken from the classical and verified energy model of the 68 nm technology processor [29, 60, 112]. f_{mem} is fixed at the maximum frequency of the core for this model. The switching activity factor, however, is determined from the active and sleep mode switching activity factors presented in [126], i.e., $\kappa_a = 0.51, \kappa_s = 0.22$. The energy consumption parameters for the cache are retrieved from CACTI for 68 nm technology [109] for a cache-line size of 64bytes. All of the results are shown for a 4 core system with an 8-way 2 MB shared cache. However, we have also shown the trend in energy consumption for varying number of cores (Figure 4-6) and cache-sizes (Figure 4-7).

4.5.2 Core-level Energy Minimization

The goal of this section is to minimize the energy consumption of only the core-subsystem in a cache-aware scenario. Since many core-energy minimization algorithms are already present in the literature, we propose to utilize these existing algorithms to demonstrate the effectiveness of our inter-core dynamic CP interference model.

Existing algorithms utilize the slack available between task executions to elongate selected tasks based on specified criteria. They then determine whether successor tasks must be pushed forward in the schedule after elongation of a predecessor task via a DAG [84, 87, 102]. These elongations continue until task deadlines are met.

The cache-related dependencies created among the tasks in our CDG model can be used to mimic dependencies among predecessor and successor tasks. However, due to the diverse nature of poly-blocking dependencies, successor nodes in the CDG are prevented from unnecessary pushing, i.e., a successor is only pushed forward if the finish times of all poly-blocking predecessors exceed the start time of the successor.

For our evaluation, we have used the algorithm presented by Schmitz et al. [102] that

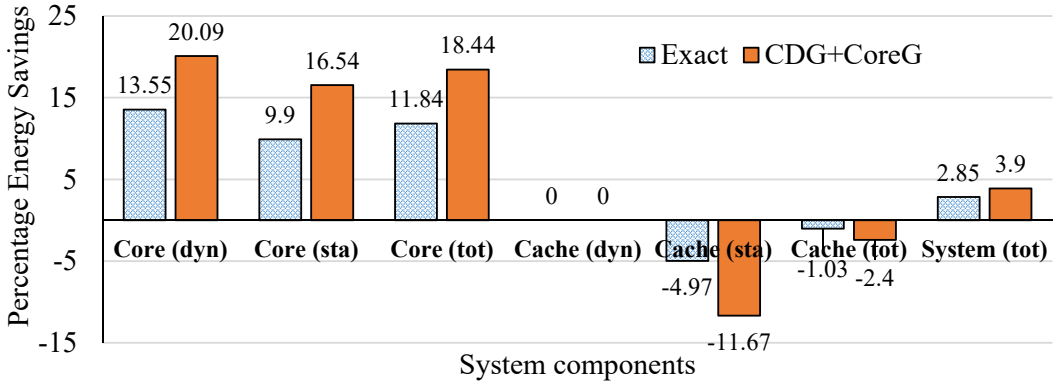


Figure 4-4: Core-level energy minimization with $d_f = 0.05$ and $N = 30$.

exploits a task's energy-gradient to minimize the cores' energy consumption. The energy-gradient is defined as the change in energy consumption of a task when the voltage/frequency of the core, during a task's execution, is reduced. Tasks with higher energy-gradients are iteratively selected for elongation until the deadline is met.

Core-level Techniques

Figure 4-4 shows the energy savings achieved using our CDG method against the baseline technique termed as Exact. Our proposed CDG model is combined with the energy minimization algorithm proposed in [102], termed as CDG+CoreG. The results display the energy savings obtained for each subsystem, i.e., core, cache and complete system (core+cache) along with the contributions made by the dynamic and static components of each subsystem.

We will use Core (dyn), Cache (dyn), Core (sta), Cache (sta), Core (tot), Cache (tot) and System (tot) for Core dynamic, Cache dynamic, Core static, Cache static, Core total, Cache total, and System-level total energy consumptions, respectively in all future results and discussions.

Results show the proposed CDG method to outperform the baseline technique. The dynamic energy portion of the cache remains the same since the number of CPs for each task remains constant resulting in a constant miss-rate. However, the static energy consumed by the cache increases since the CPs for each elongated task, brought about by DVFS at the core-level, are active for a longer time. Nevertheless, the overall system energy consumption is decreased, though heavily deterred by the cache subsystem.

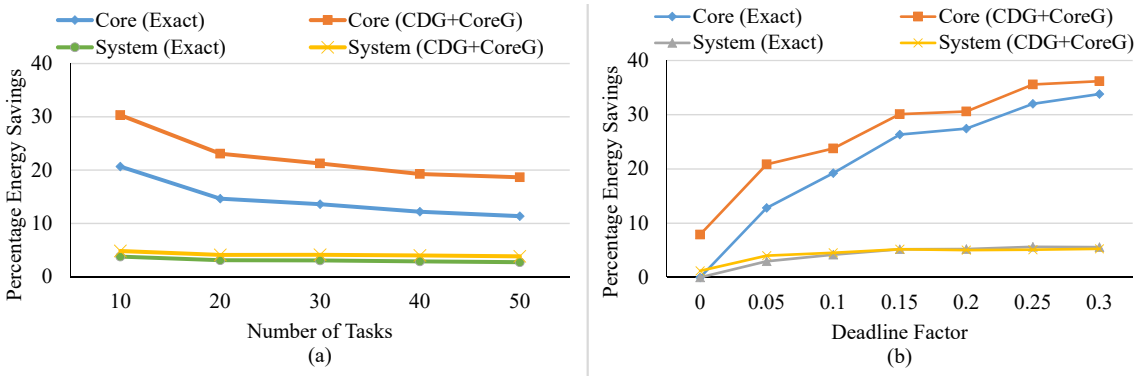


Figure 4-5: Core-level energy minimization against different values of n and d_f

Change in Task Parameters

Figure 4-5 (a) shows a change in energy savings as the number of tasks are increased from 10 to 50. Energy savings for the proposed CDG approach are higher for all ranges of the number of tasks. However, the energy savings as the number of tasks increase. This is because the CDG approach also benefits from the slack among task executions to minimize energy consumption. An increase in the number of tasks increases the inter-core dynamic contention along with a decrease in the amount of slack available for each task to utilize.

Figure 4-5 (b) shows how the energy-saving vary as the deadline-factor d_f in increased from 0.0 to 0.3. There are no energy savings for the baseline approach when d_f is set to 0. This is because the schedule cannot be stretched since there is no slack after the finish time of the last task. However, the CDG approach allows tasks to utilize the slack between task executions, thus, permitting significant energy savings. As the deadline factor is increased, the core energy savings increase for both Exact and CDG+CoreG due to an increase in available slack. However, the increase is concave due to the simultaneous increase in the static component of the core energy consumption. Furthermore, the system energy savings begin to saturate after $d_f = 0.15$ at which point the energy consumed by the cache outweighs the core energy savings due to CPs being active for a longer period of time.

Change in System Parameters

Figure 4-6 shows the change in energy savings as the core-count is varied. Energy savings tend to decrease as the number of cores are decreased. This is because our approach takes advantage of the blocking due to insufficient CPs to increase energy savings. Therefore,

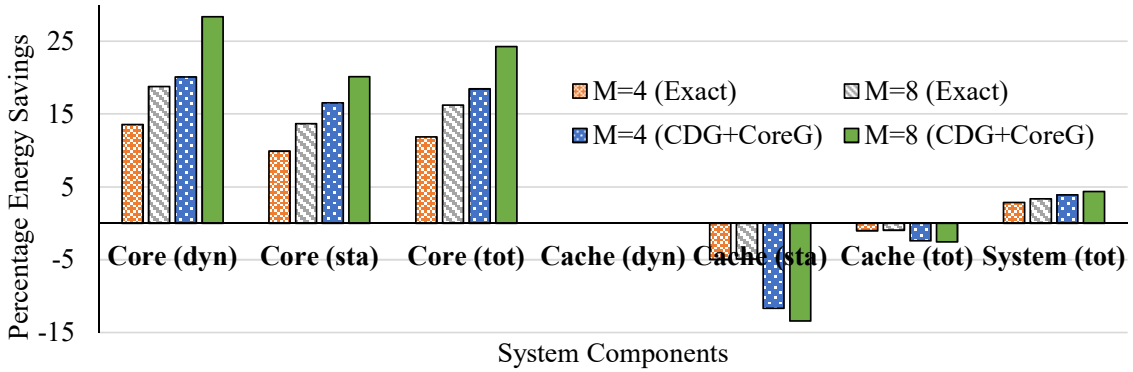


Figure 4-6: Core-level energy minimization vs. number of cores M

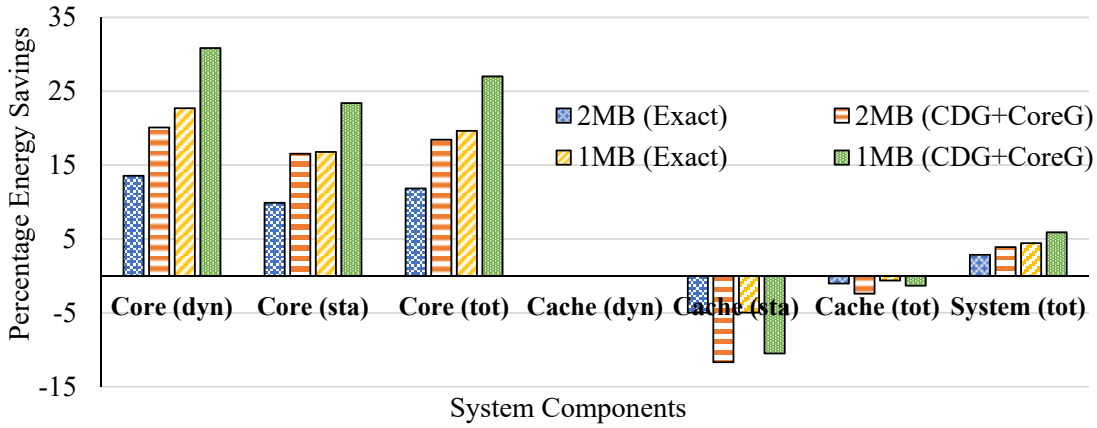


Figure 4-7: Core-level energy minimization vs. cache size

with fewer number of cores, the blocking imposed on the tasks is due to insufficient number of cores rather than insufficient CPs, thus, resulting in smaller energy saving. As M is increased, the inter-core dynamic blocking due to insufficient CPs in also increased, thus, enabling more energy savings. In all cases, the proposed CDG method outperforms the baseline technique.

Figure 4-7 shows that the ratio in energy savings between the baseline and CDG approach remains approximately the same across cache sizes. However, the overall energy savings reduce as the cache size is increased. This is because increasing cache size increases the cache energy consumption and so reduces the impact on energy savings from algorithms that focus on core-level energy minimization.

4.5.3 Cache-level Energy Minimization

The cache energy consumed by a task depends on the task miss-rate, execution length and number of CPs assigned to it. In the previous section, the execution length of a task was altered via the inherent DVFS capabilities of the core subsystem while the CPs of each task were kept constant. Changing the number of CPs, however, is a property of the cache subsystem. Thus, in this section, we propose an algorithm to change the number CPs assigned to each task with the aim to minimize the energy consumption of the cache subsystem. Decreasing the number of CPs used by a task can benefically reduce the static energy consumption of the cache subsystem. However, the problem is non-trivial as any decrease in CPs may increase the miss-rate which can in turn increase the dynamic energy consumption of the cache subsystem. Doing so can also increases the number of memory cycles, thus, increasing the execution length of the task and resultantly increase the static energy of the core subsystem. Moreover, due to the increase in task execution length, CP reductions must be performed carefully to avoid deadline misses.

Hence to efficiently utilize the slack for task elongations, tasks are selected for CP reductions based on a CP-gradient metric. The *CP-gradient* is defined as the change in execution time of a task when its CPs are decreased.

$$\Delta e_n = e_n(a_n) - e_n(a_n - 1)$$

The algorithm iteratively selects a task with the lowest CP-gradient so that minimum slack is used. Since decrementing the CPs increases the dynamic energy, the CPs of a task are not reduced if the resultant energy consumption increases. Furthermore, decreasing the CPs of the tasks also decreases the inter-core dynamic contention. As a result, the inter-core dynamic blocking is decreased permitting additional task elongations.

Cache-level Technique

The results for the proposed approach are shown in Figure 4-8. As discussed above, there is a slight increase in the dynamic energy consumption of the cache subsystem. The energy savings for the static portion of cache are substantially reduced resulting in an overall decrease in cache energy consumption. However, the core energy consumption is significantly increased due to an increase in the execution length of the tasks while the core-level power

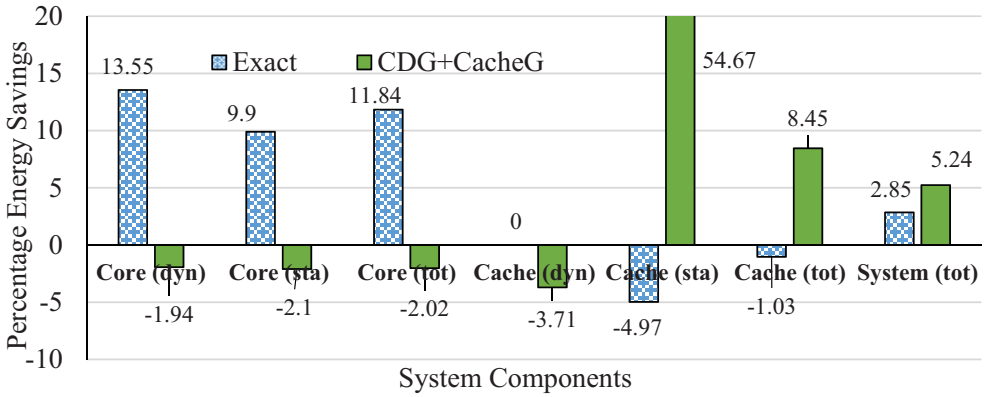


Figure 4-8: Cache energy minimization with $d_f = 0.05$ and $N = 30$.

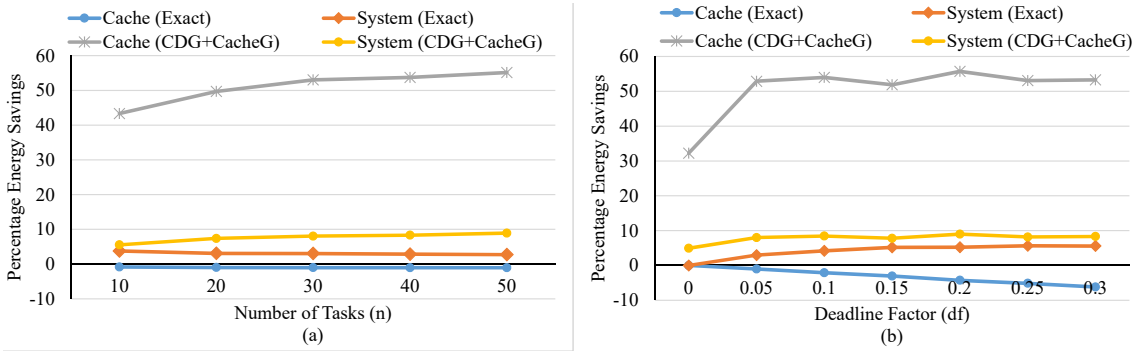


Figure 4-9: Cache energy minimization against values of n and d_f.

consumed by each task remains the same.

Changing Task Parameters

Figure 4-9 (a) and (b) show the change in energy savings as the number of tasks and deadline factor are increased. For all cases, the energy savings from the proposed approach are higher than the baseline approach. The energy savings tend to increase as the number of tasks are increased because the algorithm can reduce a greater number of active CPs. A similar trend is observed as the deadline-factor is increased.

4.5.4 System-level Energy Minimization

In the previous two sections, we demonstrated how the energy consumption of the different subsystems were influenced by the techniques that focused on energy minimization specific to the core and cache subsystems. In this section, we modify the techniques proposed for the specific subsystems to minimize system-level energy consumption. Specifically, we

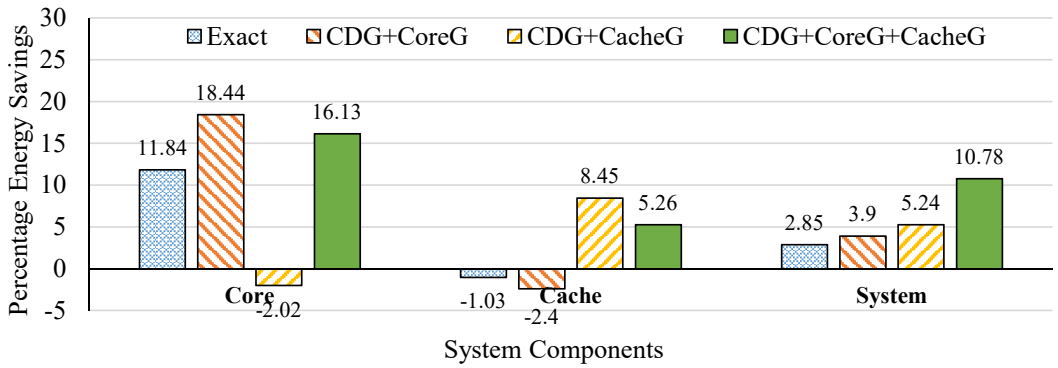


Figure 4-10: System-level energy minimization with $d_f = 0.05$ and $n = 30$.

extend the techniques proposed for core subsystem, i.e, CDG+CoreG, and cache subsystem, i.e., CDG+CacheG for system-level energy minimization termed as CDG+CoreG+CacheG. CDG+CoreG+CacheG simultaneously reduces the CPs of the lowest CP-gradient task along with a reduction in the operating core voltage of the highest energy-gradient task.

The extended algorithm, targeting system-level energy minimization, attempts to balance the energy consumed by the core and cache subsystems. Consequently, it is not as effective as the algorithms targeting energy optimizations of individual subsystems as can be seen in the left and center set of graphs in Figure 4-10. However, as far as system-level energy minimization is concerned, the extended algorithm outperforms the baseline (Exact) and proposed techniques targeting individual subsystems (CDG+CoreG, CDG+CacheG) as shown on the right-most set of graphs in Figure 4-10. There is approximately a 5× increase in the energy savings over the baseline case for CDG+CoreG+CacheG.

4.5.5 Comparison with the Optimal

In Section 4.2, we proposed a cache-aware method to map a taskset onto the cores in order to minimize the makespan. However, makespan minimization does not guarantee an energy minimization schedule. Therefore, we have compared the proposed approach against the optimal schedule for a taskset with 8 tasks running on a 4 core machine with 8 CPs. The optimal solution is found by searching for all the possible mappings and selecting the schedule that results in the minimum energy consumption via the CDG+energy minimization techniques discussed in the previous sections. Figure 4-11 displays the energy savings achieved using our proposed 2DSPP mapping technique compared to the optimal. For core-level energy minimization, i.e., CDG+CoreG, the optimal solution results in 1.4×

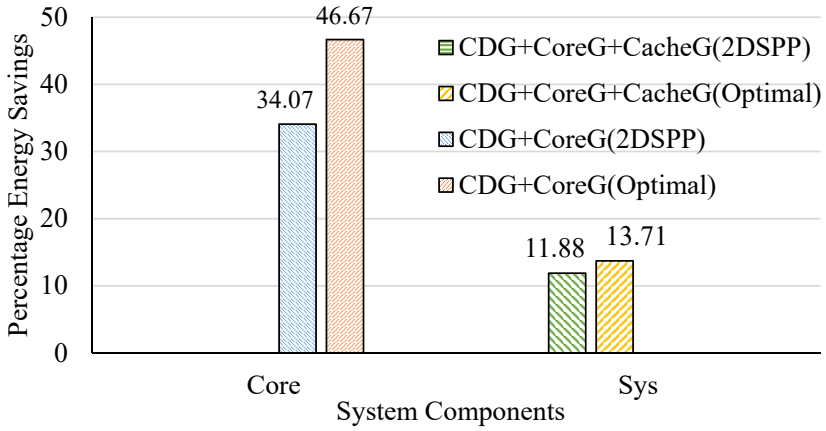


Figure 4-11: Energy savings comparison of optimal vs. 2DSPP Core- and system-level energy minimization

energy savings over the proposed 2DSPP approach. However for the system-level solution, the results for the 2DSPP are comparable to the optimal.

4.6 Discussion

The adoption of multicores has opened up new paradigms for research in real-time systems. However, despite the advancements made, existing algorithms have failed to accommodate the impact of shared caches on the predictability and energy consumption of the processor. In this chapter, and in light of the growing influence that shared caches have on such systems, we formulated the problem of cache-aware energy minimization and proposed various techniques to reduce the energy consumption for valid schedules. The proposed energy optimization techniques outperform the baseline case for individual core and cache subsystems as well as system-level (core+cache). Specifically for the system-level energy minimization, a 5× energy improvement is observed against the base-line approach. With the increasing popularity for heterogeneous multicore architectures in the domain of real-time systems, cache-contention related issues adds another interesting dimension to the schedulability and energy minimization problem. Thus, in the following chapter, we propose a solution for cache-aware energy minimization on heterogeneous multicore real-time systems.

Chapter 5

Cache-Aware Energy-Efficient Scheduling on Heterogeneous Multicores

Despite the significant contributions made towards energy-efficient deadline-constrained scheduling on heterogeneous multicores [81, 94, 105, 108], prior work is handicapped by two important limitations. First, it ignores the impact of shared caches on a task's execution time. Second, tasks are assumed to scale linearly with the core-frequency.

In this chapter, we introduce a new approach to energy-efficient scheduling on heterogeneous multicores which compensates for the simplistic assumptions made in prior work. First, we bound the shared cache unpredictability by adopting cache partitioning techniques where specific sections of the shared cache are individually assigned to cores to prevent concurrent tasks from accessing the same cache-lines. Second, we investigate the impact that the core-frequency and CPs have on the task execution cycles in a heterogeneous multicore environment, and its resultant impact on task execution time and energy consumption. To this effect, we present a model where the task characteristics dictate the task allocation strategy, i.e., we analyze the model by presenting proofs to determine the best core and core-type affinities based on the task characteristics. Finally, we present an energy-efficient scheduling algorithm that considers the nonlinear change in execution time brought about by the frequency and CPs assigned to the cores.

To assess the proficiency of our approach, we perform extensive evaluations using PARSEC [19] and Mibench [52] benchmarks on the ARM big.LITTLE architecture. Results show our approach to consume less energy than the two state-of-the-art solutions, achieving an average of 15.73%, 13.0% and a maximum of 35.0%, 31.7% in energy savings while ensuring tasks complete before their deadlines. To the best of our knowledge, this work is the first to analyze the effect that CPs have on influencing the task-to-core allocation in a heterogeneous multicore setting for hard real-time systems. In summary, contributions of this work are as follows:

- We investigate the effects of heterogeneity, core-frequency and CPs on the execution

cycles of a task.

- We propose an algorithm that performs efficient task-to-core mapping based on the task characteristics.
- We, finally, present results of our experimentation to show the proficiency of our proposed approach over the state-of-the-art.

In this work, we focus on a static CP scheme. This is due to the fact that a periodic task model is adopted in this work to match the state-of-the-art energy-efficient scheduling solutions for heterogeneous multicores. A dynamic CP scheme for periodic tasks will require careful analysis to ensure schedulability of the taskset. Initial investigations in this direction, i.e., on a dynamic CP analysis for periodic tasks are presented in Chapter 6.

5.1 Problem Setting

We start by analyzing the effects of heterogeneity, core-frequency and number of CPs on the execution cycles of a task. Figure 5-1 displays this relationship for PARSEC [19] and Mibench [52] application benchmarks. The execution cycles of selected benchmarks executing on the big.LITTLE Exynos 5 Octa (5422) processor are extracted using the Gem5 Simulator [21]. This processor is composed of 4-core clusters for A15 and A7 core-type each, with respective frequency ranges of (0.2 GHz-2.0 GHz) and (0.2 GHz-1.4 GHz) [105], respectively. A Last-Level Cache (LLC) is shared by cores of the same cluster. Figure 5-1 (a) shows the change in execution cycles w.r.t. frequency when each task is executed separately on a single A15 and A7 core and the selected core has complete access to its respective LLC, i.e., 8 CPs. It is observed that the execution cycles change by an average of 6.62%, 9.63% and maximum 13.95%, 20.64% between the frequency poles, for the A15 and A7 core-type, respectively. A higher frequency represents relatively higher execution cycles since the memory latency is not affected by the reduced clock-cycle length, thus, increasing the execution cycles contributed by the memory latency, as discussed in Chapter 3.

In a practical setting, however, cache partitioning must be established to preserve predictability in a deadline-constrained hard real-time system [68, 120]. For an equal sharing-scheme, each core in a cluster will be limited to only 2 CPs. The resultant change in execution cycles are depicted in Figure 5-1 (b), where there is an average of 14.29%, 15.23% and

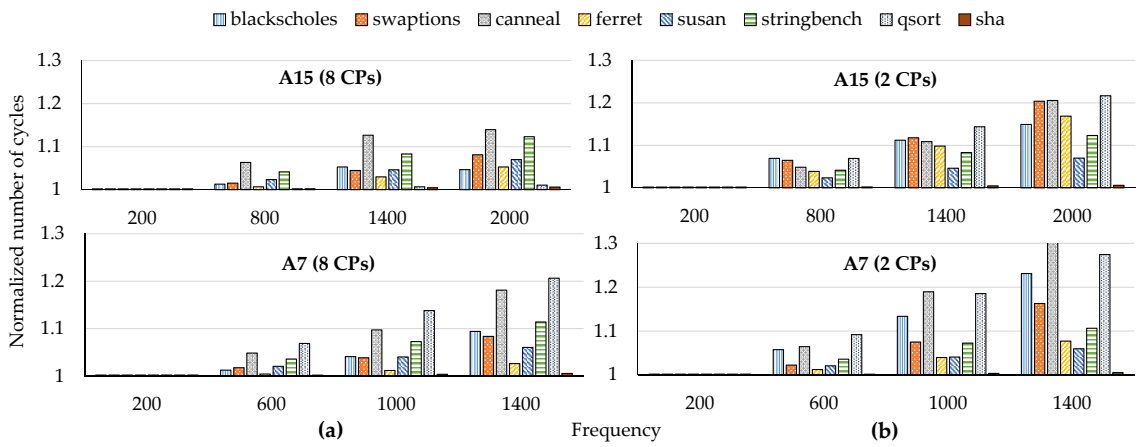


Figure 5-1: Change in execution-cycle count for applications from the PARSEC and Mibench benchmark suites running on the big.LITTLE Exynos 5 Octa (5422) processor for different core-types, CPs and frequency setting (based on simulations in Gem5). The X- and Y-axis in all graphs represent frequency and normalized number of clock cycles, respectively.

maximum of 21.67%, 30.17% difference in execution cycles between frequency poles for A15 and A7 core-type, respectively. This significant increase is due to the fact that the reduced CP assignment increases the LLC miss-rate which in turn increases the memory-latency. A greater increase for the A7 cores is because the A7 cluster has an LLC of only 512 kB on the Exynos (5422) compared to the 2 MB LLC on the A15 cluster. The Out-of-Order (OoO) pipeline on the A15 core also contributes to hiding some of the memory-latency. It is also observed that the execution cycles of some tasks scale more than others. This is due to the diverse internal characteristics of the benchmark applications. Some benchmarks, e.g., *canneal*, are more memory-intensive resulting in a greater change in execution cycles, while others, e.g., *sha*, do not scale much due to their higher compute-intensity.

The execution time of a task is dictated by its execution cycles divided by the executing core-frequency. These execution cycles are estimated statically at normal core operation, i.e., without DVFS, in order to ensure an upper bound on the WCET [73,86]. Recent works consider constant execution cycles, and therefore, assume the execution time to increase linearly with core-frequency [94,105]. However, since in reality only the computation portion of the execution cycles scale with the core-frequency, such works fail to capture the actual change in execution cycles, thus, resulting in an over-estimation of the task execution time when DVFS is applied. Therefore, apart from under-utilizing the system, such a difference can have a drastic impact on energy-efficiency since a task may have to be placed on a

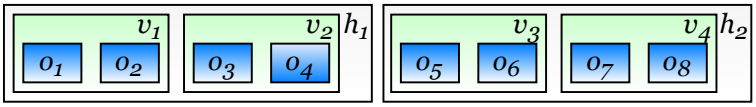


Figure 5-2: Clustered heterogeneous multicore system model with (M=8) cores, (K=4) clusters, and (J=2) core-types. The number of cores assigned to a particular cluster can be specified by the function $NUM_o(v_k)$ while the number of clusters assigned to a particular core-type can be specified by the function $NUM_v(h_j)$, e.g., $NUM_o(v_1) = 2$ and $NUM_v(h_1) = 2$.

power-hungry core even if there is some space available on the energy-efficient one. In the following sections, we aim to overcome this limitation by utilizing the improved task model, presented in Chapter 3, and by proposing a task-characteristic-aware mapping to efficiently utilize the cores for energy minimization.

5.1.1 System Model

The system builds upon the notion of a clustered heterogeneous multicore architecture composed of M cores, $O = \{o_1, o_2..o_M\}$ (indexed by m), where a core can be any one of J core-types, $H = \{h_1, h_2..h_J\}$ (indexed by j), such that same core-type cores are grouped to form clusters, resulting in a total of K clusters, $V = \{v_1, v_2..v_K\}$ (indexed by k). Figure 5-2 is an example of a clustered heterogeneous multicore system model with (M=8) cores, (K=4) clusters, and (J=2) core-types. Note that to signify the relation among components from different layers, we indicate a component's parent structure in the superscript of its expression, i.e., a core o_m belonging to core-type h_j can be expressed as $o_m^{h_j}$.

Each core-type cluster v_k can operate at an independent frequency while cores within the same cluster must operate at the same frequency. The frequency range for core-type h_j is defined by a finite set of frequencies, $F^{h_j} = \{f^1, f^2..f^{Q_j}\}$ (indexed by q_j) while the frequency assigned to a core o_m can be expressed as f_m. Furthermore, each cluster has an independent LLC that is shared by cores in the cluster. This model is adopted from the state-of-the-art clustered heterogeneous energy-efficient scheduling algorithms [94, 105].

To cater for the cache unpredictability, we assume the LLC can be partitioned into A partitions based on its associativity [122]. A core-based CP scheme is adopted where each core within a cluster is assigned a number of distinct CPs and the cumulative CPs assigned to the cores cannot exceed the associativity of the LLC. Adopting a task-based CP scheme for periodic tasks would require extensive analysis to ensure schedulability.

Initial investigations into this direction are discussed in the next chapter. The number of CPs assigned to each core is defined by $W = \{w_1, w_2..w_M\}$ (indexed by m). Note that the cores within a cluster are identical in every aspect except for the number of CPs assigned to each core making the CP assignment a distinguishing factor. Thus, the state of each core is identified by its heterogeneity, assigned cluster, operating frequency and assigned CPs. For instance, Exynos 5422 System on Chip (SoC) is an example of a clustered heterogeneous multicore SoC and can be modeled by the system parameters as M=8, J=2, K=2, F^{big}={0.2,0.3,...,2.0} GHz, F^{LITTLE}={0.2,0.3,...,1.4} GHz, and where 2048 kB and 512 kB LLCs for the big and LITTLE clusters, respectively are sectioned into A=8 CPs.

5.1.2 Task Model

The task model is defined by a set of N independent periodic tasks $T = \{\tau_1, \tau_2..\tau_N\}$ (indexed by n). Each task τ_n releases an infinite number of task-instances, each after a specified period p_n, where each task-instance must complete before a deadline d_n relative to its arrival time. We consider a partitioned scheduling scheme where tasks are statically allocated to cores and tasks within each core are scheduled via EDF.

The hyper-period of the complete taskset is defined by L. After the tasks are partitioned across all the cores, task-subsets are created for each core $\Gamma = \{T_1, T_2..T_M\}$ (indexed by m). Implicit task deadlines are considered ($p_n = d_n$), thus, the utilization of τ_n is defined by $u_n = \frac{e_n}{d_n}$, where e_n is the WCET of τ_n. According to EDF, the utilization of each core U_m must be less than or equal to 1 to ensure schedulability i.e. $U_m = \sum_{\tau_n \in T_m} \frac{e_n}{d_n} \leq 1$.

The same nonlinear task model proposed in Chapter 3 is utilized for this work, i.e., the execution cycles c_n of τ_n can be defined as:

$$c_n = cc_n + mc_n \tag{5.1}$$

To differentiate between these task-specific characteristics, we define a *compute-intensity* metric $\phi_n = \frac{c\hat{c}_n}{c\hat{c}_n + m\hat{c}_n}$, where $c\hat{c}_n$ and $m\hat{c}_n$ represents the compute-cycles and memory-cycles at maximum frequency and maximum CPs, respectively. We also define a *cache-gradient* metric, $\zeta_n^m = \frac{mc_n^{o_m}}{m\hat{c}_n}$ which represents the increase in memory-cycles of assigning τ_n to core o_m with w_m CPs, compared to a core of the same type having maximum CPs. These metrics will facilitate the task allocation strategy to determine the best core and core-type for each

task.

5.1.3 Power Model

For this work, we utilize the frequency dependent power model used in Chapter 3.

$$P_n = \kappa_n f_n^{\alpha_j} + \beta_j \tag{5.2}$$

where κ_n is the task activity factor, f_n is the core-frequency during the task's execution, and α_j and β_j are core-type specific constants [34, 100].

Due to their internal characteristics, different tasks vary in the amount of power they consume even when executing on the same core-type. This variation can be captured by the task-specific activity factors $\kappa_n \in \{0, 1\}$. Based on the derivations in Chapter 3, κ_n can be modeled as:

$$\kappa_n = \frac{\kappa_a cc_n + \kappa_s mc_n \delta_n}{cc_n + mc_n \delta_n} \tag{5.3}$$

where $\delta_n = \frac{f_n}{f_{mem}}$, f_{mem} is the constant frequency of the memory subsystem, while κ_a and κ_s are constants such that $\kappa_a > \kappa_s$.

5.1.4 Solution Space

Based on the system, task and power model, the execution time and power consumption of task τ_n becomes a function of its internal characteristics, architectural properties of the core-type h_j, executing core-frequency f_m, and CPs allocation, w_m, i.e., $e_n = F^e(\tau_n, h_j, f_m, w_m)$, $P_n = F^P(\tau_n, h_j, f_m, w_m)$, where $F^e(.)$ and $F^P(.)$ are functions mapping these variables to the execution time and power consumption of τ_n.

The energy consumption of a single instance of a task can be expressed as:

$$E_n = P_n \cdot e_n \tag{5.4}$$

Since we are using EDF scheduling, the cumulative utilization of tasks assigned to a core cannot exceed 1. When DVFS is applied, the cluster frequency must be set according to its highest utilization core in order to maintain schedulability. Furthermore, cores allocated with any tasks cannot be switched-off during periods of inactivity due to switching overheads

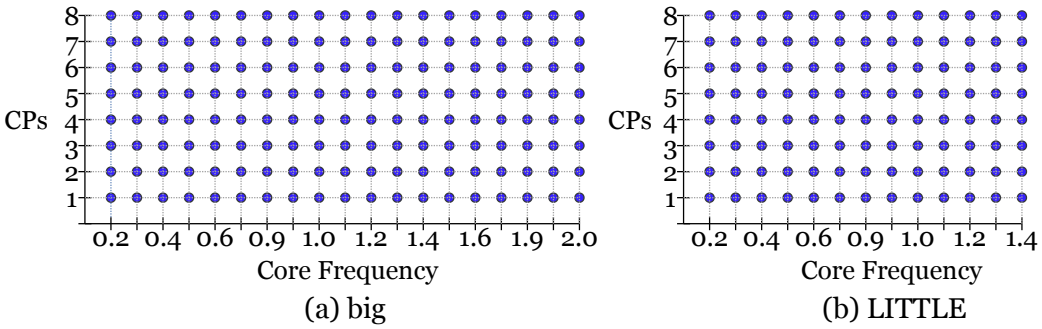


Figure 5-3: Solution space for a single task executing on a big or LITTLE core where each dot on the graph represents the energy consumption of the task for a particular core frequency and assigned CP.

and the prohibitive nature of EDF and real-time constraints [34, 105]. Therefore, the idle-power consumption of the cores must also be included into the power consumption of the system. This idle core power $P_{idle}^{h_j,f_m}$ can vary based on the core-type and core-frequency but is independent of the CP assignment.

Thus, the problem of finding the energy-optimal allocation and scheduling of a taskset can be summarized as follows:

$$\min_{h_j, f_m, w_m} \quad \sum_{T_m \in \Gamma} ((L \sum_{\tau_n \in T_m} P_n^{h_j,f_m,w_m} \frac{e_n}{d_n}^{h_j,f_m,w_m}) + (1 - U_m) P_{idle}^{h_j,f_m}) \tag{5.5a}$$

$$\text{s.t.} \quad \sum_{\tau_n \in T_m} \frac{e_n}{d_n} \leq 1 \qquad \forall\, T_m \in \Gamma \tag{5.5b}$$

$$f_m = f_k \qquad \forall o_m \in v_k \;:\; f_k \in F^j \quad \forall v_k \in h_j \tag{5.5c}$$

The objective function Eq. (5.5a) minimizes the energy consumed by the tasks and the energy lost due to idle-power, while constraints (5.5b, 5.5c) ensure taskset schedulability and that all cores in the same cluster operate at the same frequency.

Figure 5-3 displays the solution space for only 1 task on the ARM big.LITTLE Exynos 5422 SoC where the x and y axis represents the frequency and CP range on each core-type. The problem of finding the minimum energy-consumption is clearly of NP-complexity. Furthermore, a task-characteristic oblivious, aggressive allocation of tasks to a power-efficient core does not guarantee energy minimization since some tasks may be more energy-efficient when executing on a high-performance core operating at a lower frequency as opposed to running on a power-efficient core at a higher frequency [105]. To design an effective solution, it is important to analyze the dependency of each task on different core-types,

frequencies and CP sizes. We propose such a solution in the following sections. We first make assumptions that aim to build upon recently proposed analysis and then based on these assumptions, we provide proofs to further corroborate our proposed solution.

5.2 Task Assignment Strategy

The previous section identified a 3D partitioning problem for energy-minimization on a heterogeneous clustered multicore system. In this section, we solve this problem by determining the criteria for selecting the favorable core-type cluster, core within a core-type cluster, and frequency for each cluster based on the values of cc_n and mc_n of each task. We analyze each dimension separately by keeping the other two constant. This simplifies the analysis allowing us to identify the specific properties of each dimension and how it affects the power consumption and execution length of each task.

5.2.1 Core-type Selection

The performance factor of core-types is differentiated by the speed at which they execute tasks. Hence, the general notion of a *high-performance* core-type is expected to complete the execution of a task earlier than a *low-performance* core-type. This performance difference is evident from the reduction in the execution cycles of a task running on a high-performance core, i.e., specifically, the reduction in computation cycles cc_n [72], e.g. authors in [34] showed the speedup achieved by MiBench applications running on the ARM A15 (big) core compared to the A7 (LITTLE) core.

Therefore, we start by first defining a metric for the performance of core-types, i.e., we define *compute-cycle-scale*: $S_{h_j} \leq 1$, as a factor that scales the number of compute-cycles cc_n of a task when running on core-type h_j compared to reference core-type h_r. Assuming the same constant frequency between core-types, the resultant length of execution cycles of τ_n running on h_j are $cc_n S_n^{h_j} + mc_n$. This metric can be related to in-order A7 and OoO A15 cores on the big.LITTLE where $S_{A7} > S_{A15}$ and h_{A7} is the reference core resulting in a number of execution cycles on the A15 as $cc_n^{A7} S_n^{A15} + mc_n^{A7}$. Incidentally, if $S_n^{h_1} > S_n^{h_2}$ for all $\tau_n \in \Gamma$, then it is safe to assume that $P_n^{h_1}$ will be less than $P_n^{h_2}$ for all $\tau_n \in \Gamma$. This is a valid assumption based on the power-model parameters measured in [34].

Consequently, *energy-factor*: $S_n^{h_j} \frac{P_n^{h_j}}{f_n}$, represents a measure of change in the energy con-

sumption when assigning τ_n to a core of type h_j compared to a reference core-type h_r. Thus, if $S_n^{h_1}\frac{P_n^{h_1}}{f_n} < S_n^{h_2}\frac{P_n^{h_2}}{f_n}$ for τ_n then h_1 is considered the energy-efficient option for τ_n. Note that in a practical scenario, memory cycles may also reduce when switching to a *higher performance* core due to memory latency hiding induced by OoO cores, however, the impact is relatively less significant. We also assume the function S_{h_j} scales the number of compute-cycles of every task by the same factor. Again, this is a generalization of the core-type properties since the degree of reduction in compute-cycles when switched to a higher performance core-type depends on the internal characteristics of the task e.g. a task's tendency to exploit the instruction level parallelism available in OoO cores. However, abstracting away from micro-architectural features allows us to simplify the analysis leading to effective decision making.

Lemma 3. *Consider a system of two cores of different-types $o_1 \in h_1$ and $o_2 \in h_2$ having the same frequency and CP size where $S_{h_1} > S_{h_2}$ and $S_n^{h_1}\frac{P_n^{h_1}}{f_n} < S_n^{h_2}\frac{P_n^{h_2}}{f_n}\ \forall\ \tau_n$. Given two tasks τ_1 and τ_2 where $e_1 = e_2$ on the reference core-type h_1 and $\phi_1 < \phi_2$, assigning τ_1 to o_1 and τ_2 to o_2 will result in less energy consumption compared to the opposite assignment.*

Proof. The difference in energy consumption of both assignment strategies can be expressed as $E_\Delta = E' - E''$, where E' is the energy consumption of assigning τ_1 to o_1 and τ_2 to o_2:

$$E' = (cc_1 S_1^{h_1} + mc_1)\frac{P_1^{h_1}}{f_1} + (cc_2 S_2^{h_2} + mc_2)\frac{P_2^{h_2}}{f_2}$$

The opposite assignment will result in an energy consumption:

$$E'' = (cc_1 S_1^{h_2} + mc_1)\frac{P_1^{h_2}}{f_1} + (cc_2 S_2^{h_1} + mc_2)\frac{P_2^{h_1}}{f_2}$$

Since $cc_2 > cc_1 (\phi_2 > \phi_1)$, assigning τ_2 to h_2 will downscale cc_2 by a greater factor compared to assigning τ_2 to h_1. Therefore, the initial assumption that $e_1 = e_2$ will cause E' to be less than E''. Thus, the lemma is proven. □

A similar hypothesis has been proven in [94] for a linear task-model where tasks consume the same power when executing on the same core-type. Lemma 3 extends that hypothesis to incorporate the nonlinear task model adopted in this work. Doing so enables us to design an optimal core-type assignment strategy under assumptions made above.

Theorem 2. *Consider a non-DVFS enabled system of m cores and j core-types with same number of CP allocations where $j = m$ and $S_n^{h_1} > S_n^{h_2} > .. > S_n^{h_j}$ and $S_n^{h_1}\frac{P_n^{h_1}}{f_n} < S_n^{h_2}\frac{P_n^{h_2}}{f_n} < .. <$*

$S_n^{h_j} \frac{P_n^{h_j}}{f_n}$ *for all* $\tau_n \in \Gamma$. *Given a set of tasks of equal length* $e_1 = e_2 = .. = e_n$, *re-ordering the tasks in non-decreasing order of their compute-intensities,* ϕ_n, *and iteratively assigning tasks to cores in first-fit manner, starting with core-type* h_1 *will minimize the energy consumed by the taskset.*

Proof. Given that $S_n^{h_1} \frac{P_n^{h_1}}{f_n} < S_n^{h_2} \frac{P_n^{h_2}}{f_n}$ for all $\tau_n \in \Gamma$, assigning a task τ_n to h_1 rather than h_2 will maintain a lower energy consumption of τ_n. The rest of the proof follows from Lemma 3 where tasks with larger ϕ_n are more affinitive to core-types with smaller S_{h_j} factors. □

Theorem 2 establishes the following criterion:

Criterion 1: Tasks with lower compute-intensity ϕ_n should be assigned to cores-types with smaller energy-factors $S_n^{h_j} \frac{P_n^{h_j}}{f_n}$.

5.2.2 Core Selection (based on CPs)

The LLC partitions assigned to a core have a considerable impact on the execution time of a task. Fewer CPs can increase the LLC miss-rate which, in turn, increases the memory-latency cycles mc_n. It is observed that the execution time increases monotonically as the number of CPs are decreased [26, 120]. Assuming such a relationship prevails for all tasks on every core-type, the following lemma can be justified.

Lemma 4. *Consider a system of two cores at maximum frequency and of the same type* $o_1, o_2 \in h_j$ *where the CPs assigned to* o_1 *is greater than those assigned to* o_2, *i.e.,* $w_1 > w_2$. *Given two tasks* τ_1 *and* τ_2 *where* $e_1 = e_2$ *on* h_j *with 8 CPs,* $\phi_1 < \phi_2$, *and both tasks have the same monotonic execution time to CP relationship, i.e., cache-gradients* $\zeta_1^1 = \zeta_2^1 \& \zeta_1^2 = \zeta_2^2$, *then assigning* τ_1 *to* o_1 *and* τ_2 *to* o_2 *will result in less energy consumption compared to its opposite assignment.*

Proof. From Eq. (5.1) and the *cache-gradient* definition, we can infer the change in execution cycles of assigning τ_1 to o_1 as $cc_1 + mc_1\zeta_1^1$. Thus, based on Eqs. (5.2), (5.3) and (5.4), the difference in dynamic energy consumption of both assignment strategies is:

$$E_{\Delta}^{dyn} = ((cc_1\kappa_a + mc_1\zeta_1^1\kappa_s) + (cc_2\kappa_a + mc_2\zeta_2^2\kappa_s) \\ -(cc_1\kappa_a + mc_1\zeta_1^2\kappa_s) - (cc_2\kappa_a + mc_2\zeta_2^1\kappa_s))f^{\alpha-1}$$

where δ_n in Eq. (5.3) equals 1 due to $f_n = f_{max}$. $w_1 > w_2$ implies $\zeta_1^1 = \zeta_2^1 < \zeta_1^2 = \zeta_2^2$. Thus, since the execution cycles of both tasks increase by the same factor due to a smaller CP assignment and since before the assignment $mc_1 > mc_2$ ($\phi_1 < \phi_2$), increasing mc_1 by a smaller factor ζ_1^1 will result in an effective execution time smaller than compared to the opposite assignment, resulting in $E_\Delta^{dyn} < 0$. The same case follows for the static component,

$$E_\Delta^{sta} = ((cc_1 + mc_1\zeta_1^1) + (cc_2 + mc_2\zeta_2^2) - (cc_1 + mc_1\zeta_1^2) - (cc_2 + mc_2\zeta_2^1))\frac{\beta}{f^\alpha}$$

Thus, the cumulative increase in energy consumption of both tasks brought about by the increase in mc_1 and mc_2 by assigning τ_1 to o_1 and τ_2 to o_2 is less than the opposite assignment. □

Lemma 4 is extended to derive an energy optimal assignment strategy between same core-types with different CPs.

Theorem 3. *Consider a non-DVFS enabled system of m cores of the same-type h_j, such that the cores are assigned CPs in non-increasing order $w_1 > w_2 > .. > w_m$. Given a set of tasks of equal length $e_1 = e_2 = .. = e_n$ with the same monotonic execution time to CP relationship $\zeta_1^m = \zeta_2^m = .. = \zeta_n^m \forall o_m \in h_j$, re-ordering the tasks in non-decreasing order of their ϕ_n values and iteratively assigning the tasks to the cores with the largest CP will minimize the energy consumed by the taskset.*

Proof. Based on Lemma 4, and under the assumption made on the execution time and CP relationship, tasks with comparatively smaller ϕ_n values benefit in terms of energy minimization when assigned to cores with greater CPs. Thus, when frequencies are constant and tasks are of equal length, ordering tasks in non-decreasing order of their ϕ_n values promotes a task allocation that results in a energy optimal schedule. □

Theorem 3 provides a criterion for determining the appropriate CPs allocated for each task.

Criterion 2: Tasks with lower compute-intensity, ϕ_n, should be assigned to cores with more CPs.

Note that Theorem 3 only holds true for a setup where all tasks must have the same execution time to CP relationship. Nonetheless, the simplistic assumption still governs a criteria for determining the appropriate CP allocation for each task. For tasks that do not

follow the same execution time to CP relationship, it has already been shown in Chapter 3 and 4 that *cache-friendly* tasks, i.e., tasks with a comparatively higher cache-gradients, should be assigned to cores with greater number of CPs.

Criterion 3: Tasks with a higher ζ_n^m should be assigned to the cores with greater number of CPs.

5.2.3 Core Selection (based on frequency)

For DVFS enabled multicore systems, the optimal frequencies that will minimize energy consumption depend on the task allocation strategy. Theorem 4 summarizes this for the model presented in this chapter.

Lemma 5. *Consider two tasks τ_1 and τ_2, where $e_1 = e_2$ and $\phi_1 < \phi_2$, assigned to a core o_m running at frequency f_m. Reducing the frequency of the core will result in $e_1 < e_2$.*

Proof. The difference in execution length brought about on a task due to a decrease in frequency can be expressed as:

$$e_\Delta = \frac{cc_n}{f_m} - \frac{cc_n}{f_{max}}$$

Since $cc_1 < cc_2$, the increase in execution time from cc_2 cycles will be greater than the increase in execution time from cc_1 cycles. □

Lemma 5 is extended to derive optimal strategy to minimize the frequency of a global-DVFS-enabled multicore cluster.

Theorem 4. *Consider a DVFS-enabled system with m cores of the same type where all cores must run at the same frequency. Given a set of tasks of equal length $e_1 = e_2 .. = e_n$, re-ordering tasks in non-decreasing order of their ϕ_n values and assigning the tasks to the cores in the worst-fit manner will permit maximum frequency scaling.*

Proof. This comes directly from Lemma 5 where DVFS causes a task with a higher ϕ_n value to change in length more than that of a lower ϕ_n. Since the initial execution lengths of the tasks are equal, balancing tasks across the cores based on their ϕ_n values, via WFD, will minimize the difference in execution lengths among the cores once DVFS is applied. This comes directly from Lemma 5 where DVFS causes a task with a higher ϕ_n value to change in length more than that of a lower ϕ_n. Since the initial execution lengths of the tasks are

equal, balancing tasks across the cores based on their ϕ_n values, via WFD, will minimize the difference in execution-lengths among the cores once DVFS is applied. □

Criterion 4: Balancing the tasks across the cores based on their ϕ_n values will permit a minimum DVFS setting.

The analysis performed in this section agrees towards the understanding that ϕ_n plays a significant role in the energy minimization of the taskset. Note that the correctness of the proofs presented above depends on the perverse assumption of tasks with equal execution lengths. Nevertheless, it still provides a basis for task assignments between core-types, cores and CPs where tasks are not necessarily of equal lengths as will be shown in Section 5.4. In the next section, we will make use of these observations to create an effective heuristic solution for this 3D energy-minimization problem.

5.3 The Proposed Algorithm – THEAM

The presented algorithm, Task-Heterogeneity-Energy Aware Mapping (THEAM), requires knowledge of the monotonicity of core-types in the system, i.e., ordering of the core-types based on their potential energy consumptions [92]. This can be easily established based on the architectural composition of the core-types, e.g., in-order vs OoO. Furthermore, this does not limit the application space since heterogeneous platforms are built on the grounds of providing diversity in the execution environment. For this reason, two core-type heterogeneous systems are the prevalent platform as they already present most of the power and performance benefits of heterogeneity [64]. Nonetheless, the presented algorithm caters to platforms with more than two types of heterogeneity as shown in Figure 5-4. The monotonicity can also be determined by sorting core-types according to the average energy consumed by the tasks on each core-type, i.e. average energy-factors [94].

The algorithm builds upon observations made in Section 5.2, where core and core-type affinities were discovered based on task characteristics. The algorithm also takes the idle-power into account by balancing the tradeoff between maximizing DVFS and minimizing the number of active-cores. Such a balance is achieved by dividing the algorithm into two phases. The first phase determines the minimum number of active-cores that would be required to schedule the taskset. After allocating tasks across the specified cores, the second phase iteratively increases the active-core count and re-balances the workload to

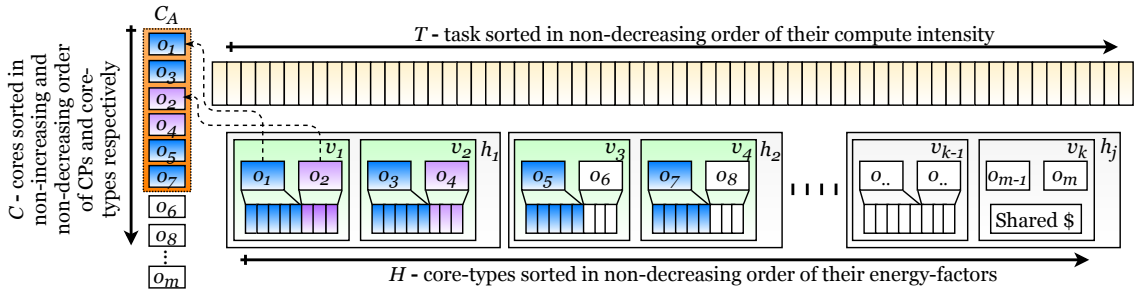


Figure 5-4: THEAM algorithm structure. The core-types $h_j \in H$ are arranged according to their energy-factors. The tasks $\tau_n \in T$ are arranged according to their *compute-intensity*. CPs are statically assigned to each core in a cluster $v_k \in V$. All cores are separately arranged in C according to their CP assignment and core-type to facilitate the task allocation strategy according to the Criteria established in Section 5.2. Cores without any tasks are shut-off along with their associated CPs while active cores are collected into an array C_A.

further reduce frequency. This continues until the energy-consumption fails to decrease any further.

Figure 5-4 represents the algorithm structure. Cores in the same cluster operate at a common frequency while the LLC is partitioned among cores. Since low ϕ_n tasks have an affinity towards energy-efficient core-types (Criterion 1), both tasks and core-types are sorted in non-decreasing order of their ϕ_n and energy-factor values, respectively. Furthermore, since low ϕ_n tasks benefit from cores with comparatively greater CPs (Criterion 2), an uneven distribution of CPs among cores can provide additional energy savings. Thus, cores within each cluster are arranged in non-increasing order of their assigned CPs to prompt tasks with lower ϕ_n values to be allocated to cores with greater CPs. This arrangement facilitates a separate Two-Dimensional (2D) sorting of all the cores in the system in non-increasing order of their assigned CPs and non-decreasing order core-type energy-factors, i.e., cores for a specific core-type are arranged in non-increasing order of their assigned CPs before moving on to the next core-type. This is shown as C in Figure 5-4 where o_1 and o_3 are given sorting preference over o_2 and o_4, and all the cores of h_1 are given sorting preference over cores in h_2. This 2D sorting of cores balances benefits of both Criteria 1 and 2.

5.3.1 Phase-1: Minimizing Active Cores

In the first phase, i.e., *Phase-1*, tasks are allocated on a minimum number of active cores to reduce the idle-power consumption. Algorithm 5 presents the pseudo-code for *Phase-*

ALGORITHM 5: Phase-1

```
1: function PHASE1(sys = {T, H, V, C, A})
2: T ← sort tasks in non-decreasing order of φ_n values.
3: C ← sort cores in non-increasing order of CPs and non-decreasing order of core-type
   energy-factors.
4: C_A ← {}.
5: for τ_n ∈ T do
6:    allocated← 0
7:    for o_m ∈ C_A do
8:       if U_m + u_n <= 1 then
9:          allocated← 1
10:         break
11:      end if
12:   end for
13:   if allocated = 0 then
14:      o_m←argmax_{o_m}(w_m)∀o_m∈h_j, ∉C_A ∧ h_j=min(H)
15:      C_A ← {C_A, o_m}
16:   end if
17: end for
18: for τ_n ∈ Γ do
19:    Assign τ_n to C_A in worst-fit manner according to φ_n;
20: end for
21: MINIMIZEFREQUENCY(h_j) ∀h_j ∈ H
22: endfunction
```

1. Core-types, cores and tasks are first sorted according to the arrangement presented in Figure 5-4. The minimum number of active-cores required to schedule the taskset is determined by making mock task assignments to determine the total utilization of the allocated taskset (lines: 5-17). Cores are added into the active-core set C_A based on the sorted arrangement of C (line: 15). Once the minimum core-count is established, tasks are allocated onto active-cores C_A in a worst-fit manner according to their ϕ_n values (line: 18-20). The assignment process allocates tasks in a core-type step-wise manner, i.e., tasks are first worst-fit allocated (Criterion 4) across the subset of cores in C_A belonging to core-type h_1 before moving onto the next core-type subset. At each step, the utilization of the task is updated according to its core-type and CPs allocation. MINIMIZEFREQUENCY then minimizes the frequency of all core-type clusters while ensuring schedulability (line: 21). The frequency is reduced on a core-type basis, thus, equalizing the frequency of clusters of the same core-type using Algorithm 6.

5.3.2 Phase-2: Maximizing DVFS

In second phase, i.e., *Phase-2*, the number of active-cores are iteratively increased in

ALGORITHM 6: MinimizeFrequency

```
1: function MINIMIZEFREQUENCY(h_j)
2: while f(h_j) ≥ f_critical^{h_j} do
3:    Decrement f(h_j);
4:    if ∃U_m > 1 ∀o_m ∈ h_j then
5:       Increment f(h_j);
6:       break;
7:    end if
8: end while
9: endfunction
```

hopes that a lower DVFS setting will further reduce the energy consumption. Algorithm 7 presents the pseudo-code for *Phase-2* which follows directly after *Phase-1*. Core-types are first sorted in non-decreasing order of their energy-factors (line: 2). After keeping a log on the current system configuration and energy consumption (line: 4), a task allocated core-type which is not already at minimum frequency is selected from the sorted set H (lines: 5-9). Note that the loop returns the core-type with the highest energy-factor under the selection criteria. This is done based on the intuition that prioritizing the frequency minimization of higher energy-factor core-types will lead to lower energy consumption [105]. Failure to find any core-type indicates that all cores are already at a minimum frequency setting, thus, prompting the algorithm to exit the loop. If a core-type h_s is found, the frequency of all clusters associated with the core-type are decremented (line: 11). Decreasing the frequency, increases the cycle utilization of the tasks. Therefore, each core's utilization is monitored at each iteration to ensure the taskset utilization does not exceed 1 (line: 12). If a core o_m in h_s becomes unschedulable, the task allocation is re-adjusted in hopes of balancing the utilization. The re-adjustment is based on Criterion 3 where tasks with higher ζ_n are assigned to cores with greater CPs using Algorithm 8.

Function ADJUSTUTILIZATION iteratively transfers selected tasks from the heaviest utilization core o_h to the lightest utilization core o_l. A task exchange after the loop exits ensures a better balance of the taskset utilization (lines: 12-14 (Algorithm 8)), after which the frequency of h_s is then minimized to enable maximum energy savings.

If the unschedulability persists (line: 15 (Algorithm 7)) and if an empty core of the same core-type is available (line: 16), the active-core count is increased by activating the empty core of the same type based on the ordering of C in Figure 5-4 (lines: 17-18). The task allocation of h_s is re-adjusted again to accommodate the new core in balancing the

ALGORITHM 7: Phase-2

function PHASE2($Sys = \{T, H, V, C, A\}, C_A$)
$H \leftarrow$ sort h_j in non-decreasing order of their energy-factors;
do
$E_{old} \leftarrow E_{sys}$, $sys_{old} \leftarrow sys$, $s \leftarrow -1$;
for $h_j \in H$ **do**
  if$(\exists o_m : o_m \in C_A \forall o_m \in h_j) \& (f(h_j) > f^{h_j}_{critical})$ **then**
    $s \leftarrow j$
  end if
end for
if $s! = -1$ **then**
  Decrement $f(h_s)$;
  if $\exists U_m > 1 \ \forall o_m \in h_s$ **then**
    ADJUSTUTILIZATION(h_s);
  end if
  if $\exists U_m > 1 \ \forall o_m \in h_s$ **then**
    if $\exists o_m : o_m \notin C_A, \in h_s$ **then**
      $o_m \leftarrow argmax_{o_m}(w_m) \ \forall o_m \notin C_A, \in h_s$
      $C_A = \{C_A, o_m\}$;
      ADJUSTUTILIZATION(h_s);
    else if $s! = J$ **then**
      if $\nexists h_{s+1} \in C_A$ **then**
        $o_m \leftarrow argmax_{o_m}(w_m) \forall o_m \in h_{s+1}, \notin C_A$
        $C_A = \{C_A, o_m\}$;
      end if
      $o_l \leftarrow argmin_{o_m}(U_m) \ \forall o_m \in h_{s+1}, \in C_A$;
      while $\exists U_m > 1 \ \forall o_m \in h_s$ **do**
        $o_h \leftarrow argmax_{o_m}(U_m) \ \forall o_m \in h_s, \in C_A$;
        $\tau_s \leftarrow argmax(\phi_n) \ \ \forall \tau_n \in T_h$
        Transfer τ_s from o_h to o_l
        if $U_h < 1$ **then**
          Transfer τ_s from o_l to o_h
          $\tau_s \leftarrow argmin(u_n) \forall \tau_n \in T_h : u_n > U_h - 1$
          Transfer τ_s from o_h to o_l
        end if
      end while
      while $\exists U_m > 1 \ \forall o_m \in h_{s+1}$ **do**
        Increment $f(h_s + 1)$;
      end while
      if $\exists U_m > 1 \forall o_m \in h_{s+1}$ **then**
        $o_m \leftarrow argmax_{o_m}(w_m) \forall o_m \in h_{s+1}, \notin C_A$
        $C_A = \{C_A, o_m\}$;
      end if
      ADJUSTUTILIZATION(h_{s+1});
    end if
  end if
  if $(E_{old} < E_{sys}) \ || \ (\exists U_m > 1 \ \forall o_m \in C_A)$ **then**
    $sys \leftarrow sys_{old}$;
    break;
  end if
end if
while$s! = -1$
endfunction

ALGORITHM 8: AdjustUtilization

1: **function** ADJUSTUTILIZATION(h_s)
2: **do**
3: $o_l \leftarrow argmin_{o_m}(U_m)\ \forall o_m \in h_s, \in C_A$;
4: $o_h \leftarrow argmax_{o_m}(U_m)\ \forall o_m \in h_s, \in C_A$;
5: **if** $w_l < w_h$ **then**
6: $\tau_s \leftarrow argmin_{\tau_n}(\zeta_n^l)\ \ \forall \tau_n \in T_h$
7: **else**
8: $\tau_s \leftarrow argmax_{\tau_n}(\zeta_n^l)\ \ \forall \tau_n \in T_h$
9: **end if**
10: Transfer τ_s from o_h to o_l
11: **while** $o_l != argmax_{o_m}(U_m)\ \forall o_m \in h_s, \in C_A$
12: Transfer τ_s from o_l to o_h
13: $\tau_s \leftarrow argmin_{\tau_n}(u_n)\ \ \forall \tau_n \in T_h : u_n > U_h - U_l$
14: Transfer τ_s from o_h to o_l
15: MINIMIZEFREQUENCY(h_s);
endfunction

utilization of the taskset (lines: 19).

If there are no empty cores of the same type and if there is a higher energy-factor core-type available h_{s+1} (line: 20), selected tasks are transferred from h_s to h_{s+1}. The transfer is based on Criterion 1 where tasks with higher ϕ_n values have more affinity to higher energy-factor core-types compared to tasks with lower ϕ_n values. If a core from h_{s+1} is not yet in C_A, it is added to the list (lines: 21-24). The lightest utilization core o_l of h_{s+1} is then selected and tasks are iteratively transferred from the heaviest cores of h_s until h_s is schedulable again (lines: 25-35). Similar to ADJUSTUTILIZATION, a task exchange at the end, (lines: 31-33), ensures a better balance of the taskset utilization. If o_l becomes unschedulable, the frequency of h_{s+1} is iteratively increased (lines: 36-38). If the unschedulablity persists, then the active-core count is increased by accommodating a core of the same type, after which the taskset is then re-balanced and the frequency is minimized (lines: 39-43). If no empty cores are available ($C \subset C_A$), or the unschedulability still persists, or if the new configuration results in more energy consumption, the system reverts to its previous configuration and exits (lines: 46-49).

5.4 Experimental Results

In this section, we present the experimental evaluations conducted to ascertain the expected performance of the proposed approach. The results are based on simulations using power-model parameters for big.LITTLE Exynos 5422 platform, similar to works presented

in [81, 94, 108]. For this work, the power-model parameter values are taken from [34]. Realistic applications from the MiBench [52] and PARSEC [19] benchmark suite are used to model the taskset. Benchmark characteristics are determined via the Gem5 Simulator. The experimental results of the proposed approach are shown in terms of percentage energy savings achieved against the state-of-the-art clustered heterogeneous energy-efficient scheduling algorithms, *HIT-LTF* [94] and *TCHAP* [105]. The approaches proposed by these state-of-the-art algorithms have already been described in detail in Section 2.2.1.

5.4.1 Setup

Platform Configuration

We select a commonly used platform, the heterogeneous big.LITTLE Exynos-5422 processor composed of ARM A7 and A15 core-type clusters, for the experimental evaluation [34, 94, 105]. Each cluster is composed of 4 cores with a 512 kB and 2 MB LLC attached to each A7 and A15 core-type cluster, respectively. The platform has a wide range of frequency settings, i.e., A7 clusters have a frequency range of {0.2, 0.3, ..., 1.4} GHz, while the A15 clusters have a frequency range of {0.2, 0.3, ..., 2.0} GHz [105].

Authors in [34] performed experiments on the Exynos-5422 processor to determine power-model parameters specific to the platform, i.e., $\kappa_a = \{1.35 \times 10^{-5}, 3.42 \times 10^{-7}\}$ mW/MHz^3, $\alpha = \{2.27, 2.88\}$ and $\beta = \{18.01, 135.07\}\, mW$ for the A7 and A15 core-type, respectively. We utilize these parameters to model power consumption of tasks running on the cores. Based on the trend observed from the power-model parameters derived in [126], we assume κ_s as half of κ_a. The idle-power values on each core-type at every frequency are taken from the experimentally measured values in [37, 38]. Furthermore, we assume the LLCs can be partitioned into 8 sections. For experimentation, the number of clusters and cores are varied to diversify the configuration setting of the platform.

Workload

Benchmark applications from the PARSEC (*canneal, blackscholes, ferret, fluidanimate and swaptions*), and MiBench (*qsort, susan, stringsearch, ispell and sha*), benchmark suite are used to model each task within the taskset. These benchmarks were selected to represent an adequate mix of both compute- and memory-intensive workloads. Benchmark execution

cycles and characteristics are determined via the Gem5 Simulator, i.e., cc_n and mc_n of each benchmark application are extracted for every core-type, CP size and frequency setting.

To generate a taskset, a benchmark application is randomly selected to model a task. The total cycle-count of the selected application is used to define the execution time of the task. The utilization of the task is determined by selecting one of three distribution, [0.01,0.1], [0.1,0.4] and [0.4,0.6], where each distribution has probabilities 3/10, 4/10 and 3/10, respectively [120]. The task utilization is fine-tuned before adding it to the set in order to keep the taskset hyper-period within bounds. The establish a conclusive comparison with existing algorithms, the evaluations are carried out over a wide range of values for taskset utilization [105].

5.4.2 Results

Different Configurations

Figure 5-5 displays the experimental results for various configuration settings on the heterogeneous platform. The results are shown as percentage energy savings achieved by our proposed algorithm, *THEAM*, against HIT-LTF and a best-fit variant of TCHAP, for a wide range of taskset utilization. The Y-axis represents the taskset utilization without DVFS after it has been mapped onto the cores via THEAM. For each configuration, the utilization is increased by 0.1 until the limit is reached. Each utilization step shows the energy savings averaged over 300 different taskset simulations.

For all configurations, THEAM performs better than its competitors. Figure 5-5 (a) shows the energy savings for a configuration where there is 1 cluster for each core-type and the 4 cores within each cluster equally share the CPs. Percentage energy savings achieved over HIT-LTF are better than TCHAP, showing an average of 15.73%, 13.0% and a maximum of 35.0%, 31.7% against HIT-LTF and TCHAP, respectively. This is because HIT-LTF does not consider the idle-power consumption and tries to balance tasks across all energy-efficient cores. TCHAP, on the other hand, tries to minimize the number of active cores to reduce idle-power consumption. This is apparent in Figure 5-6 (a) which shows the average number of active cores for each core-type and algorithm as the taskset utilization is increased. At low-workload conditions, HIT-LTF tries to maximize its active core count for the LITTLE cluster (HIT-LTF-L) while both TCHAP and THEAM try to minimize their

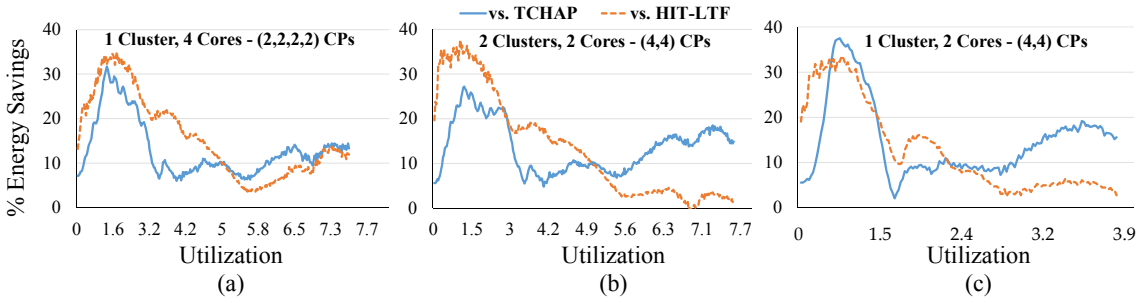


Figure 5-5: Percentage energy savings of THEAM vs. TCHAP and HIT-LTF for different cluster, core and taskset utilization values.

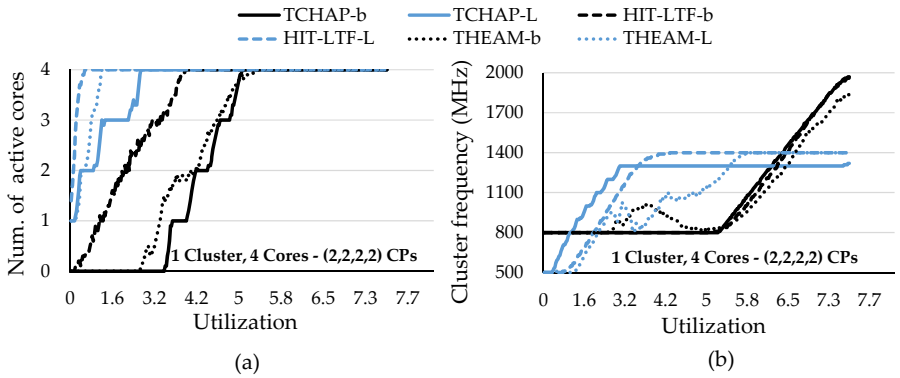


Figure 5-6: Comparison among THEAM, TCHAP and HIT-LTF (a) on active-core count and (b) on the operating frequency as the taskset utilization is increased

LITTLE core count (TCHAP-L, THEAM-L). The same trend follows for the big active-core count (HIT-LTF-b, TCHAP-b, THEAM-b) as well. The number of active cores for TCHAP increases at a slower rate compared to THEAM. This is because TCHAP tries to effectively minimize active cores via best-fit heuristic, while THEAM establishes a balance between energy lost due to idle-power and energy consumed by the tasks. Furthermore, mapping tasks across active-cores via WFD heuristic permits a lower DVFS setting. This is apparent in Figure 5-6 (b) which shows the frequency for each core-type and algorithm as the taskset utilization is increased. This along with the nonlinear cycle-count considerations enables THEAM to achieve greater energy savings compared to others.

In Figure 5-5, at a low utilization (U<3.2), the energy savings achieved against HIT-LTF and TCHAP are significant. As the utilization is increased from 0.1, the energy savings tend to increase and are maximum when the active-core count equals the little-core count for THEAM. As the utilization is increased further, the energy savings tend to decrease since the higher utilization reduces the DVFS capabilities of the cores. At medium utilization

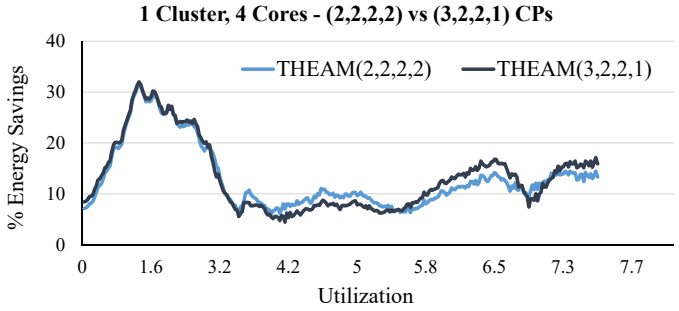


Figure 5-7: Gain in energy savings of using a selective CP setting against an equal CP setting

($3.2<U<5$), TCHAP performs better than HIT-LTF as TCHAP maintains a lower active-core count. At high utilization ($U>5$), all algorithms maintain a maximum core-count. In this case, HIT-LTF performs better than TCHAP since the worst-fit allocation adopted in HIT-LTF enables a lower DVFS setting. THEAM maintains its performance at higher utilization, representing its effectiveness in mapping the tasks across the cores to minimize energy consumption. The trend is consistent across all different configurations.

Selective Cache Partitions

Figure 5-7 affirms the observation made in Section 5.2 of assigning low ϕ_n tasks to cores with greater CPs. In this experiment, CPs are assigned to cores according to Figure 5-4 where tasks with lower ϕ_n are assigned to cores with greater CPs. Figure 5-7 displays the percentage energy savings against TCHAP achieved for a selective CP setting where cores have CPs assigned in decreasing order, against an equal CP setting where all cores have the same number of CPs. Results show a maximum of 3.3% and an average 0.23% gain in energy savings for using the selective approach. The fluctuation and lower gains in the energy savings are due to unequal execution lengths of the tasks and the fact that not all tasks have the same execution time monotonic relationship with CPs. However, benefits in energy savings are still apparent.

System-level Energy Minimization

This section presents the energy savings achieved when the energy consumption of both cores and LLCs are considered. The cache energy is computed using the cache energy model used presented in Chapter 3. The number of cache-accesses and cache-misses are

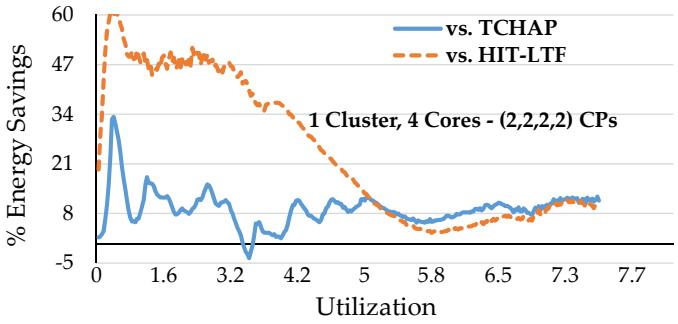


Figure 5-8: System-level % energy savings of THEAM vs. TCHAP and HIT-LTF

determined via Gem5, while the cache energy parameters are computed using the HP-CACTI power estimation tool at a 32 nm technology setting [109]. The cache is assumed to consume constant static power when no tasks are running on the active cores and CPs assigned to inactive cores are switched-off using the selective-way approach [122]. The proposed algorithm THEAM is modified to consider both core and cache energy when deciding to revert back to the previous task assignment / DVFS setting (Algorithm 7, line:46). Figure 5-8 show an average 24.28%, 9.25% and a maximum 61.1%, 33.3% energy savings against HIT-LTF and TCHAP, respectively. The significant gains against HIT-LTF is due to the difference in active-core count and active-CPs.

5.5 Discussion

This chapter presented a holistic approach to minimize energy consumption on clustered heterogeneous multicore systems while compensating for the simplistic assumptions made in prior work. Factors affecting the independent memory latency cycles due to DVFS and cache-partitioning techniques were thoroughly analyzed in order to design a heuristic scheduling algorithm THEAM. Results show a maximum and average energy savings of 35.0% and 15.73% respectively for core-level energy consumption, and 61.1% and 24.28% respectively for system-level energy consumption.

Chapter 6

Dynamic Cache-Partitioned Schedulability Analysis for Periodic Tasks

Recent integration of the cache partitioning model has ushered in new paradigms of research in the predictability and schedulability analysis for real-time systems. This has resulted in two general cache-partitioning techniques, i.e., dynamic CP and static CP schemes, as described in Chapter 2. However, simplicity in the analysis framework has prompted existing contributions to be biased towards a static CP scheme. The dynamic CP scheme has largely been untackled despite its proficiency in schedulability, flexibility, and energy-efficiency. In Chapter 4, we attempted to address this problem for the simpler frame-based taskset. Since partitioned fixed-priority scheduling has become the de facto standard in the automotive multicore real-time systems domain, e.g., mandated by AUTOSAR standard for automotive systems [117], increasing its schedulability in the CP scenario is imperative and can be achieved by adopting a dynamic CP scheme.

In this chapter, we make initial contributions to a dynamic CP schedulability analysis for preemptive scheduling of fixed-priority periodic tasks. We devise a sufficient schedulability test and then propose to refine the upper bound by adopting techniques to reduce the interference caused by cache contention. The schedulability test can be used for future dynamic CP energy-efficient scheduling algorithms by ensuring all tasks meet their deadlines.

6.1 System Model and Problem Setting

We consider a set of N independent sporadic tasks represented as $\Gamma = \{\tau_1, \tau_2, .., \tau_N\}$ partitioned across a multicore system with M identical cores $O = \{o_1, o_2, .., o_M\}$. Each core has access to a shared-cache similar to the model presented in [28, 122] where the cache is divided into A partitions represented as $W = \{w_1, w_2, .., w_A\}$. Each CP is treated as a mutually exclusive and preemptive resource, and a ready task must acquire the required CPs before it can execute on its assigned core. The cache-management technique proposed by Xu et al. [122] for dynamically administering the CPs to ready HPTs, which can be implemented at the OS-level, is assumed in this work.

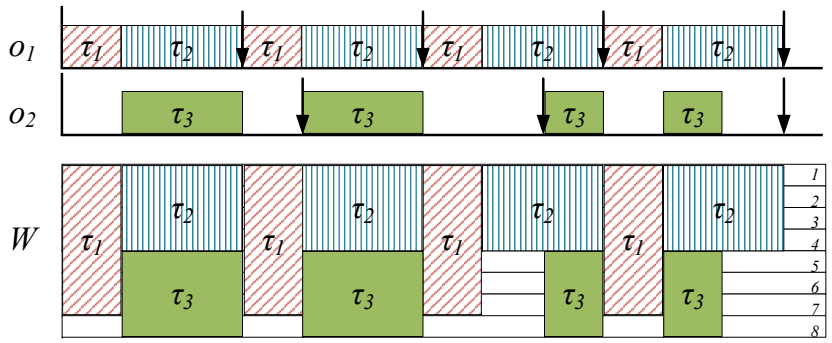


Figure 6-1: Problem setup with three tasks τ_1, τ_2 and τ_3 scheduled onto a (M=2, A=8) multicore system with both core- and cache-blocking.

Each task $\tau_i(e_i, D_i, T_i, cp_i, o_i, \pi_i, \phi_i)$ is defined by a WCET e_i, a relative deadline D_i, an inter-arrival time T_i, CP requirement cp_i, a local priority among tasks assigned to the same core π_i, and a global priority due to the globally shared CPs ϕ_i such that ϕ_i has a higher priority than ϕ_j for all $i<j$. Implicit task deadlines (D_i=T_i) permit the utilization of a task to be defined as u_i=e_i/T_i. τ_i is further characterized by a sequence of jobs where r_i^J and d_i^J is the release-time and deadline of a specific job τ_i^J, respectively and the response-time R_i is the worst-case finish time of all jobs in τ_i. Thus, a taskset is only schedulable if R_i of each τ_i is less than D_i. Similar to [122], cp_i is chosen to be the smallest CPs that lead to a minimum e_i. All non-negligible overheads are factored into the WCET of the tasks to ensure simplicity in the analysis framework.

Figure 6-1 depicts such a dynamic CP setup with tasks $\tau_1(1, 3, 3, 7, 1, 1, 1)$, $\tau_2(2, 3, 3, 4, 1, 2, 2)$ and $\tau_3(2, 4, 4, 4, 2, 1, 3)$ scheduled onto a (M=2, A=8) multicore system. τ_2 is blocked by HPT τ_1 executing on the same core while τ_3 is also blocked due to insufficient CPs during HPT τ_1's execution despite being allocated on a different core. However, τ_3 resumes execution in parallel with τ_2 since the cache can easily accommodate the cumulative CP requirements of τ_2 and τ_3. It must be noted that a core-based scheduling scheme, on the other hand, fails to schedule the taskset with the same system setting since 7 CPs would be statically allocated to o_1 to accommodate cp_1. The remaining 1 CP allocated to o_2 would be insufficient to allow τ_3 to execute.

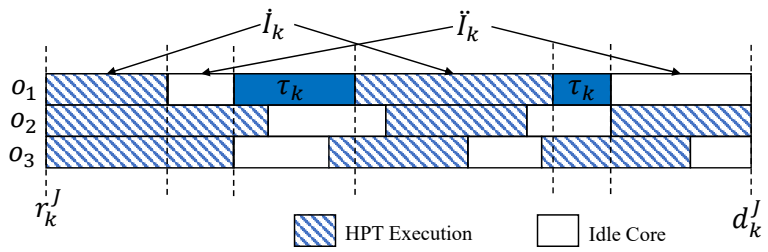


Figure 6-2: Problem window to calculate upper-bound interference on τ_k.

6.2 Schedulability Analysis

In this section, we derive a schedulability test for the problem setup defined in the previous section. We utilize the problem window approach to determine the maximum response time of the problem task τ_k. Figure 6-2 shows the execution of τ_k^J in its problem window along with the execution of its HPTs, where τ_k is assigned to o_1. The problem window is defined between intervals $[r_k^J, d_k^J]$. τ_k^J can be seen to experience two types of blocking. $\dot{I}_k$ is the interference caused due to execution of HPTs on the same core as τ_k, i.e., $\tau_i \in \dot{P}_k$, where $\dot{P}_k$ represents the set of HPTs of τ_k allocated to the same core as τ_k. However, when no HPTs are executing on o_1, τ_k^J can still be prevented from execution during certain intervals. Since the scheduling algorithm is work-conserving, this blocking must be due to CP contention of executing HPTs on other cores which can occur whenever the combined CPs utilized by these HPTs is greater than the CPs required by τ_k, i.e., at any time instant, τ_k is blocked from execution if $\sum cp_i \geq \widehat{cp}_k$, for $\tau_i \in \ddot{P}_k$ where $\ddot{P}_k$ represents the set of HPTs of τ_k not allocated to the same core as τ_k, and $\widehat{cp}_k = A - cp_k + 1$. We classify this interference as $\ddot{I}_k$. This can be further elobrated in Figure 6-3 where both interferences from Figure 6-2 are segregated into seperate blocks and a minimum of $\widehat{cp}_k$ CPs are required to prevent τ_k from executing.

The resultant schedulability test for τ_k can be rendered as:

$$e_k + max(\dot{I}_k^J + \ddot{I}_k^J) \leq D_k \qquad \forall J \in \tau_k \tag{6.1}$$

An upper-bound for $\dot{I}_k^J$ is first determined. Isolating $\dot{I}_k^J$ enables a direct relation with the interference experienced by tasks executing in fixed priority preemptive scheduling algorithms for single-core processors, where a task can only be blocked by HPTs executing on the same core. On single-cores, the critical instant is known and corresponds to

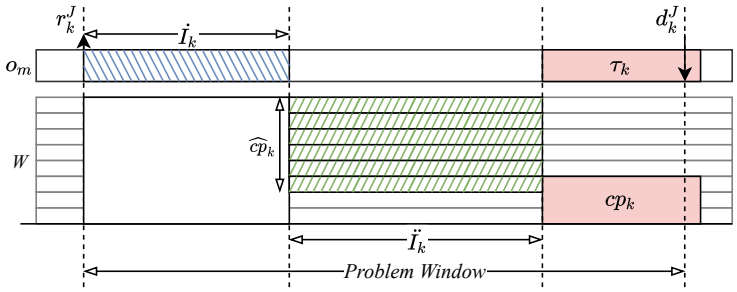


Figure 6-3: Interference from HPTs seperated into seperate blocks to represent how τ_k is blocked within its problem window.

the synchronous release of HPTs. Thus, Audsley's iterative response-time analysis (RTA) schedulability test [35] can be used to find the maximum response time:

$$\dot{R}_k = e_k + \sum_{\tau_i \in \dot{P}_k} \left\lceil \frac{\dot{R}_k}{T_i} \right\rceil e_i \tag{6.2}$$

where $\dot{I}_k$ is implicitly included in the resultant response time as $\dot{I}_k = \dot{R}_k - e_k$.

For $\ddot{I}_k$, since the CPs are a global resource and $\ddot{P}_k$ can acquire any of the A CPs to preemptively execute and ensure a work-conserving schedule, the critical instant cannot be assumed to be the synchronous release of its HPTs. Therefore, an upper-bound on the interference must be found.

As such, it is important to define the workload of a task τ_i during interval $[r_k^J, d_k^J]$ as $W_k^i(r_k^J, d_k^J)$ and its corresponding interference on τ_k as $I_k^i(r_k^J, d_k^J)$. $W_k^i(r_k^J, d_k^J)$ represents on the amount of computation time that τ_i requires within the specified interval. An HPT's interference cannot be greater than its workload within the specified interval and determining the workload of each HPT is a first step to calculating the interference $\ddot{I}_k$ [12].

Given that an HPT executes for $W_k^i(r_k^J, d_k^J)$ time units while occupying cp_i CPs within the problem window and since a minimum number of $\widehat{cp}_k$ CPs can block τ_k from execution, the combined CPs that need to be occupied by $\ddot{P}_k$ during the problem window can be bounded by [122]:

$$\sum_{\tau_i \in \ddot{P}_k} min(cp_i, \widehat{cp}_k) W_k^i(r_k^J, d_k^J) \tag{6.3}$$

Therefore, an upper bound on the interfrence time $\ddot{I}_k$ caused by such HPTs can be found by dividing their combined CP usage (Eq. 6.3) by the minimum number of CPs to block

τ_k:

$$\ddot{I}_k(r_k^J, d_k^J) \geq \frac{\sum_{\tau_i \in \ddot{P}_k} min(cp_i, \widehat{cp}_k) W_k^i(r_k^J, d_k^J)}{\widehat{cp}_k} \tag{6.4}$$

Bertogna et al. [12] proposed a method to determine the workload of each HPT by characterizing their jobs as carry-in, carry-out and body jobs, resulting in:

$$W_k^i(r_k^J,d_k^J) = N_i(r_k^J,d_k^J)e_i + min(e_i,(d_k^J - r_k^J) + D_i - e_i - N_i(r_k^J,d_k^J)T_i) \tag{6.5}$$

where $N_i(r_k^J, d_k^J) = \left\lfloor \frac{(d_k^J - r_k^J) + D_i + e_i}{T_i} \right\rfloor$.

The interference $\ddot{I}_k$ can now be estimated by performing RTA on τ_k. The workload of each $\tau_i \in \ddot{P}_k$ is calculated at each iteration using Eq. (6.5) between intervals defined by the monotonically increasing problem window size specified in each iteration. Therefore, the maximum response time of τ_k due to $\ddot{I}_k$ can be calculated as:

$$\ddot{R}_k = e_k + \left\lfloor \frac{\sum_{\tau_i \in \ddot{P}_k} min(cp_i, \widehat{cp}_k) W_k^i(\ddot{R}_k)}{\widehat{cp}_k} \right\rfloor \tag{6.6}$$

where $W_k^i(\ddot{R}_k)$ can be calculated from Eq. (6.5) for the interval specified by $\ddot{R}_k$ and $\ddot{I}_k$. Based on Eqs. (6.2), (6.4) and (6.6), an upper-bound on the total interference on τ_k can be calculated as:

$$R_k = e_k + \sum_{\tau_i \in \dot{P}_k} \left\lceil \frac{R_k}{T_i} \right\rceil e_i + \left\lfloor \frac{1}{\widehat{cp}_k} \sum_{\tau_i \in \ddot{P}_k} min(cp_i, \widehat{cp}_k) W_k^i(R_k) \right\rfloor \tag{6.7}$$

6.3 Discussion

In this chapter, we presented a schedulability analysis for dynamic CP scheduling on multicore real-time systems. However, the response time attained through this method can be pessimistic since it does not consider the specific combinations of $\tau_i \in \ddot{P}_k$ that can block τ_k and, thus, fails to reflect the low contention scenario, i.e., time instants when $\sum cp_i < \widehat{cp}_k \ \ \forall \tau_i \in \ddot{P}_k$. Therefore, we are currently working on devising improvements to Eq. (6.7) to tighten the upper-bound on cache interference. Towards this aim, we must determine how a given set of HPTs $\tau_i \in \ddot{P}_k$ can block τ_k, i.e., could a single HPT block τ_k or would there be combinations of HPTs that, when run in parallel, could cause enough cache interference to block τ_k from execution.

Chapter 7

Conclusion

Energy-efficiency has become a growing concern worldwide. Particular to real-time systems, the constant demand for higher performance, portability and reliability has led to a proliferation of energy-efficient algorithms for both homogeneous and heterogeneous multicore system. However, existing algorithms are oblivious to unpredictable nature of shared caches.

This thesis introduced and evaluated various aspects of cache-partitioning on energy-efficient scheduling algorithms for both homogeneous and heterogeneous multicore real-time systems in order to cater for the unpredictability posed by shared caches and to accommodate the shared cache energy into the energy minimization problem.

An improved system and task model was first introduced in order to incorporate the shared cache along with its non-linear memory latency scaling. The presented model was evaluated for frame-based tasksets on a single-core system. This work was then extended for the homogeneous multicore system where a cache-dependency graph was developed to model the dynamic cache-partition interference between tasks allocated to different cores, thus, allowing existing well-established algorithms to utilize this model in their energy-minimization problem. Since existing algorithms primarily focus on minimizing only core-level energy consumption, energy-efficient scheduling algorithms for the cache-level and system-level were also proposed. The experimental results showed a 5× improvement in percentage energy savings for system-level energy minimization against the base-line approach.

The same problem was then introduced into the heterogeneous multicore domain, where a holistic approach to minimize system-level energy consumption was proposed for periodic tasks in a static CP scheme. Factors affecting the independent memory latency cycles and cache-paritioning techniques were thorougly analyzed in order to design a heuristic scheduling algorithm. Results show an average of 24.28% 9.25% and maximum of 61.1%, 33.3% percentage energy savings for system-level energy minimization against two state-of-the-art algorithms, respectively.

Finally, a dynamic CP schedulability analysis for periodic tasks was proposed. This

schedulability analysis is currently a work in progress and is intended for future techniques on dynamic CP energy-efficient scheduling of periodic tasks on multicore real-time systems.

The experimental results of this thesis prove our hypothesis that improved task-models along with inclusion of the shared-cache into the energy-minimization problem can result in energy-saving gains for system-level energy consumptions.

7.1 Future Prospects

Thermal-aware scheduling has received significant interest lately for both homogeneous and heterogeneous multicores [1, 39, 75]. Concentration of specific components of the CPU for computation or resource handling can produce thermal hotspots on the CPU die. These hotspots can threaten processor performance and reliability and, therefore, are a high risk factor for time and safety critical applications. [75,111]. Depending upon the task allocation strategy and CP scheme adopted, thermal hotspots may arise on specific CPs that are constantly and repeatedly used by a core. Determining the impact that CP techniques have on the thermal condition of the processor is an interesting future extension of the work proposed in this thesis.

Game theoretical models have shown potential in solving complex resourse management problems especially in the domain of control systems [58,59]. This has motivated researchers to utilize such models for the computing domain to effectively schedule tasks onto multicore and manycore systems [97]. Introducing game theoretical models into real-time systems scheduling, particularly in the cache-aware domain, can produce favourable results for both increasing schedulability and minimizing energy consumption.

List of Publications

- S. Z. Sheikh and M. A. Pasha, "Energy-Efficient Multicore Scheduling for Hard Real-Time Systems: A Survey," 2018, ACM Transactions in Embedded Computing Systems (TECS) 17, 6, Article 94, 26 pages. (Chapter 2)

- S. Z. Sheikh and M. A. Pasha, "An Improved Model for System-Level Energy Minimization on Real-Time Systems," 2019, IEEE 27th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS), Rennes, FR, 2019, pp. 276-282. (Chapter 3)

- S. Z. Sheikh and M. A. Pasha, "Energy-Efficient Real-Time Scheduling on Multicores: A Novel Approach to Model Cache Contention" 2020, ACM Transactions in Embedded Computing Systems (TECS) 19, 4, Article 28, 25 pages. (Chapter 4)

- S. Z. Sheikh and M. A. Pasha, "Cache-Aware Energy-Efficient Scheduling on Heterogeneous Multicores" IEEE Transactions in Parallel and Distributed Systems (TPDS) [Under Review] (Chapter 5)

- S. Z. Sheikh and M. A. Pasha, "Dynamic Cache-Partition Schedulability Analysis for Partitioned Scheduling on Multicore Real-Time Systems. IEEE Letters of Computer Society (LOCS) [Accepted] (Chapter 6)

Bibliography

[1] Mohamad Hammam Alsafrjalani and Tosiron Adegbija. Tasat: Thermal-aware scheduling and tuning algorithm for heterogeneous and configurable embedded systems. In *Proceedings of the 2018 on Great Lakes Symposium on VLSI*, GLSVLSI '18, page 75–80, New York, NY, USA, 2018. Association for Computing Machinery.

[2] Muhammad Ali Awan. *Energy and Temperature Aware Real-Time Systems*. PhD thesis, University of Porto, 2014.

[3] Muhammad Ali Awan and Stefan M Petters. Energy-aware partitioning of tasks onto a heterogeneous multi-core platform. In *IEEE 19th Conference on Real-Time and Embedded Technology and Applications Symposium (RTAS), 2013*, pages 205–214. IEEE, 2013.

[4] Hakan Aydin, Rami Melhem, Daniel Mossé, and Pedro Mejía-Alvarez. Determining optimal processor speeds for periodic real-time tasks with different power characteristics. In *13th Euromicro Conference on Real-Time Systems, 2001.*, pages 225–232. IEEE, 2001.

[5] Hakan Aydin and Qi Yang. Energy-aware partitioning for multiprocessor real-time systems. In *International Proceedings on Parallel and Distributed Processing Symposium, 2003*, pages 9–pp. IEEE, 2003.

[6] Mario Bambagini, Mauro Marinoni, Hakan Aydin, and Giorgio Buttazzo. Energy-aware scheduling for real-time systems: A survey. *ACM Trans. Embed. Comput. Syst.*, 15(1):7:1–7:34, January 2016.

[7] Sanjoy Baruah, Marko Bertogna, and Giorgio Buttazzo. *Multiprocessor Scheduling for Real-Time Systems*. Springer, 2015.

[8] Sanjoy K Baruah, Neil K Cohen, C Greg Plaxton, and Donald A Varvel. Proportionate progress: A notion of fairness in resource allocation. *Algorithmica*, 15(6):600–625, 1996.

[9] Michela Becchi and Patrick Crowley. Dynamic thread assignment on heterogeneous multiprocessor architectures. In *Proceedings of the 3rd conference on Computing frontiers*, pages 29–40. ACM, 2006.

[10] Lotfi Belkhir and Ahmed Elmeligi. Assessing ict global emissions footprint: Trends to 2040 & recommendations. *Journal of Cleaner Production*, 177:448–463, 2018.

[11] Brice Berna and Isabelle Puaut. Pdpa: period driven task and cache partitioning algorithm for multi-core systems. In *Proceedings of the 20th International Conference on Real-Time and Network Systems*, pages 181–189. ACM, 2012.

[12] M. Bertogna, M. Cirinei, and G. Lipari. Schedulability Analysis of Global Scheduling Algorithms on Multiprocessor Platforms. *IEEE Trans. Parallel Distrib. Syst.*, 20(4):553–566, April 2009.

[13] Marko Bertogna. Real-time scheduling analysis for multiprocessor platforms. *PhD Defense*, 2008.

[14] Khurram Bhatti, Cecile Belleudy, and Michel Auguin. Power management in real time embedded systems through online and adaptive interplay of dpm and dvfs policies. In *IEEE/IFIP 8th International Conference on Embedded and Ubiquitous Computing (EUC), 2010*, pages 184–191. IEEE, 2010.

[15] Muhammad Khurram Bhatti, Cécile Belleudy, and Michel Auguin. An inter-task real time dvfs scheme for multiprocessor embedded systems. In *Conference on Design and Architectures for Signal and Image Processing (DASIP), 2010*, pages 136–143. IEEE, 2010.

[16] Ashikahmed Bhuiyan, Zhishan Guo, Abusayeed Saifullah, Nan Guan, and Haoyi Xiong. Energy-efficient real-time scheduling of dag tasks. *ACM Transactions on Embedded Computing Systems (TECS)*, 17(5):1–25, 2018.

[17] Ashikahmed Bhuiyan, Di Liu, Aamir Khan, Abusayeed Saifullah, Nan Guan, and Zhishan Guo. Energy-efficient parallel real-time scheduling on clustered multi-core. *IEEE Transactions on Parallel and Distributed Systems*, 31(9):2097–2111, 2020.

[18] Ashikahmed Bhuiyan, Sai Sruti, Zhishan Guo, and Kecheng Yang. Precise scheduling of mixed-criticality tasks by varying processor speed. In *Proceedings of the 27th International Conference on Real-Time Networks and Systems*, pages 123–132, 2019.

[19] Christian Bienia, Sanjeev Kumar, Jaswinder Pal Singh, and Kai Li. The parsec benchmark suite: Characterization and architectural implications. In *Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques*, PACT '08, pages 72–81, New York, NY, USA, 2008. ACM.

[20] Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R Hower, Tushar Krishna, Somayeh Sardashti, et al. The gem5 simulator. *ACM SIGARCH Computer Architecture News*, 39(2):1–7, 2011.

[21] Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K. Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R. Hower, Tushar Krishna, Somayeh Sardashti, Rathijit Sen, Korey Sewell, Muhammad Shoaib, Nilay Vaish, Mark D. Hill, and David A. Wood. The Gem5 Simulator. *SIGARCH Comput. Archit. News*, 39(2):1–7, August 2011.

[22] Stephen Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[23] Bach D Bui, Marco Caccamo, Lui Sha, and Joseph Martinez. Impact of cache partitioning on multi-tasking real time embedded systems. In *Embedded and Real-Time Computing Systems and Applications, 2008. RTCSA'08. 14th IEEE International Conference on*, pages 101–110. IEEE, 2008.

[24] Alan Burns, Robert I Davis, P Wang, and Fengxiang Zhang. Partitioned edf scheduling for multiprocessors using a c= d task splitting scheme. *Real-Time Systems*, 48(1):3–33, 2012.

[25] Jason F. Cantin and Mark D. Hill. Cache performance for selected spec cpu2000 benchmarks. *SIGARCH Comput. Archit. News*, 29(4):13–18, September 2001.

[26] Gustavo A Chaparro-Baquero, Soamar Homsi, Omara Vichot, Shaolei Ren, Gang Quan, and Shangping Ren. Cache allocation for fixed-priority real-time scheduling on multi-core platforms. In *Computer Design (ICCD), 2015 33rd IEEE International Conference on*, pages 589–596. IEEE, 2015.

[27] Gang Chen, Biao Hu, Kai Huang, Alois Knoll, Di Liu, and Todor Stefanov. Automatic cache partitioning and time-triggered scheduling for real-time mpsocs. In *2014 International Conference on Reconfigurable Computing and FPGAs (ReConFig 2014)*, 2014.

[28] Gang Chen, Kai Huang, Jia Huang, and Alois Knoll. Cache partitioning and scheduling for energy optimization of real-time mpsocs. In *Application-Specific Systems, Architectures and Processors (ASAP), 2013 IEEE 24th International Conference on*, pages 35–41. IEEE, 2013.

[29] Gang Chen, Kai Huang, and Alois Knoll. Energy optimization for real-time multiprocessor system-on-chip with optimal dvfs and dpm combination. *ACM Trans. Embed. Comput. Syst.*, 13(3s):111:1–111:21, March 2014.

[30] Jian-Jia Chen, Heng-Ruey Hsu, Kai-Hsiang Chuang, Chia-Lin Yang, Ai-Chun Pang, and Tei-Wei Kuo. Multiprocessor energy-efficient scheduling with task migration considerations. In *Real-Time Systems, 2004. ECRTS 2004. Proceedings. 16th Euromicro Conference on*, pages 101–108. IEEE, 2004.

[31] Jian-Jia Chen, Heng-Ruey Hsu, and Tei-Wei Kuo. Leakage-aware energy-efficient scheduling of real-time tasks in multiprocessor systems. In *Proceedings of the 12th IEEE Real-Time and Embedded Technology and Applications Symposium, 2006.*, pages 408–417. IEEE, 2006.

[32] Hui Cheng and Steve Goddard. Integrated device scheduling and processor voltage scaling for system-wide energy conservation. In *Proceedings of the 2005 workshop on power aware real-time computing*. Citeseer, 2005.

[33] Hoon Sung Chwa, Jaebaek Seo, Jinkyu Lee, and Insik Shin. Optimal real-time scheduling on two-type heterogeneous multicore platforms. In *Real-Time Systems Symposium, 2015 IEEE*, pages 119–129. IEEE, 2015.

[34] Alexei Colin, Arvind Kandhalu, and Ragunathan (Raj) Rajkumar. Energy-efficient allocation of real-time applications onto single-isa heterogeneous multi-core processors. *Journal of Signal Processing Systems*, 84(1):91–110, Jul 2016.

[35] Robert I. Davis, Sebastian Altmeyer, Leandro S. Indrusiak, Claire Maiza, Vincent Nelis, and Jan Reineke. An extensible framework for multicore response time analysis. *Real-Time Systems*, 54(3):607–661, Jul 2018.

[36] Robert I. Davis and Alan Burns. A survey of hard real-time scheduling for multiprocessor systems. *ACM Comput. Surv.*, 43(4):35:1–35:44, October 2011.

[37] E. Del Sozzo, G. C. Durelli, E. M. G. Trainiti, A. Miele, M. D. Santambrogio, and C. Bolchini. Workload-aware power optimization strategy for asymmetric multiprocessors. In *2016 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 531–534, March 2016.

[38] Emanuele Del Sozzo. Workload-aware Power Optimization Strategy for Heterogeneous Systems. Master's thesis, Politecnico di Milano, 2015.

[39] G. Desirena-López, A. Ramírez-Treviño, J. L. Briz, C. R. Vázquez, and D. Gómez-Gutiérrez. Thermal-aware real-time scheduling using timed continuous petri nets. *ACM Trans. Embed. Comput. Syst.*, 18(4), July 2019.

[40] Vinay Devadas and Hakan Aydin. On the interplay of voltage/frequency scaling and device power management for frame-based real-time embedded applications. *IEEE Transactions on Computers*, 61(1):31–44, 2012.

[41] Steven Diamond and Stephen Boyd. Cvxpy: A python-embedded modeling language for convex optimization. *The Journal of Machine Learning Research*, 17(1):2909–2913, 2016.

[42] A. Edun, R. Vazquez, A. Gordon-Ross, and G. Stitt. Dynamic scheduling on heterogeneous multicores. In *2019 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 1685–1690, March 2019.

[43] Abdullah Elewi, Mohamed Shalan, Medhat Awadalla, and Elsayed M Saad. Energy-efficient task allocation techniques for asymmetric multiprocessor embedded systems. *ACM Transactions on Embedded Computing Systems (TECS)*, 13(2s):71, 2014.

[44] Xing Fu, Khairul Kabir, and Xiaorui Wang. Cache-aware utilization control for energy efficiency in multi-core real-time systems. In *23rd Euromicro Conference on Real-Time Systems (ECRTS), 2011*, pages 102–111. IEEE, 2011.

[45] Marco ET Gerards, Johann L Hurink, and Jan Kuper. On the interplay between global dvfs and scheduling tasks with precedence constraints. *IEEE Transactions on Computers*, 64(6):1742–1754, June 2015.

[46] Marco ET Gerards and Jan Kuper. Optimal dpm and dvfs for frame-based real-time systems. *ACM Transactions on Architecture and Code Optimization (TACO)*, 9(4):41, 2013.

[47] Giovani Gracioli, Ahmed Alhammad, Renato Mancuso, Antonio Augusto Frohlich, and Rodolfo Pellizzoni. A survey on cache management mechanisms for real-time embedded systems. *ACM Comput. Surv.*, 48(2):32:1–32:36, November 2015.

[48] Nan Guan, Martin Stigge, Wang Yi, and Ge Yu. Cache-aware scheduling and analysis for multicores. In *Proceedings of the seventh ACM international conference on Embedded software*, pages 245–254. ACM, 2009.

[49] Zhishan Guo, Ashikahmed Bhuiyan, Di Liu, Aamir Khan, Abusayeed Saifullah, and Nan Guan. Energy-efficient real-time scheduling of dags on clustered multi-core platforms. In *2019 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 156–168. IEEE, 2019.

[50] Zhishan Guo, Ashikahmed Bhuiyan, Abusayeed Saifullah, Nan Guan, and Haoyi Xiong. Energy-efficient multi-core scheduling for real-time dag tasks. In *LIPIcs-Leibniz International Proceedings in Informatics*, volume 76. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.

[51] Zhishan Guo, Kecheng Yang, and Fan Yao Amro Awad. Inter-task cache interference aware partitioned real-time scheduling. In *Proceedings of the 35th Symposium On Applied Computing (SAC)*, page Accepted. Association for Computing Machinery, March 2020.

[52] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown. Mibench: A free, commercially representative embedded benchmark suite. In *Proceedings of the Fourth Annual IEEE International Workshop on Workload Characterization. WWC-4 (Cat. No.01EX538)*, pages 3–14, Dec 2001.

[53] Jian-Jun Han, Xiaodong Wu, Dakai Zhu, Hai Jin, Laurence T Yang, and Jean-Luc Gaudiot. Synchronization-aware energy management for vfi-based multicore real-time systems. *IEEE Transactions on Computers*, 61(12):1682–1696, 2012.

[54] John L Hennessy and David A Patterson. *Computer architecture: a quantitative approach*. Elsevier, 2011.

[55] J. L. Henning. Spec cpu2000: measuring cpu performance in the new millenniumm. *Computer*, 33(7):28–35, July 2000.

[56] Kai Huang, Luca Santinelli, Jian-Jia Chen, Lothar Thiele, and Giorgio C Buttazzo. Periodic power management schemes for real-time event streams. In *Proceedings of the 48th IEEE Conference on Decision and Control, 2009 held jointly with the 2009 28th Chinese Control Conference. CDC/CCC 2009.*, pages 6224–6231. IEEE, 2009.

[57] Tohru Ishihara and Hiroto Yasuura. Voltage scheduling problem for dynamically variable voltage processors. In *Proceedings of the 1998 international symposium on Low power electronics and design*, pages 197–202. ACM, 1998.

[58] H. Jaleel, W. Abbas, and J. S. Shamma. Robustness of stochastic learning dynamics to player heterogeneity in games. In *2019 IEEE 58th Conference on Decision and Control (CDC)*, pages 5002–5007, 2019.

[59] I. Jang, H. Shin, and A. Tsourdos. A game-theoretical approach to heterogeneous multi-robot task assignment problem with minimum workload requirements. In *2017 Workshop on Research, Education and Development of Unmanned Aerial Systems (RED-UAS)*, pages 156–161, 2017.

[60] Ravindra Jejurikar and Rajesh Gupta. Dynamic voltage scaling for systemwide energy minimization in real-time embedded systems. In *the 2004 International Symposium on Low Power Electronics and Design*, pages 78–81, Aug 2004.

[61] Ravindra Jejurikar, Cristiano Pereira, and Rajesh Gupta. Leakage aware dynamic voltage scaling for real-time embedded systems. In *Proceedings of the 41st annual Design Automation Conference*, pages 275–280. ACM, 2004.

[62] Hyeran Jeon, Woo Hyong Lee, and Sung Woo Chung. Load unbalancing strategy for multicore embedded processors. *IEEE Transactions on Computers*, 59(10):1434–1440, 2010.

[63] Jaeyeon Kang and Sanjay Ranka. Dvs based energy minimization algorithm for parallel machines. In *2008 IEEE International Symposium on Parallel and Distributed Processing*, pages 1–12, April 2008.

[64] Omer Khan and Sandip Kundu. A self-adaptive scheduler for asymmetric multi-cores. In *Proceedings of the 20th symposium on Great lakes symposium on VLSI*, pages 397–400. ACM, 2010.

[65] Hyoseung Kim, Arvind Kandhalu, and Ragunathan Rajkumar. A coordinated approach for practical os-level cache management in multi-core real-time systems. In *25th Euromicro Conference on Real-Time Systems*, pages 80–89, July 2013.

[66] Hyoseung Kim, Arvind Kandhalu, and Ragunathan Rajkumar. Coordinated cache management for predictable multi-core real-time systems. *Technical report*, 2014.

[67] Hyoseung Kim and Ragunathan (Raj) Rajkumar. Predictable shared cache management for multi-core real-time virtualization. *ACM Trans. Embed. Comput. Syst.*, 17(1):22:1–22:27, December 2017.

[68] Y. Kim, A. More, E. Shriver, and T. Rosing. Application performance prediction and optimization under cache allocation technology. In *2019 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 1285–1288, March 2019.

[69] Fanxin Kong, Nan Guan, Qingxu Deng, and Wang Yi. Energy-efficient scheduling for parallel real-time tasks based on level-packing. In *Proceedings of the 2011 ACM Symposium on Applied Computing*, pages 635–640. ACM, 2011.

[70] Fanxin Kong, Yiqun Wang, Qingxu Deng, and Wang Yi. Minimizing multi-resource energy for real-time systems with discrete operation modes. In *2010 22nd Euromicro Conference on Real-Time Systems*, pages 113–122, July 2010.

[71] Fanxin Kong, Wang Yi, and Qingxu Deng. Energy-efficient scheduling of real-time tasks on cluster-based multicores. In *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2011*, pages 1–6. IEEE, 2011.

[72] David Koufaty, Dheeraj Reddy, and Scott Hahn. Bias scheduling in heterogeneous multi-core architectures. In *Proceedings of the 5th European Conference on Computer Systems*, EuroSys '10, page 125–138, New York, NY, USA, 2010. Association for Computing Machinery.

[73] VP Kozyrev. Estimation of the execution time in real-time systems. *Programming and Computer Software*, 42(1):41–48, 2016.

[74] Chin-Fu Kuo and Yung-Feng Lu. Energy-efficient assignment for tasks on non-dvs heterogeneous multiprocessor system. In *Proceedings of the 2014 Conference on Research in Adaptive and Convergent Systems*, RACS '14, page 314–319, New York, NY, USA, 2014. Association for Computing Machinery.

[75] Y. Lee, H. S. Chwa, K. G. Shin, and S. Wang. Thermal-aware resource management for embedded real-time systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 37(11):2857–2868, 2018.

[76] Yann-Hang Lee, Krishna P Reddy, and C Mani Krishna. Scheduling techniques for reducing leakage power in hard real-time systems. In *Proceedings 15th Euromicro Conference on Real-Time Systems, 2003.*, pages 105–112. IEEE, 2003.

[77] Vincent Legout, Mathieu Jan, and Laurent Pautet. A scheduling algorithm to reduce the static energy consumption of multiprocessor real-time systems. In *Proceedings of the 21st International Conference on Real-Time Networks and Systems*, pages 99–108. ACM, 2013.

[78] D. Li and J. Wu. Energy-aware scheduling for frame-based tasks on heterogeneous multiprocessor platforms. In *2012 41st International Conference on Parallel Processing*, pages 430–439, 2012.

[79] Keqin Li. Scheduling precedence constrained tasks with reduced processor energy on multiprocessor computers. *IEEE Transactions on Computers*, 61(12):1668–1681, Dec 2012.

[80] D. Liu, J. Spasic, G. Chen, and T. Stefanov. Energy-efficient mapping of real-time streaming applications on cluster heterogeneous mpsocs. In *2015 13th IEEE Symposium on Embedded Systems For Real-time Multimedia (ESTIMedia)*, pages 1–10, 2015.

[81] D. Liu, J. Spasic, P. Wang, and T. Stefanov. Energy-efficient scheduling of real-time tasks on heterogeneous multicores using task splitting. In *2016 IEEE 22nd International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, pages 149–158, Aug 2016.

[82] Andrea Lodi, Silvano Martello, and Michele Monaci. Two-dimensional packing problems: A survey. *European Journal of Operational Research*, 141(2):241 – 252, 2002.

[83] Johan Löfberg. Yalmip: A toolbox for modeling and optimization in matlab. In *Proceedings of the CACSD Conference*, volume 3. Taipei, Taiwan, 2004.

[84] Jiong Luo and Niraj K Jha. Power-efficient scheduling for heterogeneous distributed real-time embedded systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 26(6):1161–1170, 2007.

[85] Yangchun Luo, Venkatesan Packirisamy, Wei-Chung Hsu, and Antonia Zhai. Energy efficient speculative threads: dynamic thread allocation in same-isa heterogeneous multicore systems. In *Proceedings of the 19th international conference on Parallel architectures and compilation techniques*, pages 453–464. ACM, 2010.

[86] Mingsong Lv, Nan Guan, Jan Reineke, Reinhard Wilhelm, and Wang Yi. A survey on static cache analysis for real-time systems. *Leibniz Transactions on Embedded Systems*, 3(1):05–1–05:48, 2016.

[87] Amjad Mahmood, Salman A. Khan, Fawzi Albalooshi, and Noor Awwad. Energy-aware real-time task scheduling in multiprocessor systems using a hybrid genetic algorithm. *Electronics*, 6(2), 2017.

[88] Rajib Mall. *Real-time systems: theory and practice*. Pearson Education India, 2009.

[89] José Luis March, Julio Sahuquillo, Salvador Petit, Houcine Hassan, and José Duato. Power-aware scheduling with effective task migration for real-time multicore embedded systems. *Concurrency and Computation: Practice and Experience*, 25(14):1987–2001, 2013.

[90] Alessandra Melani, Marko Bertogna, Robert I Davis, Vincenzo Bonifaci, Alberto Marchetti-Spaccamela, and Giorgio Buttazzo. Exact response time analysis for fixed priority memory-processor co-scheduling. *IEEE Transactions on Computers*, 66(4):631–646, April 2017.

[91] Sparsh Mittal. A survey of architectural techniques for improving cache power efficiency. *Sustainable Computing: Informatics and Systems*, 4(1):33–43, 2014.

[92] Sparsh Mittal. A survey of techniques for cache locking. *ACM Trans. Des. Autom. Electron. Syst.*, 21(3):49:1–49:24, May 2016.

[93] Takashi Nakada. Low-power circuit technologies. In *Normally-Off Computing*, pages 11–25. Springer, 2017.

[94] S. Pagani, A. Pathania, M. Shafique, J. Chen, and J. Henkel. Energy Efficiency for Clustered Heterogeneous Multicores. *IEEE Trans. Parallel Distrib. Syst.*, 28(5):1315–1330, May 2017.

[95] Antonio Paolillo, Joël Goossens, Pradeep M Hettiarachchi, and Nathan Fisher. Power minimization for parallel real-time systems with malleable jobs and homogeneous frequencies. In *IEEE 20th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA), 2014*, pages 1–10. IEEE, 2014.

[96] Sangyoung Park, Jaehyun Park, Donghwa Shin, Yanzhi Wang, Qing Xie, Massoud Pedram, and Naehyuck Chang. Accurate modeling of the delay and energy overhead of dynamic voltage and frequency scaling in modern microprocessors. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 32(5):695–708, 2013.

[97] Anuj Pathania, Vanchinathan Venkataramani, Muhammad Shafique, Tulika Mitra, and Jörg Henkel. Distributed scheduling for many-cores using cooperative game theory. In *Proceedings of the 53rd Annual Design Automation Conference*, DAC '16, New York, NY, USA, 2016. Association for Computing Machinery.

[98] Vinicius Petrucci, Orlando Loques, Daniel Mossé, Rami Melhem, Neven Abou Gazala, and Sameh Gobriel. Energy-efficient thread assignment optimization for heterogeneous multicore systems. *ACM Transactions on Embedded Computing Systems (TECS)*, 14(1):15, 2015.

[99] Mihai Pricopi, Thannirmalai Somu Muthukaruppan, Vanchinathan Venkataramani, Tulika Mitra, and Sanjay Vishin. Power-performance modeling on asymmetric multicores. In *International Conference on Compilers, Architecture and Synthesis for Embedded Systems (CASES), 2013*, pages 1–10. IEEE, 2013.

[100] A. Roy, H. Aydin, and D. Zhu. Energy-efficient fault tolerance for real-time tasks with precedence constraints on heterogeneous multicore systems. In *2019 Tenth International Green and Sustainable Computing Conference (IGSC)*, pages 1–8, Oct 2019.

[101] Abusayeed Saifullah, Sezana Fahmida, Venkata P Modekurthy, Nathan Fisher, and Zhishan Guo. Cpu energy-aware parallel real-time scheduling. In *32nd Euromicro Conference on Real-Time Systems (ECRTS 2020)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2020.

[102] Marcus T Schmitz and Bashir M Al-Hashimi. Considering power variations of dvs processing elements for energy minimisation in distributed systems. In *Proceedings of the 14th international symposium on Systems synthesis*, pages 250–255. ACM, 2001.

[103] Euiseong Seo, Jinkyu Jeong, Seonyeong Park, and Joonwon Lee. Energy efficient scheduling of real-time tasks on multicore processors. *IEEE Transactions on Parallel and Distributed Systems*, 19(11):1540–1552, Nov 2008.

[104] Shaoxiong Hua and Gang Qu. Power minimization techniques on distributed real-time systems by global and local slack management. In *Proceedings of the ASP-DAC 2005. Asia and South Pacific Design Automation Conference, 2005.*, volume 2, pages 830–835 Vol. 2, NY USA, Jan 2005. ACM.

[105] A. Suyyagh and Z. Zilic. Energy and task-aware partitioning on single-isa clustered heterogeneous processors. *IEEE Trans. Parallel Distrib. Syst.*, pages 1–1, 2019.

[106] Yudong Tan and Vincent Mooney. Integrated intra-and inter-task cache analysis for preemptive multi-tasking real-time systems. In *International Workshop on Software and Compilers for Embedded Systems*, pages 182–199. Springer, 2004.

[107] Tomohiro Tatematsu, Hideki Takase, Gang Zeng, Hiroyuki Tomiyama, and Hiroaki Takada. Checkpoint extraction using execution traces for intra-task dvfs in embedded systems. In *Sixth IEEE International Symposium on Electronic Design, Test and Application (DELTA), 2011*, pages 19–24. IEEE, 2011.

[108] Mason Thammawichai and Eric C Kerrigan. Energy-efficient real-time scheduling for two-type heterogeneous multiprocessors. *Real-Time Systems*, 54(1):132–165, 2018.

[109] Shyamkumar Thoziyoor, Naveen Muralimanohar, Jung Ho Ahn, and Norman P Jouppi. Cacti 5.1. Technical report, Technical Report HPL-2008-20, HP Labs, 2008.

[110] P. K. Valsan, H. Yun, and F. Farshchi. Taming non-blocking caches to improve isolation in multicore real-time systems. In *2016 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 1–12, April 2016.

[111] Jian Wang and Fu-yuan Hu. Thermal hotspots in cpu die and it's future architecture. In Ran Chen, editor, *Intelligent Computing and Information Science*, pages 180–185, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[112] Weixun Wang and Prabhat Mishra. Leakage-aware energy minimization using dynamic voltage scaling and cache reconfiguration in real-time systems. In *VLSI Design, 2010. VLSID'10. 23rd International Conference on*, pages 357–362. IEEE, 2010.

[113] Weixun Wang and Prabhat Mishra. System-wide leakage-aware energy minimization using dynamic voltage scaling and cache reconfiguration in multitasking systems. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 20(5):902–910, May 2012.

[114] Weixun Wang, Prabhat Mishra, and Sanjay Ranka. Dynamic cache reconfiguration and partitioning for energy optimization in real-time multi-core systems. In *Design Automation Conference (DAC), 2011 48th ACM/EDAC/IEEE*, pages 948–953. IEEE, 2011.

[115] Bryan C Ward. Relaxing resource-sharing constraints for improved hardware management and schedulability. In *2015 IEEE Real-Time Systems Symposium*, pages 153–164. IEEE, 2015.

[116] Bryan C Ward, Jonathan L Herman, Christopher J Kenna, and James H Anderson. Outstanding paper award: Making shared caches more predictable on multicore platforms. In *2013 25th Euromicro Conference on Real-Time Systems*, pages 157–167. IEEE, 2013.

[117] A. Wieder and B. B. Brandenburg. Efficient partitioning of sporadic real-time tasks with shared resources and spin locks. In *2013 8th IEEE International Symposium on Industrial Embedded Systems (SIES)*, pages 49–58, 2013.

[118] Xiaodong Wu, Yuan Lin, Jian-Jun Han, and Jean-Luc Gaudiot. Energy-efficient scheduling of real-time periodic tasks in multicore systems. In *IFIP International Conference on Network and Parallel Computing*, pages 344–357. Springer, 2010.

[119] J. Xiao, S. Altmeyer, and A. Pimentel. Schedulability analysis of non-preemptive realtime scheduling for multicore processors with shared caches. In *IEEE Real-Time Systems Symposium(RTSS)*, pages 199–208, 2017.

[120] M. Xu, L. T. X. Phan, H. Choi, Y. Lin, H. Li, C. Lu, and I. Lee. Holistic resource allocation for multicore real-time systems. In *2019 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 345–356, April 2019.

[121] Meng Xu, Robert Gifford, and Linh Thi Xuan Phan. Holistic multi-resource allocation for multicore real-time virtualization. In *Proceedings of the 56th Annual Design Automation Conference 2019*, DAC '19, pages 168:1–168:6, New York, NY, USA, 2019. ACM.

[122] Meng Xu, Linh Thi Xuan Phan, Hyon-Young Choi, and Insup Lee. Analysis and implementation of global preemptive fixed-priority scheduling with dynamic cache allocation. In *Real-Time and Embedded Technology and Applications Symposium (RTAS), 2016 IEEE*, pages 1–12. IEEE, 2016.

[123] Meng Xu, Linh Thi Xuan Phan, Hyon-Young Choi, Yuhan Lin, Haoran Li, Chenyang Lu, and Insup Lee. Holistic resource allocation for multicore real-time systems. In *2019 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 345–356, April 2019.

[124] Frances Yao, Alan Demers, and Scott Shenker. A scheduling model for reduced cpu energy. In *Proceedings 36th Annual Symposium on Foundations of Computer Science, 1995*, pages 374–382. IEEE, 1995.

[125] Han-Saem Yun and Jihong Kim. On energy-optimal voltage scheduling for fixed-priority hard real-time systems. *ACM Transactions on Embedded Computing Systems (TECS)*, 2(3):393–430, 2003.

[126] Heechul Yun, Po-Liang Wu, Anshu Arya, Cheolgi Kim, Tarek Abdelzaher, and Lui Sha. System-wide energy optimization for multiple dvs components and real-time tasks. *Real-Time Systems*, 47(5):489, 2011.

[127] Gang Zeng, Tetsuo Yokoyama, Hiroyuki Tomiyama, and Hiroaki Takada. Practical energy-aware scheduling for real-time multiprocessor systems. In *15th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications, 2009. RTCSA'09.*, pages 383–392. IEEE, 2009.

[128] Yumin Zhang, Xiaobo Sharon Hu, and Danny Z Chen. Task scheduling and voltage selection for energy minimization. In *Proceedings of the 39th annual Design Automation Conference*, pages 183–188. ACM, 2002.

[129] Liu Zheng. A task migration constrained energy-efficient scheduling algorithm for multiprocessor real-time systems. In *Wireless Communications, Networking and Mobile Computing, 2007. WiCom 2007. International Conference on*, pages 3055–3058. IEEE, 2007.

[130] Xiliang Zhong and Cheng-Zhong Xu. System-wide energy minimization for real-time tasks: Lower bound and approximation. *ACM Transactions on Embedded Computing Systems (TECS)*, 7(3):28, 2008.

[131] Junlong Zhou, Jianming Yan, Tongquan Wei, Mingsong Chen, and Xiaobo Sharon Hu. Energy-adaptive scheduling of imprecise computation tasks for qos optimization in real-time mpsoc systems. In *IEEE Conference on Design, Automation Test in Europe*, pages 1402–1407, March 2017.

[132] Dakai Zhu, Rami Melhem, and Bruce R Childers. Scheduling with dynamic voltage/speed adjustment using slack reclamation in multiprocessor real-time systems. *IEEE Trans. Parallel Distrib. Syst.*, 14(7):686–700, 2003.