Energy-Aware Real-Time Scheduling on Heterogeneous and Homogeneous Platforms in the Era of Parallel Computing

Ashik Ahmed Bhuiyan
University of Central Florida

ENERGY-AWARE REAL-TIME SCHEDULING ON HETEROGENEOUS AND HOMOGENEOUS PLATFORMS IN THE ERA OF PARALLEL COMPUTING

by

ASHIK AHMED BHUIYAN
B.Sc., Bangladesh University of Engineering and Technology, 2013

A dissertation submitted in partial fulfilment of the requirements
for the degree of Doctor of Philosophy
in the Department of Electrical and Computer Engineering
in the College of Engineering and Computer Science
at the University of Central Florida
Orlando, Florida

Summer Term
2021

Major Professor: Zhishan Guo

ABSTRACT

Multi-core processors increasingly appear as an enabling platform for embedded systems, e.g., mobile phones, tablets, computerized numerical controls, etc. The parallel task model, where a task can execute on multiple cores simultaneously, can efficiently exploit the multi-core platform's computational ability. Many computation-intensive systems (e.g., self-driving cars) that demand stringent timing requirements often evolve in the form of parallel tasks. Several real-time embedded system applications demand predictable timing behavior and satisfy other system constraints, such as energy consumption.

Motivated by the facts mentioned above, this thesis studies the approach to integrating the dynamic voltage and frequency scaling (DVFS) policy with real-time embedded system application's internal parallelism to reduce the worst-case energy consumption (WCEC), an essential requirement for energy-constrained systems. First, we propose an energy-sub-optimal scheduler, assuming the per-core speed tuning feature for each processor. Then we extend our solution to adapt the clustered multi-core platform, where at any given time, all the processors in the same cluster run at the same speed. We also present an analysis to exploit a task's probabilistic information to improve the average-case energy consumption (ACEC), a common non-functional requirement of embedded systems.

Due to the strict requirement of temporal correctness, the majority of the real-time system analysis considered the worst-case scenario, leading to resource over-provisioning and cost. The mixed-criticality (MC) framework was proposed to minimize energy consumption and resource over-provisioning. MC scheduling has received considerable attention from the real-time system research community, as it is crucial to designing safety-critical real-time systems. This thesis further addresses energy-aware scheduling of real-time tasks in an MC platform, where tasks with varying criticality levels (i.e., importance) are integrated into a common platform. We propose an algorithm GEDF-VD for scheduling MC tasks with internal parallelism in a multiprocessor platform. We also prove the correctness of GEDF-VD, provide a detailed quantitative evaluation, and reported extensive experimental results. Finally, we present an analysis to exploit a task's probabilistic information at their respective criticality levels. Our proposed approach reduces the average-case energy consumption while satisfying the worst-case timing requirement.

Dedicated to my family members.

ACKNOWLEDGMENTS

First of all, thanks to my almighty lord for providing me the strength, patience, ability to make a successful end of this long journey. Without the shower of his blessing throughout my graduate life, it would be impossible to complete the research successfully.

I was fortunate to have two supervisors, *Zhishan Guo* and *Abusayeed Saifullah* (Wayne State University), in my Ph.D. life. Without them, it wouldn't have been possible to draw a successful ending. I've received an enormous amount of help and support and from both of them. I am infinitely indebted to my supervisors, and I want to express my deep and sincere gratitude. Probably like everyone else, the beginning of this journey was extremely challenging and frustrating. However, they always believe in me, give me the freedom to think and develop the research that attracts me, and help me to understand how interesting to dive into computer science research and how wonderful an academic life is. All these factors allow me to nourish myself as a successful researcher. Their guidance and motivation have a profound positive impact throughout my Ph.D. Career. Ph.D. is not only about mastering a specific problem or research domain but also about learning many valuable life lessons, such as time management, learning how to learn, communicating with people etc. Under their supervision, I have learned the methodology to carry out the research and the essential real-life skills mentioned above. During the job-hunting period, numerous recommendation letters from my supervisors (along with other recommenders) helps me tremendously to achieve my first academic position.

My sincerest appreciation goes to my other dissertation committee members: Rickard Ewetz, Fan Yao, and Yanjie Fu. Thanks to all of them for agreeing to serve on my dissertation committee and for their insightful comments on my dissertation. I am also thankful to my research collaborators: Samsil Arefin, Aamir Khan, Sai Sruti, Kecheng Yang, Di Liu, Nan Guan, Federico Reghenzani, William Fornaciari, and Hayoi Xiong. Their fruitful discussion and collaboration help to produce several impactful works.

My six years long graduate life was indeed a pleasant experience, thanks to all members of the real-time and intelligent systems (RTIS) research group. I am lucky to have you as my labmates and co-workers. My special thanks go to Samsil Arefin, Sai Sruti, Aamir khan, Sudharsan Vaidhun, and Abdullah Arafat. I am indebted to you for your collaboration, feedback, suggestions in the presentation, job talks, research paper writing. You also helped a lot build a friendly, professional, and collaborative environment in the lab.

Finally, I am indebted to my parents for their unconditional love, sacrifice, and inspiration. They have made every effort to prepare me for a better future. To my father, I wish you could stay with us till today to see this achievement: thank you for all your effort throughout your life. To my mother, no matter how far I go, I will never forget you taught me how to make the first step. To my wife, Fayrose Sayeda, thank you for your constant support, trust, and patience. Thank you for your love and belief during many frustrating and challenging times in my life. Thanks to my brother and sister-in-law for their supports towards the completion of my degree and for taking care of our family, which keeps me relaxed and tension-free. Finally, thanks to my only niece, Rufaida, who makes me feel that life is more beautiful than I thought.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1: INTRODUCTION

A real-time system application requires strict timing guarantees, energy efficiency, and high performance. Besides the high-performance requirement and *temporal* constraints, a real-time system must guarantee the *logical* constraints also, i.e., in response to the event generated by the surrounding environment, the system must react within precise time constraints. In modern society, an increasing number of complex systems depend on computer control, and real-time systems are widely deployed in these areas. In such a system, the computing platforms need to satisfy the time preciseness, e.g., chemical and nuclear plant control, automotive applications, flight control systems, smart grid, telecommunication systems, robotics, space missions, and many more [34].

Many embedded system applications have hard real-time constraints, and such an application must preserve precise timing correctness. Many real-time embedded systems include critical applications requiring not only a predictable timing behavior, but also to satisfy other system constraints. Energy consumption is one of them, which might be a non-functional or functional requirement. Embedded systems applications rely on unreliable energy sources, e.g., batteries, energy harvesters, making it essential to consider the energy-efficient design [108]. Energy-efficient design of an embedded system is a prime requirement for many reasons, such as increased battery life, reducing power bills, controlling heat dissipation etc.

Multi-core processors are increasingly receiving attention as an enabling platform for embedded system applications. In a multi-core platform, multiple tasks can execute simultaneously. However, an individual task can not execute on multiple cores simultaneously. Therefore, such a task model, i.e., the *sequential task model* fails to utilize the benefits of a multi-core platform. The *parallel task model* is introduced to tackle the problem mentioned above, where an individual task can simultaneously utilize multiple cores. Many computation-intensive systems (e.g., self-driving cars) that demand stringent timing requirements often evolve in the form of parallel tasks. Also, energy-efficient scheduling of parallel task model is a promising direction to utilize the multi-core platform's computational power. This model promotes a balanced distribution and simultaneous execution of the tasks among the computing platforms, which lead to energy efficiency and reduced computation. The directed acyclic graph (DAG) model is considered as one of the most generalized deterministic workload models to represent intra-task parallelism [17], where nodes represent threads of execution and edges represent their dependencies.

1.1 Challenges in Energy-Aware Real-Time Scheduling

Many recent studies on real-time scheduling and analysis have focused on the DAG task model [102, 17, 81, 30, 109, 82, 16]. However, to date, only a little work has been done for energy-aware real-time scheduling of the DAG task models. In general, minimizing energy/power consumption of a real-time system is challenging for the following reasons:

- In real-time system, a task model is often characterized by the task arrival pattern, the deadline, and the execution time. The relation ship between platform (where the task is running) execution speed, energy consumption, and execution time of each task is non-linear.
- Some approaches proposed minimizing system energy consumption via per-core dynamic voltage and frequency scaling (DVFS). Unfortunately, per-core DVFS becomes inefficient as it increases the hardware cost [70].
- Existing policies (for energy efficiency) assumed that a task would execute up to its Worst-Case Execution Time (WCET), which often differs from reality. A task rarely executes up to its WCET [111]. For several real-life applications, WCET based schedulability analysis is proved to be very pessimistic [84], mainly because of the significant variability be-

tween actual execution requirements and their WCET. Such a WCET-based energy-efficient approach may lead to system over-provisioning, low utilization, high costs, and excessive power/energy consumption [25, 88].

- Mixed-Criticality (MC) framework [122] was proposed to efficiently utilize the non-negligible gap between the WCET and the actual execution time and to minimize energy consumption. Different software components with varying levels of criticality are integrated into a common platform in an MC setup. However, energy-efficient MC scheduling is still immature as it may entirely discard the low-criticality tasks when the system switches to a high-criticality mode [26, 25].

- Although the MC platform allows integrating different software components with varying levels of criticality into a common platform, further improvement in the resource management is possible, as the state of the arts MC scheduling approaches still rely on WCET based estimation at each criticality-levels [25].

1.2 Thesis Statement

Many real-time embedded systems include critical applications requiring timing constraints and other system requirements, such as energy consumption. This thesis aims to answer whether we can integrate the existing energy-aware approaches to the intra-task parallelism so that the overall energy consumption is minimized while maintaining the temporal correctness of a real-time system application. This thesis presents the new energy-aware scheduling techniques, verifies their correctness, and reports empirical evaluation, supporting the following thesis statement:

By appropriately leveraging the energy-saving techniques with the internal parallelism of real-time embedded system applications, it is possible to significantly reduce the *system energy consumption without violating the strict timing requirement imposed by the real-time system applications.*

1.3 Thesis Contribution and Organization

In this thesis, we tackle the above challenges and make a series of contributions to the theory and system for energy-aware real-time scheduling of parallel task model, as mentioned below:

First, we propose an energy-efficient task scheduling approach considering the DAG task model. Considering a multiprocessor platform, we propose an energy sub-optimal federated scheduling algorithm for sporadic DAG tasks with implicit deadlines (Chapter 3). Based on the solution under federated scheduling, we also present a greedy intra-task processor merging technique which improves the power efficiency. We further improve the processor intra-merge technique by allowing multiple processors to merge at a single one. Besides, we also present the inter-task processor technique (Chapter 4).

Second, we propose a novel technique for energy-efficient scheduling of both the constrained and implicit deadline parallel tasks in a cluster-based homogeneous multi-core system (Chapter 5). As stated earlier, the per-core DVFS technique is becoming inefficient because of its increased hardware cost. The cluster-based platform (processors in the same island execute at the same speed) seems a promising platform to balance energy efficiency and hardware cost. However, the cluster-based platform introduces several challenges as well. The existing solutions were deduced considering the per-core DVFS is no longer valid in a clustered platform. Also, the execution speed of a cluster becomes unpredictable if multiple tasks share it with sporadic release patterns. We introduce a new concept of *speed-profile* that models per-task and per-cluster energy-consumption variations during run-time to minimize the expected long-term energy consumption. Later, we extend our technique to adapt the more realistic features such as discrete processor frequency scheme and the platform heterogeneity (Chapter 6).

Third, we incorporate the MC context into other well-known parallel task models, i.e., the gang task model (Chapter 7). To schedule such task sets, we propose a new technique GEDF-VD, which integrates Global Earliest Deadline First (GEDF) and Earliest Deadline First with Virtual Deadline (EDF-VD). We also derive the first speedup bound (a widely accepted tool for evaluating the effectiveness of multiprocessor scheduling algorithms) for GEDF schedulability of non-MC gang tasks and further derived the bound for GEDF-VD of MC gang tasks.

Finally, we show the response time analysis of MC task so that the probabilistic technique described in [25] minimizes the average energy consumption while guaranteeing the (worst-case) timing correctness for all tasks, under any execution condition (Chapter 8).

CHAPTER 2: PRELIMINARIES AND NOTATION

This chapter introduces a widely used representative of a real-time parallel task model, power model and some of the major notations used throughout this thesis.

2.1 Real-Time Parallel Task Model

Now, we describe two different types of parallel tasks considered in this thesis. There are multiple parallel languages and libraries, e.g., Intel Cilk Plus [73], OpenMP [96], etc., to write parallel programs. The directed acyclic graphs (DAG) task model and the gang task model is often used to model these programs.

2.1.1 DAG Task Model

The set of DAG task is denoted by denoted by $\tau = \{\tau_1, \cdots, \tau_n\}$, where each $\tau_i \in \tau$ $(1 \leq i \leq n)$ is represented as a DAG with a minimum inter-arrival separation (i.e., period) of T_i time units, and a relative deadline of $D_i (\leq T_i)$ time units. An implicit deadline task has the same relative deadline and period, i.e., $D_i = T_i$, while for a *constrained deadline* task, the relative deadline D_i is less than T_i. The *nodes* in a DAG stand for different execution requirements while the *edges* represent the dependencies among the corresponding execution requirements. A parallel task τ_i contains a total number of N_i nodes, each denoted by $\mathcal{N}_i^j (1 \leq j \leq N_i)$. A directed edge from $\mathcal{N}_i^j$ to $\mathcal{N}_i^k (\mathcal{N}_i^j \rightarrow \mathcal{N}_i^k)$ implies that execution of $\mathcal{N}_i^k$ can start if $\mathcal{N}_i^j$ finishes for every instance

The contents of this chapter have been previously published at and available at:
1. Bhuiyan, A., Guo, Z., Saifullah, A., Guan, N., & Xiong, H. (2018). Energy-efficient real-time scheduling of DAG tasks. ACM Transactions on Embedded Computing Systems (TECS), 17(5), 1-25.
2. Bhuiyan, A., Yang, K., Arefin, S., Saifullah, A., Guan, N., & Guo, Z. (2021). Mixed-criticality real-time scheduling of gang task systems. Real-Time Systems, 1-34.

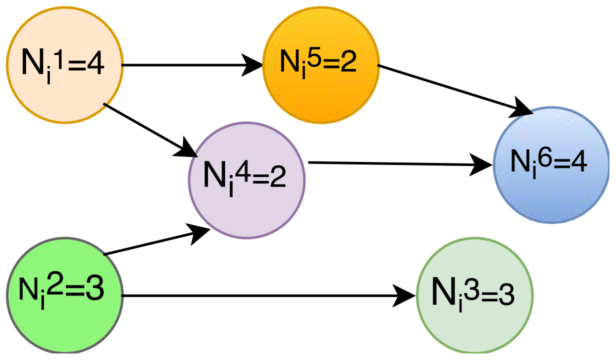


Figure 2.1: An example heavy DAG task τ_i.

(precedence constraints). In this case, $\mathcal{N}_i^j$ is called a parent of $\mathcal{N}_i^k$ ($\mathcal{N}_i^k$ is a child of $\mathcal{N}_i^j$). A node may have multiple parents or children. The *degree of parallelism*, $\mathcal{M}_i$, of τ_i is the number of nodes that can be simultaneously executed. c_i^j denotes the execution requirement of node $\mathcal{N}_i^j$. $C_i := \sum_{j=1}^{N_i} c_i^j$ denotes the worst case execution requirement (WCET) of τ_i.

A *critical path* is a directed path with the maximum total execution requirements among all other paths in a DAG. L_i is the sum of the execution requirements of all the nodes that lie on a critical path. It is the minimum make-span of τ_i, i.e., in order to make τ_i schedulable, at least L_i time units are required even when number of cores is unlimited. Since at least L_i time units are required for τ_i, the condition $T_i \geq L_i$ (implicit deadline tasks) and $D_i \geq L_i$ (constrained deadline tasks) must hold for τ_i to be schedulable. A schedule is said to be feasible if upon satisfying the precedence constraints, all the sub-tasks (nodes) receive enough execution from their arrival times, i.e., C_i within T_i (implicit deadline) or D_i (constrained deadline) time units.

A DAG is *heavy* if its execution requirement is greater than its period (i.e. $C_i > T_i$). A schedule is said to be *feasible* when all sub-tasks (nodes) receive enough execution (up to their execution requirements) within T_i time units from their arrivals, while all precedence constraints are satisfied. The aforementioned terms are illustrated with the help of Figure 2.1 and Example 1.

Example 1. *The DAG task τ_i (shown in Figure 2.1) is a heavy DAG with total execution time $C_i = 18$ and minimum inter-arrival separation $T_i = 12$. It has a critical path length of 10 ($\mathcal{N}_i^1 \rightarrow \mathcal{N}_i^4 \rightarrow \mathcal{N}_i^6$ or $\mathcal{N}_i^1 \rightarrow \mathcal{N}_i^5 \rightarrow \mathcal{N}_i^6$). As the critical path length L_i is less than the task period T_i, this task may meet its deadline provided enough processors.*

2.1.2 Gang Task Model

In traditional gang task model, each task τ_i is represented with a 4-tuple (m_i, c_i, T_i, D_i), where m_i is the degree of parallelism and each job of task τ_i requires access to m_i cores for at most c_i time units to complete its execution, T_i is the task period, and D_i is the relative deadline. In gang scheduling, each task is consists of multiple threads (referred to as a gang), and each thread of the same task occupies a processor for the same time quantum [76, 46]. Hence, in the time-space, the execution requirement of any job $\tau_{i,j} \in \tau_i$ can be represented as an $m_i \times c_i$ rectangle. The relative deadline D_i specifies that for each of the released jobs $\tau_{i,j}$ (of task τ_i), its deadline $d_{i,j} = r_{i,j} + D_i$, where $r_{i,j}$ denotes the release time of $\tau_{i,j}$ [46].

The *utilization* u_i of each task $\tau_i \in \tau$ is given by $u_i = (m_i c_i)/T_i$, and the overall system utilization is: $U_{sum} = \sum_{\tau_i \in \tau} u_i$. Note that, it is possible that the value of u_i is larger than one, which is different from the traditional sequential task model. Based on the scheduling flexibility, a gang task τ_i can be categorized into three groups. A task τ_i is said to be:

- *rigid*, if m_i is fixed a priori and does not change throughout the execution,
- *moldable*, if m_i is fixed during its activation and does not change throughout the execution,
- *malleable*, if m_i is not fixed and can be changed during its execution by the scheduler.

In this thesis, we restrict our attention to the *rigid* task model. This model suits various applications that use parallelism, some of which are implemented using the message-passing approach and tools like MPI.

2.2 Power/Energy Model

Let $s(t)$ (we are assuming continuous frequency scheme) denote the main frequency (speed) of a processor at a certain time t. Then its power consumption $P(s)$ can be modeled as:

$$P(s) = P_{sta} + P_{dyn}(s) = \beta + \alpha s^{\gamma}, \tag{2.1}$$

where P_{sta} denotes the static power consumption which is introduced in the system due to the leakage current and $P_{dyn}(s)$ denotes the active power consumption. $P_{dyn}(s)$ is introduced due to the switching activities and it depends on the processor frequency. $P_{dyn}(s)$ can be represented as αs^{γ} where the constant $\alpha > 0$ depends on the effective switching capacitance[98], $\gamma \in [2, 3]$ is a fixed parameter determined by the hardware, and $\beta > 0$ represents the leakage power (i.e., the static part of power consumption whenever a processor remains on). Clearly, the power consumption function is a convex-increasing function of the processor frequency. By means of dynamic voltage and frequency scaling (DVFS), it is possible to reduce $P_{dyn}(s)$ by reducing the processor frequency. In this paper, we focus on minimizing the active energy consumption (due to $P_{dyn}(s)$) by means of DVFS. We also target to minimize the static power consumption (due to P_{sta}) by reducing the number of processors by allowing intra-task processor sharing.

Note that static power consumption is partially related to frequency. It is mainly because highest frequency (that can be used) can be margined by the voltage level which is inversely proportional to the circuit delay. Such relationship is counted towards the dynamic part of the power consumption by adopting a slightly larger γ value. Various platforms may take different values, yet the proposed approach should work in general. Besides, we assume that each core can execute with any frequency ranged between f_{min} to f_{max}, where, $f_{min} \geq$ critical frequencies[1] [98]. Within this allowable frequency range [$f_{min} - f_{max}$] we consider the upper bound of static energy consumption. Hence we can consider that static energy consumption is fixed w.r.t. the frequency.

Our motivation behind selecting this power model comes from the fact that it complies with many existing works in the community, few to mention [7, 98, 99, 72, 92, 67, 23]. Beside this, recently this model was shown to be highly realistic by showing its similarity with actual power consumption [99]. Figure 2.2 (adopted from [98]) shows comparison between the original power consumption results from [71] and the power model in Equation (2.1), where we consider $\alpha = 1.76 Watts/GHz^3$, $\gamma = 3$, and $\beta = 0.5$ Watts.

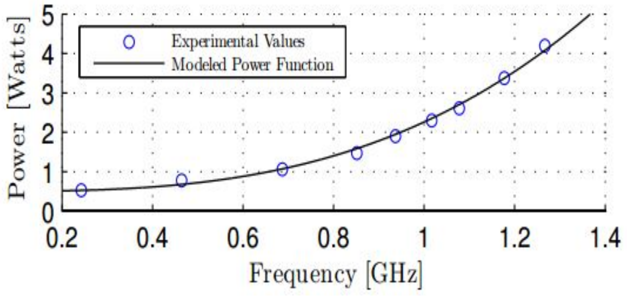


Figure 2.2: Comparison of the power model (Equation (2.1)) with experimental results in [71].

The energy consumption of any given period $[b, f]$ can be calculated as $E(b, f) = \int_b^f P(s(t))\, dt$, which is known as a nice approximation to the actual energy consumption of many known systems. Specifically, given a unit-amount of workload to be executed on a speed-s processor, the total energy consumption is the integral of power over the period of length $1/s$; i.e.,

$$E(s) = (\beta + \alpha s^{\gamma})/s = \beta/s + \alpha s^{\gamma-1} \tag{2.2}$$

[1]Optimal frequency to reduce the energy consumption (considering a negligible sleeping overhead).

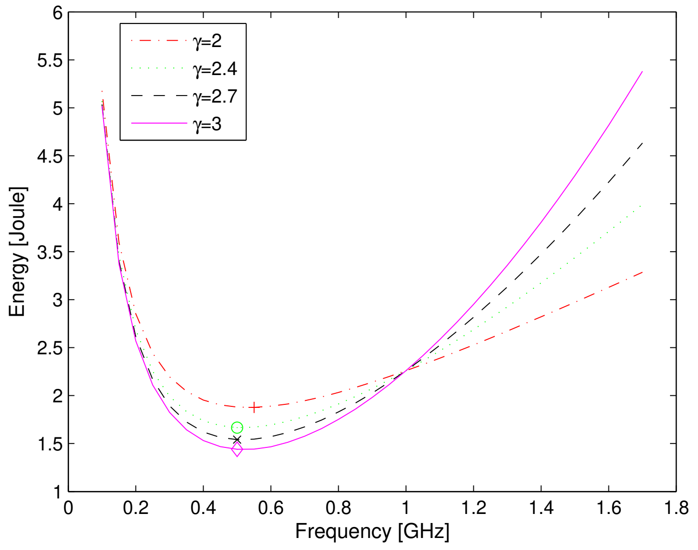


Figure 2.3: Energy consumption for executing a job with 10^9 computation cycles with a fixed α, β and a variable γ value.

Figure 2.3 shows how different values of γ (varying from 2 to 3) and processor speed s may affect the total energy consumption to complete a certain (fixed) amount of computation. This figure reports the energy consumption for executing a job with 10^9 computation cycles under various γ values, where $\alpha = 1.76$ watts/GHz, and $\beta = 0.5$ watts. Energy Consumption is calculated according to Equation (2.2), and the lowest points represent the most energy efficient execution frequency,i.e., *critical frequency*. It is obvious that execution under a speed much lower than the critical frequencies is extremely energy inefficient (as leakage power becomes the major "contributior"). The power model we adapted complies with much existing (and recent) work in the community, e.g., [7] [125] [44] [98] [99] [67].

CHAPTER 3: ENERGY-EFFICIENT FEDERATED SCHEDULING OF REAL-TIME DAG TASKS WITH INTRA-TASK PROCESSOR MERGING

This chapter studies energy-aware real-time scheduling of a set of sporadic Directed Acyclic Graph (DAG) tasks with implicit deadlines. While meeting all real-time constraints, we try to identify the best task allocation and execution pattern such that the average power consumption of the whole platform is minimized. This work addresses the power consumption issue in scheduling multiple DAG tasks on multi-cores and allows intra-task processor sharing. We adapt the decomposition-based framework for federated scheduling and propose an energy-sub-optimal scheduler. Then, we derive an approximation algorithm to identify processors to be merged together for further improvements in energy-efficiency. The effectiveness of the proposed approach is evaluated both theoretically via approximation ratio bounds, and also experimentally through simulation study. Experimental results on randomly generated workloads show that our algorithms achieve an energy saving of 60% to 68% compared to existing DAG task schedulers.

3.1 Introduction

Due to size and weight limitations in embedded systems, energy consumption is a cornerstone in their design, especially for battery-operated ones. Energy-efficient and power-aware computing therefore are gaining increasing attention in the embedded systems research. It is important due to the market demand of increased battery life for portable devices. Moreover, reducing energy consumption could lead to smaller power bills. Being motivated by this goal, there has been a trend in embedded system design and development towards multi-core platforms. In order to better uti-

This chapter has been previously published at ACM Transactions on Embedded Computing Systems and available at Bhuiyan, A., Guo, Z., Saifullah, A., Guan, N., & Xiong, H. (2018). Energy-efficient real-time scheduling of DAG tasks. ACM Transactions on Embedded Computing Systems (TECS), 17(5), 1-25.

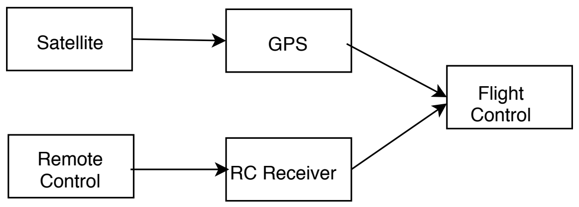


Figure 3.1: Partial work-flow of a UAV system represented as a DAG.

lize the capacity of multi-core platforms, parallel computation (where an individual task executes in multiple processors simultaneously) needs to be considered. For Example, a recent study [98] has shown that the energy consumption of executing certain workload perfectly distributed in two cores is significantly less than that of executing the same workload in one core at double frequency.

In this chapter, we deal with workloads that are represented as directed acyclic graph (DAG) — that are considered to be one of the most representative models of deterministic parallel tasks [17]. Several real-world application uses the DAG model ([68, 102, 74]). For example, consider the application shown in Figure 3.1. In this figure, we present the partial work-flow of an *unmanned aerial vehicle (UAV)* which is adopted from [112]. Here, each rectangular box denotes a sub-task (or node in the DAG) with their execution requirements (not shown in the figure) and edges represent data dependencies among them. As described above, a node cannot start execution until all of its parents are completed, and some of the nodes can be executed in parallel, e.g. *Satellite* and *Remote Control*. There are many existing works that focused on real-time scheduling of parallel tasks or their schedulability analysis [17] [30] [81][10] [82]. However, none of them addressed the energy consumption issue.

Energy-Aware Real-Time Scheduling. In the design of embedded systems, energy minimization is a prime requirement. Much work has been done on minimizing the energy cost with respect to sequential tasks for multi-core systems [55] [99] [98] [92]. Specifically, [98] and [99] present an energy efficient task partitioning scheme, where the cores are grouped in frequency islands. The authors in [7] considers both feasibility and energy-awareness while partitioning periodic real-time tasks on a multi-core platform. For EDF scheduling, they show that if the workload is balanced evenly among the processors, deriving optimal energy consumption and finding a feasible partition is NP-Hard. Till date, only a little work has been done for energy-aware real-time scheduling of parallel tasks. In general, minimizing energy/power consumption of a real-time system is challenging due to the complex (non-linear) relationship between frequency, energy consumption, and execution time of each task.

In this chapter, we study the scheduling of a set of sporadic DAG tasks with implicit deadlines on a multi-core platform. We assume that all the cores that are assigned to a DAG task will always remain active, which leads to a non-negligible power consumption. In order to reduce this effect, we allow intra- and inter-task processor sharing to remove or reduce the number of lightly loaded cores. After merging, the cores that are not required can be shut off completely. When the average case execution times are typically small compared to the worst-case execution time (WCET), the cores will remain idle (in that case the active power consumption will be minimized, refer to the Power model described at Chapter 2). Specifically, we make the following key contributions:

(i) We propose a power-consumption-aware scheduling algorithm for sporadic DAG tasks with implicit deadlines.

(ii) Under the federated scheduling and task decomposition framework, our table-driven scheduler is shown to be optimal in the sense of average power consumption (i.e., named sub-optimal due to extra constraints included).

(iii) We propose an efficient processor merging technique that is widely applicable for energy-efficiency improvements to most of the existing work on federated DAG task scheduling. We formally prove the NP-completeness of the problem, propose an approximation algorithm, and prove the upper bound of its approximation ratio.

(iv) Based on randomly generated workload, simulations are conducted to verify the theoretical results and demonstrate the effectiveness of our algorithm.

The rest of this chapter is organized as follows. Section 3.2 discusses related work. Section 3.3 adapts the task decomposition scheme and proposes an (sub-) optimal federated scheduler based on segment extension and problem transformation (into a convex optimization with linear inequality constraints). Section 3.4 relaxes the federated limitation by presenting and analyzing techniques for intra-DAG and inter-DAG processor merging, so that energy consumption is further reduced. Section 3.5 implements gradient based solvers and compares the proposed method with state-of-the-art schedulers. Section 3.6 concludes the chapter.

3.2 Related Work

The work that deals with schedulability tests for various scheduling policies on parallel task model is already mentioned in Section 3.1. None of them has considered power/energy consumption issues. In addition, much work has been done in energy/power consumption minimization for sequential tasks. Bini et al. discuss the problem of finding an optimal solution for a system with discrete speed levels for a set of periodic/sporadic tasks [28]. They have considered both EDF and Fixed-Priority (FP) scheduling policies. Jejurikar has considered non-preemptive tasks in order to deal with shared resources [75]. Chen et al.and Liu et al. presents an energy-efficient design for heterogeneous multiprocessor platform in [39] and [85] respectively. No previous work considers parallel task model.

For parallel task models, several results are obtained on schedulability tests under various scheduling policies in [17] [30] [10]. [30] prove a speedup bound of $(2 - 1/m)$ for Earliest Deadline First (EDF) and $(3 - 1/m)$ for Deadline Monotonic (DM) respectively, where m is the number of processors. For global EDF scheduling, these techniques are further generalized [10] with an improved pseudo-polynomial time sufficient schedulability test. Analysis of federated and global EDF scheduling is performed in [81] [82]. Processor-speed augmentation bounds for both preemptive and non-preemptive real-time scheduling on multi-core processors are derived in [109]. The work in [16] studies global EDF scheduling for conditional sporadic DAG tasks, which is an extension to the normal sporadic DAG task model. Certain conditional control-flow constructs (such as *if-then-else* constructs) can be modeled using the conditional sporadic DAG task model. Despite those nice preliminary work on the schedulability analysis of parallel tasks, none of them addresses the energy/power consumption issue.

Actually, intra-task parallelization and power consumption issues have not yet received sufficient attention. Zhu et al. have considered power-aware scheduling for graph-tasks [129]. The work in [130] proposed the greedy slack stealing algorithm that is able to deal with the task represented by AND/OR graphs. It proves the correctness of the proposed algorithm in terms of meeting the applications time constraint considering it is executing on an N-processor system. Through simulation, it has also analyzed the performance of the algorithm in terms of processor energy saving and showed that the GSS is able to achieve some energy efficiency. However, that work considered the scheduling of only a single DAG and the DAG was not periodic/recurrent. For dependent tasks, [37] provides techniques that combine dynamic voltage and frequency scaling (DVFS) and dynamic power management, where each core in the platform can be switched on and off individually. For block-partitioned multi-core processors (where cores are grouped into blocks and each block has a common power supply scaled by DVFS), energy efficiency is investigated in [103]. The authors in [101] consider power-aware policy for scheduling parallel hard real-time systems, where the multi-thread processing is used. [100] considers dealing with parallel tasks under Gang scheduling policy, where all parallel instances of a task use a processor in the same window. The authors in [124] have considered energy minimization for frame-based tasks (i.e., same arrival time and a common deadline for all the tasks) with implicit deadlines. Similar frame based model is considered in [64], where precedence constraints can be specified among the tasks. As mentioned previously, no existing work allows intra and inter-task processor sharing when considering the (more general) sporadic DAG task workload model.

3.3 Energy-Sub-Optimal Federated Scheduling for DAG Tasks

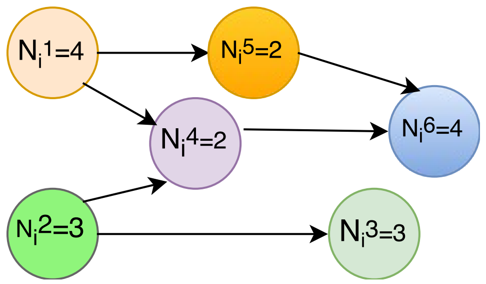
In this section, we restrict our focus on the federated scheduling of DAG tasks, refer to Section 2.1.1 for details regarding the DAG model. Under the federated approach to multi-core scheduling, each individual task is either restricted to execute on a single processor (as in partitioned scheduling), or has exclusive access to all the processors on which it may execute. Since each processor is dedicated to one DAG task, we can consider each task individually, and try to minimize the energy consumption for a single DAG task (which is the goal of this section).

Given a DAG task, we first apply the existing task decomposition [109] technique to transform a parallel task into a set of sequential tasks with scheduling window (for a specific node it denotes the time frame from its release offset to its deadline) constraints for each node (Subsection 3.3.1) – they are further relaxed into necessary conditions by segment extension (Subsection 3.3.2). By variable substitution, we then transform the energy minimization problem into a convex optimization problem with linear inequality constraints, which can be solved *optimally* with gradient-based methods (Subsection 3.3.3).

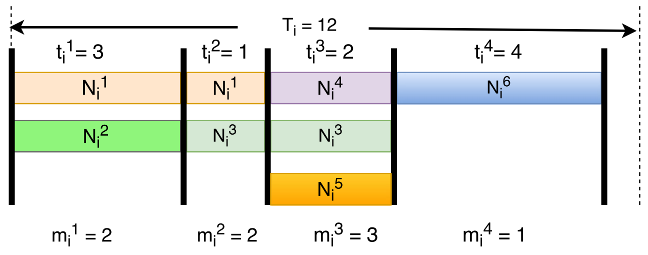
3.3.1 Task Decomposition

Task decomposition is a well-known technique that simplifies the scheduling analysis of parallel real-time tasks [109]. In our approach, we adopt task decomposition as the first step that converts each node $\mathcal{N}_i^l$ of the DAG task τ_i to an individual sub-task τ_i^l with a release offset (b_i^l), deadline (f_i^l), and execution requirement (c_i^l). The release time and deadlines are assigned in a way that all dependencies (represented by edges in the DAG) are respected. Thus the decomposition ensures that if all the sub-tasks are schedulable then the DAG is also schedulable. For the sake of completeness, we briefly describe how task decomposition works in this subsection with an example.

We adapt a slightly modified version of the approach used in [109]. First, we perform the task decomposition using the techniques in [109] as described below. Assuming the execution of the task is on an unlimited number of cores, we draw a vertical line at every time instant where a node starts or ends for each node starting from the beginning. These vertical lines split the DAG into segments, and each segment consists of an equal amount of execution by the nodes that lie in the segment. Parts of different nodes in the same segment can now be considered as threads of execution that run in parallel, and the threads in a segment can start only after those in the preceding segment have finished their executions. Now we will say that the resulting segmented structure of the task is converted into synchronous form and will denote it as τ_i^{syn}. We first allocate time to the segments and then add all times assigned to different segments of a node to calculate its allocated time.



(a) A DAG task, τ_i.



(b) Scheduling window constraints for all nodes in τ_i after task decomposition.

Figure 3.2: A DAG task and its equivalent decomposed structure.

Since the minimum makespan, $L_i \leq T_i$, at the end of each period, there may be a slack where all processors are idle (which is typically energy inefficient). We allocate such idle period *uniformly* by multiplying each segment by a common factor of T_i/L_i for task τ_i. Task decomposition provides its processor assignment $\mathcal{M}_i^l$ (i.e., a node-to-processor mapping) and a scheduling window $[b_i^l, f_i^l)$ on top of it, in which each node $\mathcal{N}_i^l$ of a task τ_i will be scheduled. In Example 2 and Example 3, we demonstrate the concept of task decomposition and scheduling window.

Example 2. *Consider task τ_i shown in Figure 3.2(a). First, we assign all the nodes with no parent ($\mathcal{N}_i^1$ and $\mathcal{N}_i^2$) to separate processors. Then we continue to consider nodes only when all its parent node(s) are assigned. As a result, the beginning of the node will be the latest finishing time of its parent(s) — these are boundaries of the segments, denoted by vertical lines in Figure 3.2(b). Specifically, if a node has a single parent, we can start to consider the node right after the finishing time of its parent. For example, when $\mathcal{N}_i^2$ is completed, $\mathcal{N}_i^3$ is immediately assigned to the same processor (as $\mathcal{N}_i^2$ is the only parent).*

When a node has multiple parents, we consider the parent that has the latest finishing time. The child node may be assigned to the same processor assigned to its parent with the latest finishing time. For example, $\mathcal{N}_i^4$ has two parents $\mathcal{N}_i^1$ and $\mathcal{N}_i^2$ where $\mathcal{N}_i^1$ completes execution later. So $\mathcal{N}_i^4$ is assigned to the same processor of $\mathcal{N}_i^1$. Please note that a node may have multiple siblings such that it may not always share the same processor with its latest finished parent node — under such scenario, a new processor is allocated to the node. For example, the only parent of $\mathcal{N}_i^5$ is $\mathcal{N}_i^1$ which completes execution at t_i^2. So $\mathcal{N}_i^5$ would be able to execute in the same processor starting from the third segment. But $\mathcal{N}_i^5$ is assigned to a different processor as that specific processor at t_i^3 is already "taken" by its sibling $\mathcal{N}_i^4$.

Example 3. *Let, m_i^k denotes the degrees of parallelism at k^{th} segment and the node-core mapping is: $\bar{\mathcal{M}}_i = \{1, 2, 2, 1, 3, 1\}$. Scheduling windows for these nodes (from Figure 2.1) are as follows: $\mathcal{N}_i^1 = [1, 2]$, $\mathcal{N}_i^2 = [1, 1]$, $\mathcal{N}_i^3 = [2, 3]$, $\mathcal{N}_i^4 = [3, 3]$, $\mathcal{N}_i^5 = [3, 3]$, $\mathcal{N}_i^6 = [4, 4]$. In this example,*

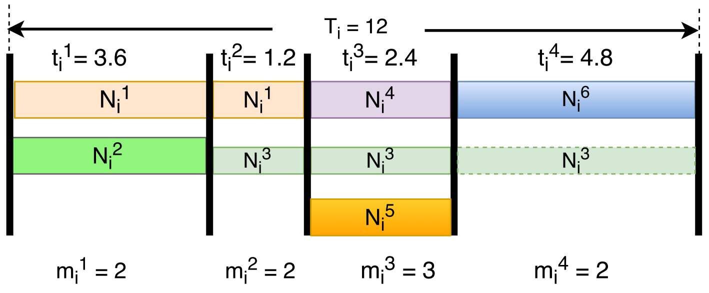


Figure 3.3: The segment-node mapping for τ_i (from Figure 2.1) after segment extension.

the average power consumption (under such settings) is 3.33 *Watts after extending each segment by a common factor of* $T_i/L_i = 1.2$.

3.3.2 Segment Extension

For a DAG task τ_i, the aforementioned task decomposition results in a mapping between a node ($\mathcal{N}_i^l$) and a processor ($\mathcal{M}_i^l$). One of the key issues with the task decomposition process is that the identified scheduling window constraints for the nodes may not be necessary. Take the task described in Figure 3.2 as an example, where Node $\mathcal{N}_i^3$ may execute in the 4th segment. However, task decomposition requires that Node $\mathcal{N}_i^3$ must finish by the end of Segment 3, which is unnecessary. In this subsection, we describe a systematic way of eliminating such unnecessities so that the boundary constraints for all nodes (b_i^l's and f_i^l's) are both *necessary* and *sufficient*.

Each DAG τ_i is first converted to a synchronous form denoted by τ_i^{syn} with techniques described in Section 3.3.1. We use m_i^k to denote the number of parallel threads in the k-th segment of τ_i^{syn}. We then apply Algorithm 1 to greedily extend the deadlines f_i^l of each node $\mathcal{N}_i^l$, following any topological order. Note that while performing task decomposition, a node starts execution immediately when all of its predecessors finish execution. Thus the starting time b_i^l cannot be moved earlier — only f_i's have room to be relaxed.

Task decomposition technique determines scheduling window constraints for the nodes which are *sufficient* but may not be *necessary*. If segment extension is performed (using Algorithm 1) after applying task decomposition, scheduling window constraints become *necessary* and *sufficient* (see Lemma 1). It may happen that for some particular DAG structure, Algorithm 1 fails to change (i.e. expand) scheduling window for any node. However, it does not impact the schedulability of the DAG. From this discussion, it should be clear that performing segment is not mandatory but it is done to further reduce the energy consumption.

Note that we have considered table-driven schedulers which usually pre-compute which task would run when. This schedule is stored in a table at the time the system is designed. When a set of n tasks is to be scheduled, then the entries in the table will replicate themselves after LCM ($T_1, T_2 \cdots T_n$), where LCM ($T_1, T_2 \cdots T_n$) is the hyper-period for the tasks. However, while considering energy consumption we did not consider the space complexity of the scheduling solutions.

Algorithm 1: Segment Extension

```
1: **Input:** A DAG $\tau_i$, scheduling windows after decomposition $[b_i^l, f_i^l]$ for any node $\mathcal{N}_i^l \in \tau_i$.
2: **Output:** Extended segment window $[b_i^l, f_i^l)$ for each node $\mathcal{N}_i^l \in \tau_i$.
3: Assume that all nodes $\mathcal{N}_i^l$ are ordered topologically, such that predecessor constraint may
   only occur between $\mathcal{N}_i^l \longrightarrow \mathcal{N}_i^{l'}$ when $l < l'$.
4: **for** each node $\mathcal{N}_i^l \in \tau_i$ **do**
5:    **if** node $\mathcal{N}_i^l$ has successor node(s); i.e., $\exists l', \mathcal{N}_i^l \longrightarrow \mathcal{N}_i^{l'}$
6:    **then** $f_i^l \leftarrow \min_{l' | \mathcal{N}_i^l \longrightarrow \mathcal{N}_i^{l'}} \{b_i^{l'}\} - 1$;
7:    **else** $f_i^l \leftarrow$ last segment of $\tau_i^{syn}$;
8: **end for**
9: **return** $[b_i^l, f_i^l]$ for each node $\mathcal{N}_i^l$.
```

Example 4. *Consider again the DAG task τ_i shown in Figure 2.1. Algorithm 1 greedily extends the ending segment f_i^l of the nodes as much as possible in the topological order (i.e., increasing l). After applying Algorithm 1, scheduling windows for all the nodes changes as follows:* $[1, 2], [1, 1], [2, \mathbf{4}], [3, 3], [3, 3], [4, 4]$, *i.e., Node $\mathcal{N}_i^3$ can now execute in Segment 4 (dashed rectan-*

gle at Figure 3.3) and the execution window for all the other nodes remain unchanged. Note that the processor assignment $\mathcal{M}_i^l$ for any node $\mathcal{N}_i^l$ of a task τ_i remains unchanged in the segment extension process. Such an extension results in an average power consumption of 3.08 Watts.

Lemma 1. *Under the task decomposition and scheduling framework, after running Algorithm 1, the timing constraints we set for each node in a DAG become* necessary *and* sufficient*.*

Proof. First, we show that if task decomposition is considered the timing constraints set for each node in a DAG is only sufficient. Then we prove that if the segment extension is applied after employing the task decomposition, it makes the timing constraints necessary and sufficient.
Upon task decomposition, calculated scheduling window for each node satisfies all predecessor constraints, without changing the deadline for the DAG and it completes the proof of sufficient part. Now we prove that the timing constraints are unnecessary by proving a simple example. Consider the task described in Figure 3.2 again. Task decomposition requires that Node $\mathcal{N}_i^3$ must finish by the end of Segment 3, which is unnecessary because it may execute in the 4^{th} segment.

Now consider both task decomposition and segment extension are applied to a DAG. In that case, the sufficient part is trivial. The scheduling window satisfies all predecessor constraints, while the deadline for the DAG task does not change.

Assume the window after modification $[b_i^l, f_i^l]$ for some node $\mathcal{N}_i^l$ is not necessary; i.e., it can be further extended. Then it must be one of the following two cases:

- An earlier b_i^l still satisfies all predecessor constraints, which is impossible since it is the time all parents are finished.
- A later f_i^l is possible, which contradicts with Lines 5 - 7 of Algorithm 1 as it is already the starting point of its child, or the deadline of the whole DAG.

□

3.3.3 Problem Transformation

After task decomposition and segment extension, we have identified the scheduling window $[b_i^l, f_i^l]$ for each node $\mathcal{N}_i^l$, and there is no overlap for any two windows (for different nodes) on the same processor. A natural question arises: *Given a specific node (job) with a pre-determined scheduling window on a dedicated processor, what is the most energy-efficient execution (speed) pattern?*

Theorem 2. *The total energy consumption (assuming processor remains on) $\int_a^{a+\Delta} s(t)^\gamma \, dt$ is minimized in any scheduling window $[a, a+\Delta]$ of length Δ when execution speed remains the same; i.e., $s(t) \equiv C/\Delta$, where $C = \int_a^{a+\Delta} s(t) \, dt$ is the (given) task demand in the window.*

Proof. We define $p(t) = s(t)/C$, then $p(x)$ is a *probability density function (PDF)* over $[a, a+\Delta]$; i.e.,

$$\int_a^{a+\Delta} p(t) \, dt = 1. \tag{3.1}$$

As a result,

$$\begin{aligned} \int_a^{a+\Delta} s(t)^\gamma \, dt &= \int_a^{a+\Delta} (C \cdot p(t))^\gamma \, dt \\ &\{\text{re-arranging}\} \\ &= \frac{C^\gamma}{\Delta^{\gamma-1}} \cdot \left(\frac{1}{\Delta} \int_a^{a+\Delta} (\Delta \cdot p(t))^\gamma \, dt \right) \qquad (3.2) \\ &\{\text{Jensen's Inequality [36], the convexity of function } x^\gamma \\ &\quad \text{when } 2 \le \gamma \le 3 \text{ and } x \ge 0, \text{ and } p(x) \text{ being a PDF}\} \end{aligned}$$

$$\geq \frac{C^\gamma}{\Delta^{\gamma-1}} \left(\int_a^{a+\Delta} p(t)\, dt \right)^\gamma$$

$\{$From (3.1)$\}$

$$= \frac{C^\gamma}{\Delta^{\gamma-1}} \tag{3.3}$$

$\{$Definition of integrating a constant function$\}$

$$= \int_a^{a+\Delta} \left(\frac{C}{\Delta} \right)^\gamma dt.$$

Thus, the minimal total energy consumption in the specified interval $\int_a^{a+\Delta} s(t)^\gamma\, dt$ can be achieved when speed $s(t)$ remains constant (C/Δ) throughout the interval $[a, a+\Delta]$. $\square$

According to Theorem 2, executing all segments with a uniform speed yields minimum possible power consumption under such framework. Hence we can assume that *the speed of any processor does not change within a segment.* Let S_j^k denote the speed of processor j in the k-th segment (executing node $\mathcal{N}_i^l$), and t_i^k denote the length of the segment. The objective is to determine the length of each segment $t_i^k (\geq 0)$ and its execution speed $S_j^k (\geq 0)$ such that total power consumption is minimized.

The first set of constraints guarantees the real-time correctness that each node $\mathcal{N}_i^l$ receives enough execution within its designated window $[b_i^l, f_i^l)$ on its assigned processor $\mathcal{M}_i^l$; i.e.,

$$\forall l, \mathcal{N}_i^l \in \tau_i : \sum_{k=b_i^l}^{f_i^l} t_i^k S_{\mathcal{M}_i^l}^k \geq c_{i,l}. \tag{3.4}$$

We need one more set of inequalities to bound the total length for all segments of each DAG by its period:

$$\forall i, \sum_{k} t_i^k \leq T_i. \tag{3.5}$$

Any non-negative speed assignment and segment length setting that satisfy the constraints described in (3.4) and (3.5) yield a *correct* schedule that all nodes receive enough execution in their specified scheduling windows (that satisfy all predecessor constraints). Based on these constraints, we would like to add our objective for minimizing average energy consumption per period:

$$\textbf{Minimize}_{\{t_i^k, S_j^k\}} \ M_i \beta T_i + \sum_{l=1}^{\zeta_i} \sum_{k=b_i^l}^{f_i^l} t_i^k \alpha (S_{\mathcal{M}_i^l}^k)^{\gamma},$$

where M_i is the degree of parallelism (and also the number of processors assigned to the task) and ζ_i is the total number of segments assigned to DAG task τ_i (determined in the previous step). Since the constraints represented in (3.4) are non-convex quadratic inequalities, it is in general computationally intractable to solve in polynomial time. We transform this problem into a convex optimization by substituting some variables.

Remark 1. *According to Theorem 2, executing all segments with a uniform speed yields minimum possible power consumption. If any segment of any core remains idle (scheduling window for any node does not fall at that segment), we consider that the execution speed for that segment is 0.*

Remark 2. *In this chapter, we are assuming that the time required to finish a task is exactly equaled to their WCET. However, it may happen that some of the tasks may finish early than their WCET. In that case, some of the cores (that are assigned to that tasks) may remain idle for some time. In this work, we did not consider that the cores can switch into a deep sleep state during the idle time. Entering into the deep sleep state costs additional energy consumption and it is not beneficial if the idle time slot is less than a certain threshold. By remaining idle we mean that one or more cores are active but not executing any task. It helps to reduce the active power consumption (i.e.* $P_d(s)$ *in*

Equation (2.1)). It would lead to the further minimization of the total power consumption (refer to the Power/Energy model described at Section 2). So our model actually provides the upper bound of the energy consumption.

Replacing speed with period lengths and executions. Fortunately, Theorem 2 provides us the basis to get rid of part of the variables. Since all nodes are executed at *constant* speeds within their scheduling windows, given the total length of each assigned segments (i.e., scheduling window), the execution speed of any given node can be determined. As a result, the energy consumption to finish this node can also be calculated. I.e., given a node $\mathcal{N}_i^l$ with total execution requirement of c_i^l, to be executed on segments between b_i^l and f_i^l, we have:

$$\forall k \in [b_i^l, f_i^l], S_{\mathcal{M}_i^l}^k = c_i^l / (\sum_{j=b_i^l}^{f_i^l} t_i^j), \tag{3.6}$$

which means although a node may be executed in consecutive segments $\forall k \in [b_i^l, f_i^l]$, the speed remains constant throughout the scheduling window and can be represented by a function of executions c_i^l and segment lengths t_i^j. Substituting Equation (3.6) into the second term of the objective function, we have:

$$\begin{aligned}
\sum_{l=1}^{\zeta_i} \sum_{k=b_i^l}^{f_i^l} t_i^k \alpha (S_{\mathcal{M}_i^l}^k)^\gamma &= \sum_{l=1}^{\zeta_i} \left(\sum_{k=b_i^l}^{f_i^l} t_i^k \alpha (c_i^l)^\gamma (\sum_{j=b_i^l}^{f_i^l} t_i^j)^{-\gamma} \right) \\
&\{\text{moving unrelated terms out of the summations}\} \\
&= \alpha \sum_{l=1}^{\zeta_i} \left((\sum_{j=b_i^l}^{f_i^l} t_i^j)^{-\gamma} (\sum_{k=b_i^l}^{f_i^l} t_i^k)(c_i^l)^\gamma \right)
\end{aligned} \tag{3.7}$$

$$\{\text{combining similar terms}\}$$

$$= \alpha \sum_{l|\mathcal{M}_i^l=j} c_l^\gamma (\sum_{k=b_i^l}^{f_i^l} t_i^k)^{1-\gamma}.$$

Thus, the original optimization problem can be equivalently transformed into the following one with only t_i^k as variables.

$$\begin{aligned} &\textbf{Minimize}_{\{t_i^k\}} \; M_i\beta T_i + \alpha \textstyle\sum_{l|\mathcal{M}_i^l=j} c_l^\gamma (\sum_{k=b_i^l}^{f_i^l} t_i^k)^{1-\gamma} \\ &\textbf{Subject to} \quad \forall i, \textstyle\sum_k t_i^k \le T_i, \\ &\qquad\qquad\quad \forall i, t_i^k \ge 0. \end{aligned} \tag{3.8}$$

Lemma 3. *The objective function (according to Equation (3.8)) is a convex function.*

Proof. Since leakage power consumption remains constant (which is convex), we will prove that the dynamic part of the energy consumption function is convex:

$$E(\tau) = \sum_{1\le i\le n} C_i^\gamma (<\alpha_i, \tau>)^{1-\gamma}. \tag{3.9}$$

Here τ refers to a k-dimension positive vector, in which each element is positive and refers to the length of a specific segment of a DAG task. α_i is a binary vector, in which each element $\alpha_{i,j} \in \{0,1\}$ identifies if the node is selected for the segment. $|\alpha_i| \ge 1$ since at least one segment must be assigned). $<\alpha_i, \tau>$ refers to the inner-product of the two vectors, C_i refers to a non-negative constant, and $\gamma \in [2,3]$. Thus the energy consumption is modeled as $E(\tau)$ – a function over the time-allocation $\tau \in \mathbb{R}_+^k$.

We prove the convexity of $E(\tau)$ when $\tau \in \mathbb{R}_+^k$ with the following four steps:

1. We name $f(\tau) =< \alpha, \tau >$ as a function of inner-product of τ with any binary vector α and $|\alpha| \geq 1$. Obviously, this function is a linear function over τ and should be both *convex* and *concave*. Further, given $\tau \in \mathbb{R}^k_+$, we have $f(\tau) > 0$. Thus we can conclude $f(\tau)$ is a *positive concave* function.

2. According to page 3-3 of [31], x^p is convex when $x > 0$ and $p \leq 0$. Thus, when $\gamma \in [2, 3]$ (i.e., $-2 \leq 1 - \gamma \leq -1$) and $x > 0$, the function $g(x) = x^{1-\gamma}$ should be a *non-increasing* convex function.

3. According to page 3-17 of [31], if $g(x)$ is a *non-increasing convex* function and $f(\tau)$ is a *concave* function over $\forall \tau \in \mathbb{R}^k_+$, then $g(f(\tau))$ is a convex function over $\forall \tau \in \mathbb{R}^k_+$.

4. The function $E(\tau)$ and $f_i(\tau)$ could be written as:

$$E(\tau) = \sum_{1 \leq i \leq n} C_i^\gamma g(f_i(\tau)) \tag{3.10}$$

$$f_i(\tau) = (< \alpha_i, \tau >) \tag{3.11}$$

As C_i^γ is non-negative, $E(\tau)$ could be considered as the *non-negative-weighted sum* of convex functions (i.e., $g(f_i(\tau))$), and $E(\tau)$ is a *convex function*.

□

Theorem 4. *Any gradient based method (e.g., fmincon[56] in Matlab) would lead to sub-optimal power consumption under federated scheduling scheme with task decomposition.*

Proof. The sub-optimality comes from three facts:

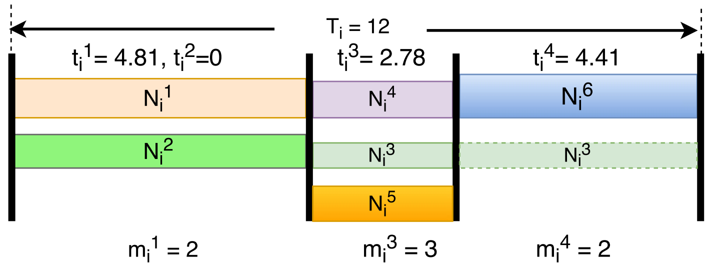


Figure 3.4: The sub-optimal segment length assignment for power efficiency of the sample task τ_i (in Figure 2.1), with an average power consumption of 2.94 Watts.

- The objective function is *convex* as it is a sum of several convex (including linear) functions of the variables t_i^k (detailed proof in Lemma 3).
- The linear equality constraints are necessary and sufficient (Lemma 1) for real-time schedulability and predecessor conditions from the input DAG task.
- The variables t_i^k are sufficient to represent a possible optimal scheduler regarding power consumption; i.e., it is safe to assume uniform speed during each segment (Theorem 2).

$\square$

Figure 3.4 shows the sub-optimal segment length assignment for the given task τ_i. As usual, the height of each block represents the speed of the processor during each segment.

3.4 Processor Sharing: Efficiency Improvement

Task decomposition transforms the parallel task into a set of sequential tasks. The process tries to maximize the degree of parallelism (i.e., assigning as many processors to each DAG task as possible). However, some of these processors may be lightly loaded with poor energy efficiency as

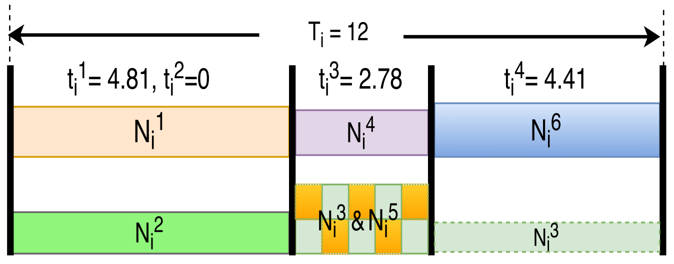


Figure 3.5: The execution pattern for τ_i (in Figure 2.1) after merging two different Processors.

the leakage power consumption becomes the majority cost (as demonstrated in Figure 2.3). Thus the solution derived in Section 3.3 is only sub-optimal and can be further improved if we allow merging the lightly loaded processors into a single one, such that leakage power is reduced (see Figure 3.5 and Example 5).

Example 5. *The execution pattern for τ_i (in Figure 2.1) after merging Processors 2 and 3. Here, Node $\mathcal{N}_i^3$ and Node $\mathcal{N}_i^5$ will share Processor 2 (i.e., execute under EDF) within time window* $[4.81, 7.59)$ *at a higher execution speed. Such a merging results in a reduction of average power consumption to* 2.80 *Watts.*

In this section, we deal with this issue and further improve the overall energy efficiency of our scheduler by merging the workloads assigned to different processors onto a single one. Specifically, in Subsection 3.4.1, we merge processors that have been assigned to the same DAG task. In this step, each DAG task is handled individually and the resulting processor-node/DAG assignment remains in the federated scheduling framework. However, in this subsection, we have assumed that each processor to be merged only once. That is we only allow the combination of two processors that have never been part of any merging previously. In Subsection 4.1, this constraint is relaxed and we allow merging three or more processors into one. In Subsection 4.2, we discuss the importance (in order to improve the overall energy efficiency of our scheduler) of applying any gradient based method to calculate the optimal segment length after an intra-DAG processor merging. In

Subsection 4.3, we further extend the technique for inter-DAG processor merging.

Remark 3. Normally, the effect of task migrations and context switches is not considered while deriving schedulability test for real time tasks. We are also ignoring the effects of these phenomena.

3.4.1 Merging Processors Assigned to the Same DAG

Federated scheduling of DAG tasks provides isolation among tasks during execution, such that inter-task interference as well as context switch delays remain small during run-time. In this subsection, we stay in the federated scheduling framework and only consider potential merges among processors of the *same* DAG.

Given a DAG task τ_i with a federated task-to-processor assignment $j = \mathcal{M}_i^k$, the processor execution speeds S_j^k for each segment, segment lengths t_i^k, and the period T_i. For any processor j assigned to this DAG, its original power consumption can be calculated as

$$P_j = \beta + \sum_k \frac{t_i^k}{T_i}(S_j^k)^\gamma. \tag{3.12}$$

Any pair of processors $\{j, j'\}$ share the same period and segment information as they are assigned to the same DAG task. As a result, the new execution speed for each segment (when merged together) will simply be the sum of the two old ones; i.e., $S_j^l + S_{j'}^l$, and the average power consumption for this new processor can be calculated as:

$$P_{j,j'} = \beta + \sum_k \frac{t_i^k}{T_i}(S_j^k + S_{j'}^k)^\gamma. \tag{3.13}$$

The pairwise potential power saving can be calculated directly by:

$$\mathcal{P}_{j,j'} = P_j + P_{j'} - P_{j,j'}. \tag{3.14}$$

With the pairwise potential power saving, the Maximization of Power Saving (MPS) problem we are dealing with in the section can be described as follows:

- Given the potential power savings ($\mathcal{P}_{M_i \times M_i}$) for merging each pair of the M_i processors, we wish to find a list of mutual exclusive processor-pairs $\{(p_1, p'_1), ..., (p_N, p'_N)\}(N \leq M_i/2)$, such that the total power saving $\mathcal{P}_i = \sum_{j=1}^{N} \mathcal{P}_{p_j, p'_j}$ is maximized.

Theorem 5. *The MPS problem is NP-Complete.*

Proof. MPS is in NP as it takes linear time to verify whether a given solution satisfies the mutual exclusion constraints. The NP-Hardness comes from the reduction of a well known NP-Complete problem: *Maximum Independent Set* (MIS). An independent set is a set of (weighted) nodes in a graph that no two of which are adjacent. For each node in the graph of MIS, we can construct an edge with same weight in the graph of MPS, and adjacency of those edges (whether or not they share a node) in MPS can be determined by the adjacency of the edges in the graph of MIS; i.e. each edge in MIS corresponds to a node in MPS, refer to Figure 3.6 for details. Figure 3.6(a) shows a DAG of four processor assignments with potential power savings for merging each pair of the processors, while Figure 3.6(b) shows its equivalent expression with vertices representing all edges in (a), and edges representing the mutual exclusive constraints. Since this polynomial (linear)-time mapping maintains the adjacency relationship of weighted nodes (in MIS) or edges (in MPS), a solution of MIS (a subset of n_m non-adjacent nodes with maximum total weight) will correspond to a solution of MPS (n_m non-adjacent edges with maximum total weight), and vice

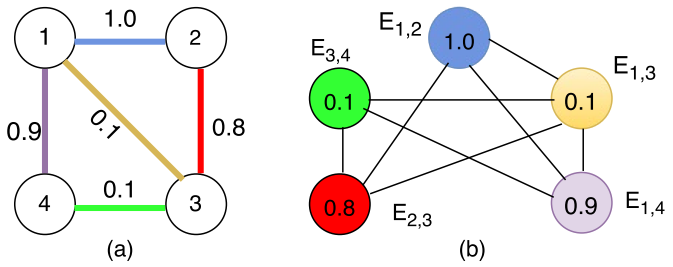


Figure 3.6: The equivalence of the MPS problem and the MIS problem

versa. □

Example 6. *In Figure 3.6, four processors are assigned to a DAG task. The weight $\mathcal{P}_{i,j}$ for each edge represents the potential power saving when merging processors i and j, calculated from* (3.14)*. The edge $\{2,4\}$ is missing since merging these two processors will lead to higher power consumption (i.e., $\mathcal{P}_{2,4} < 0$). For each vertex in Figure 3.6 (b), there is a corresponding edge with the same weight in Figure 3.6 (a), and vice versa. A feasible subset of edges in Figure 3.6 (a) (e.g., $\{1,4\}$ and $\{2,3\}$) corresponds to a subset of vertices in Figure 3.6 (b) (e.g., $E_{1,4}$ and $E_{2,3}$) that none of the two are directly connected by an edge.*

For this example, we could choose to merge Processors $1\&2$ and $3\&4$ (with a gain of 1.1 Watts), $1\&4$ and $2\&3$ (with a gain of 1.7 Watts), or $1\&3$ (with a gain of 0.1 Watts). Although obviously the second option is leading to the optimal solution, we need to explore all combinations to find that out (Theorem 5 already shows the intractability). As a result, instead of seeking for the global optimal solution for merging, here we choose to greedily select (see Step 2 below) the pair with the maximum gain in each step.

Now we describe the **key steps of our proposed processor merging method**:

1. **For** *each* pair of processors $\{j, j'\}$ of the (same) DAG, calculate the potential power savings

$\mathcal{P}_{j,j'}$ for merging them together according to (3.14).

2. *Greedily* choose the pair $\{j, j'\}$ of processors with the maximum power saving $\mathcal{P}_{j,j'}$, and merge them together by updating $\mathcal{P}'$ value(s) of the nodes on j' to j. The merged nodes will be executed on processor j under EDF, with given per-segment (fixed) speed settings. Note that EDF is an optimal uni-processor scheduler for sporadic task systems, and thus will guarantee temporal correctness as far as cumulative capacity remains the same.

3. *Remove* the two processors (and also the new one) from the candidate pool, by updating elements in the jth row, the j'-th row, the jth column, and the j'-th column of the power saving matrix $\mathcal{P}$ into 0.

4. **If** there are no positive elements in $\mathcal{P}$, return the updated mapping $\mathcal{P}'$, **else** go to Step 2 (i.e., merging two un-touched processors may lead to further energy savings).

Although the MIS problem in general cannot be approximated to any constant factor in polynomial time (unless P = NP) [22], fortunately, each edge in the original figure can be joint with at most $2(M_i - 2)$ other edges, which indicates that the degree of each vertex in the graph after problem transformation is upper bounded by $2(M_i - 2)$, leading to the following *approximation ratio bound.*

Theorem 6. *The greedy approach has an approximation ratio no greater than* $(2M_i - 2)/3$*, where* $M_i \geq 3$ *is the total number of processors before merging of DAG task* τ_i*; i.e., the degree of parallelism of the task.*

Proof. Since we only allow a processor to be considered in *one* pair in each round, the graph resulted from the reduction in Theorem 5 is a $(2M_i - 4)$-regular graph; i.e., the degree of each vertex cannot exceed $2M_i - 4$. According to Theorem 5 in [69], the greedy algorithm achieves an approximation ratio of $(2M_i - 2)/3$. □

Remark 4. *When $M_i = 2$, there are only two processors in the candidate pool, and the decision is straightforward – based on whether merging them can lead to lower power consumption.*

Remark 5. *As mentioned in Section 2, we consider a continuous frequency scaling and, at any segment, the computed frequency can be rounded up to the next discrete mode. Initially, it may seem that increasing the speed (by stepping up to the next level) give more room to perform inter-core merging. However, the speed at any segment depends on the workload at this segment and its sub-optimal length which is calculated through a gradient-based method. So rounding up the frequency to its next level may break the sub-optimality and may not yield better energy efficiency.*

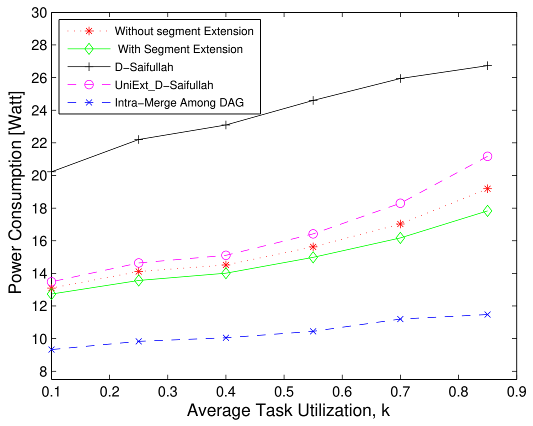
3.5 Simulation Study

In this section, we use experiments to evaluate the power efficiency of the proposed mechanisms, and compare them with existing algorithms for DAG task systems.

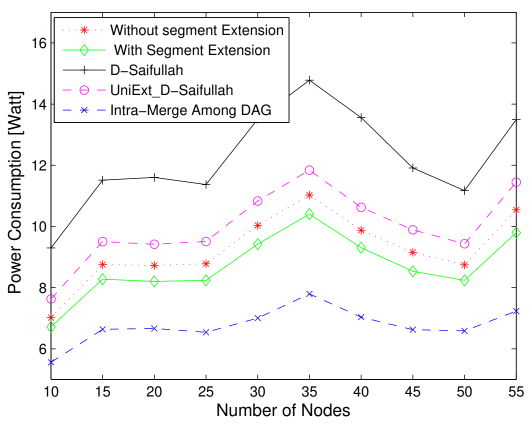
Generation of workloads. Our DAG generator follows the Erdos-Renyi method [42] with a given number of nodes. For the *harmonic period* case, the periods are multiples of each other [109] by enforcing them to be powers of 2. Specifically, we find the smallest value χ such that $L_i \leq 2^{\chi}$ and set T_i to be 2^{χ}. Regarding the *arbitrary period* case, we use *Gamma distribution* [58] to generate a random parameter, and set the period as $T_i = L_i + 2(c_i/m)(1 + \Gamma(2, 1)/4)$ (according to [109]). Power consumption is calculated using Equation (2.1) and the value of $\alpha, \beta\ and\ \gamma$ are set to 1.76, 0.5 and 3 respectively.

3.5.1 Experiment Under Single Merging of Processors

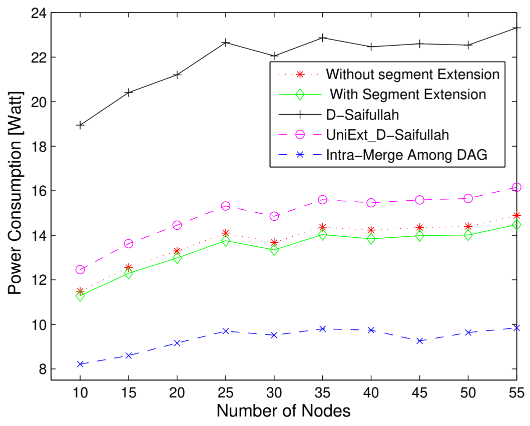
Here compare the power consumption by varying two parameters: (i) task periods (densities) (Subsubsection 3.5.1.1) and (ii) number of nodes in each DAG task (Subsubsection 3.5.1.2). Under each



(a) Comparison of power consumption with different approaches for DAGs with a fixed number of nodes as 30.



(b) Power consumption comparison with different approaches for tasks with harmonic periods.



(c) Power consumption comparison with different approaches for tasks with arbitrary periods.

Figure 3.7: Power consumption comparisons for task sets for various settings under single merging.

parameter setting, we randomly generate 100 different DAG task sets, each consisting of 5 DAG tasks. Note that, we could report the energy consumption over a hyper-period for each task set. However, reporting energy consumption over a hyper-period for each task set will not be a fair comparison as tasks are randomly generated and their periods and hyper-periods vary a lot. To ensure a fair comparison, we have divided such energy consumption (in Joule) over a hyper-period by the length of the hyper-period (in seconds), such that the reported data represents average power consumption (in Watt) on the y-axis. We compare the *average power consumption* of the following scheduling algorithms:

- Federated scheduling with task decomposition, where each node is executed as soon as possible under full speed [109], denoted by *D-Saifullah*;
- Federated scheduling with task decomposition (length of each segment is further extended uniformly according to their loads) [109], denoted by *UniExt_D-Saifullah* ;
- Federated scheduling with task decomposition, where lengths of segments are determined by the proposed convex optimization (Subsection 3.3.3);
- Energy-sub-optimal federated scheduling with task decomposition, where lengths of segments are determined by convex optimization (Subsection 3.3.3) after performing segment extension (Subsection 3.3.2);
- Federated scheduling with intra-DAG processor merging (Subsection 3.4.1);

3.5.1.1 Varying Task Periods

Here we vary the minimum inter-arrival separation for each task, such that the average utilization of a set is controlled. We vary the period in an allowable range ($L_i \le T_i \le C_i$) by assigning T_i as $L_i + (1-k)(C_i - L_i)$, where $k \in [0, 1]$ is named as the *utilization* of the task — note that this is different from the normal utilization definition for sequential tasks. We fix the number of nodes within each DAG task as 30, and show the average power consumption in Figure 3.7(a).

The first thing we notice from Figure 3.7(a) is that the average energy consumption increases as the average utilization of the set increases (due to decreasing of the period). This phenomenon makes sense as higher utilization would lead to tighter real-time restrictions, which lead to less room for our segment length optimization.

As shown in Figure 3.7(a), stretching each segment would lead to significant power savings compared to finishing them at full speed and leaving the processor idle for some portion of time (matching Theorem 2). Comparing to the existing uniform stretching for all segments of each DAG task, our convex optimization based methods would find a better execution pattern in terms of power efficiency. We also found that segment extension is helpful in removing unnecessary constraints for finding better execution patterns.

It is easy to tell that the improvements to the average power consumption are huge when applying the processor merging techniques (see Section 3.4). The improvement is larger when utilization of the task is high. On average, our proposed methods (including segment extension and intra-DAG merging) are leading to a reduction of the power consumption ranging from 29.2% to 40.5%.

3.5.1.2 Varying Numbers of Nodes in a DAG Task

Now we vary the number of nodes within each DAG without changing the period. In this set of comparisons, we consider both harmonic (reported in Figure 3.7(b)) and arbitrary periods (reported in Figure 3.7(c)) for a set. For each setting of parameters, we randomly generate 100 task sets with various number of nodes (10 to 55, with an increment of 5) and report the average performances of the power consumption over the 100 sets.

First of all, we observe similar improvements in energy efficiency with the proposed techniques when the number of nodes vary, comparing to the previous set of experiments (with fixed number of nodes and varying task utilization). Specifically, the intra-DAG merging technique (refer to Subsection 3.4.1) leads to a reduction in the power consumption for at least 27.29% (34.27%)

for harmonic (arbitrary) periods, (compared to the result of convex optimization with segment extension discussed in Subsection 3.3.3), while the average power savings are 28.23% and 37.80%.

Secondly, when comparing curves in Figures 3.7(b) and 3.7(c), we observe that task sets with harmonic periods typically result in lower energy consumption compared to arbitrary periods (under same task utilization and number of nodes per task).

Finally, from the reported performances, we did not observe significant dependencies between the power consumption and the number of nodes for the DAG tasks. This indicates that the proposed methods are *robust* to various settings of parameters and combination of DAG tasks.

3.6 Conclusion

This chapter studies the scheduling of a set of sporadic DAGs with implicit deadlines. Upon guaranteeing real-time correctness, we try to minimize the overall power consumption of the platform. A power-sub-optimal scheduler is proposed under the condition of federated scheduling and task decomposition. Achieving the optimal solution for the more general (non-federated) case is shown to be NP-Complete. Based on the solution under federated scheduling, a greedy heuristic is proposed to further improve the power efficiency, with proved upper bound of the approximation ratio. The effectiveness of the proposed approach is also evaluated through simulation study. Experimental results on randomly generated workloads show that our algorithms achieve an energy saving of 60% to 68% compared to existing DAG task schedulers.

CHAPTER 4: ENERGY-EFFICIENT FEDERATED SCHEDULING OF REAL-TIME DAG TASKS WITH INTER-TASK PROCESSOR MERGING

In the previous chapter, we have described the technique of merging processors assigned to the same DAG. Such an intra-task processor merging (i.e., merging two lightly loaded processors onto one) reduces the total number of processors and reduces power consumption. So far, we have assumed that each processor can be merged only once. However, allowing multiple merging to any specific processor can further increase energy saving. In this chapter, we will relax the assumption of a single merge per processor. We also discuss the inter-task processor merging. We further select the lightest loaded unmerged processors of each DAG (after intra-task merging) as candidates and merge them with lightly loaded processors from a different DAG.

4.1 Multiple Merging Among the Processors Assigned to the Same DAG

In this section, we will describe the technique of multiple merging. Here are the key steps of our proposed intra-DAG processor merging (multiple times) method:

1. **For** *each* pair of processors $\{j, j'\}$ of the (same) DAG, calculate the potential power savings $\mathcal{P}_{j,j'}$ for merging them together according to Equation (3.14).
2. **If** there is no positive element in $\mathcal{P}$, return with no power saving, **else** go to Step 3.
3. *Greedily* choose the pair $\{j, j'\}$ of processors with the maximum power saving $\mathcal{P}_{j,j'}$, and merge them together by updating $\mathcal{P}'$ value(s) of the nodes on j' to j. The merged nodes

This chapter has been previously published at ACM Transactions on Embedded Computing Systems and available at Bhuiyan, A., Guo, Z., Saifullah, A., Guan, N., & Xiong, H. (2018). Energy-efficient real-time scheduling of DAG tasks. ACM Transactions on Embedded Computing Systems (TECS), 17(5), 1-25.

will be executed on processor j under EDF, with given per-segment (fixed) speed settings. Note that EDF is an optimal uni-processor scheduler for sporadic task systems, and thus will guarantee temporal correctness as far as cumulative capacity remains the same.

4. *Remove* the two processors (and also the new one, see Remark 3) from the candidate pool by updating elements in the jth row, the j'-th row, the jth column, and the j'-th column of the power saving matrix $\mathcal{P}$ into 0.

5. **If** there is no positive elements in $\mathcal{P}$, go to Step 6, **else** go to Step 3 (i.e., merging two un-touched processors may lead to further energy savings).

6. Let M' be the total number of merges conducted in Steps 1 to 4, where $M' \leq M_i/2$ ($M_i =$ the total number of processor allocated to τ_i). Update M_i as $M_i \leftarrow M_i - M'$.

7. Repeat Steps 1 to 4.

4.2 Calculating Optimal Segment Length After the Intra-DAG Processor Merging

In Chapter 3, we have discussed the technique to optimally determine the segment length for a DAG task τ_i (Subsection 3.3.3). We also have described the processor merging technique among the same DAG task (Subsection 3.4.1). We already know from Theorem 4 that any gradient based method would lead to sub-optimal power consumption under federated scheduling scheme with task decomposition. Now we will show that if we re-apply any gradient based method to calculate segment length after applying the intra-DAG processor merging it will further reduce the power consumption. Once we solve the optimization problem mentioned in Subsection 3.3.3, for each node $\mathcal{N}_i^l$ we have the information regarding its execution speed within its scheduling window. As all the processors assigned to the same DAG share the same period and segment information, after merging the new execution speed for each segment will simply be the sum of the two old ones.

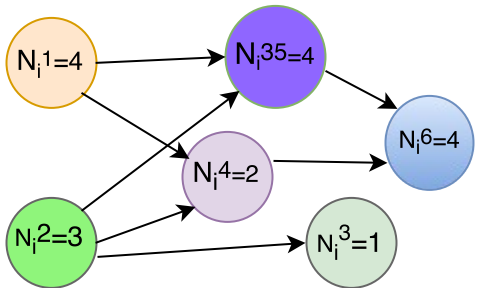


Figure 4.1: The modified DAG task τ_i (from Figure 2.1) after applying the intra-task processors merging technique.

We consider that the DAG τ_i is reduced to τ_i' after a merge. The consecutive time windows where the speed remains same will be considered under the same node (according to Theorem 2). As we have the updated information regarding the execution speed of τ_i' within its scheduling window it is easy to calculate the execution time for each node as well.

Example 7. *Figure 4.1 shows how the DAG τ_i from Figure 2.1 is modified after intra-task processor sharing. For this specific task τ_i (see Figure 3.5) node $\mathcal{N}_i^3$ and $\mathcal{N}_i^5$ share processor 2 within time window $[4.81, 7.59)$ at a higher execution speed. In this case, we will consider that the node $\mathcal{N}_i^3$ and the node $\mathcal{N}_i^5$ jointly form a new node $\mathcal{N}_i^{35}$. Note that, we must respect the predecessor constraint, i.e., $\mathcal{N}_i^2 \to \mathcal{N}_i^3$ and $\mathcal{N}_i^1 \to \mathcal{N}_i^5$. Hence, we consider both the nodes $\mathcal{N}_i^1$ and $\mathcal{N}_i^2$ as the parent of the newly formed node $\mathcal{N}_i^{35}$. The remaining part of node $\mathcal{N}_i^3$ (which is executing within the time window [7.59,12)) will be considered as another node.*

Now we have a changed DAG τ_i' where each node $\mathcal{N}_i^l$ has some execution requirements and predecessor constraints. After merging we did not change the segment length. As we are merging two different processors into one, the execution requirement at specific segment changes. So it is worth calculating the segment length of the DAG τ_i'. In order to determine the sub-optimal segment length t_i^k, we will use any *gradient based* (e.g., fmincon [56] in Matlab) method because:

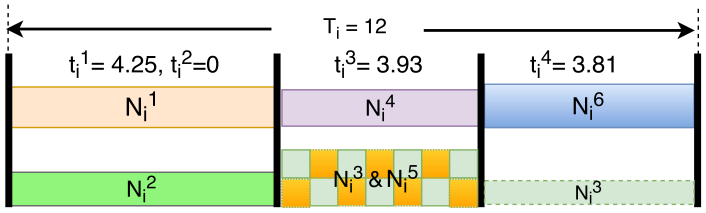


Figure 4.2: The sub-optimal segment length assignment (after merging Processors) for power efficiency of the sample task τ_i.

- The objective function still remains *convex* as it is the sum of multiple convex functions of the variables t_i^k; and
- The linear equality constraints are necessary and sufficient (Lemma 1) for real-time schedulability and predecessor conditions from the *modified* DAG task.

Figure 4.2 shows the sub-optimal segment length assignment for the modified DAG (which is achieved by merging the lightly loaded processors) τ_i'.

4.3 Merging Processors Assigned to Different DAGs

The merging process described in the previous subsections may significantly reduce the total number of lightly loaded (energy-inefficient) processors. However, due to the federated scheduling limitation, one (or more) lightly loaded processor(s) for each DAG may still not get paired with just because it (they) cannot find a good "partner" that was assigned the same DAG task. In this subsection, we further select the lightest loaded unmerged processors of each DAG (after intra-DAG merging) as a candidate, and perform inter-DAG merging under a similar approach; i.e., calculate all pairwise energy savings and greedily merge the pairs with maximum possible power saving.

Note that different tasks may have different periods, sporadic release patterns, and execution patterns (segment lengths after decomposition), such that we cannot simply cumulate the execution speeds with Equation (3.13) when calculating the new speed pattern for power consumption upon inter-DAG merging. In this section, we propose a fast algorithm to estimate the average energy consumption of the two processors from two different DAG tasks after merging them into one processor, with (potential) non-synchronized release.

With respect to the unknown phase difference between the two DAGs, we assume that all phases are equally likely to occur, and model the speed patterns of them as two random processes $\mathcal{S}_i(t)$ and $\mathcal{S}_j(t)$, where $t \in [0, +\infty)$. Power consumption of the merged processor at time instant t is:

$$e_{i,j}(t) = \beta + \alpha \cdot (\mathcal{S}_i(t) + \mathcal{S}_j(t))^\gamma. \tag{4.1}$$

The expectation of $e_{i,j}(t)$ over $t \in [0, +\infty)$ is:

$$\begin{aligned}\mathbb{E}\left(e_{i,j}(t)\right) &= \beta + \alpha \cdot \mathbb{E}\left((\mathcal{S}_i(t) + \mathcal{S}_j(t))^\gamma\right) \\ &\quad \{\text{The values of } \mathcal{S}_1(t) \text{ and } \mathcal{S}_2(t) \text{ are finite}\} \\ &= \beta + \alpha \sum_{s \in S_{1,2}} s \cdot p(s),\end{aligned} \tag{4.2}$$

where $S_{1,2}$ is the (finite) set of values that $(\mathcal{S}_1(t) + \mathcal{S}_2(t))^\gamma$ can possibly take, which can be calculated as:

$$S_{1,2} = \left\{(s_1^l + s_2^{l'})^\gamma | 1 \leq l \leq M_1, 1 \leq l' \leq M_2\right\},$$

and $p(s)$ refers to the probability of the value $s \in S_{1,2}$; i.e.,

$$p(s) = \sum_{1 \le l \le M_1} \sum_{1 \le l' \le M_2} \frac{1}{M_1 M_2} \cdot \delta(s, (s_1^l + s_2^{l'})^\gamma).^1$$

The key to calculating average power consumption is to identify all the possible execution speeds (sum of a pair of speeds, each of which is selected from the set of possible execution speeds of the two processors being merged), and the likelihood (or joint distribution) of this speed to occur according to the original execution patterns. We calculate the average power consumption because in a real-time system energy saving scheme for the average behavior (while guaranteeing the worst case requirements) seems more beneficial in terms of energy saving.

4.4 Experiment Under Multiple Merging of Processors

In this section, we show the improvement in power consumption using our proposed technique of multiple merging among the processors. We generate the DAGs using the Erdos-Renyi method [42] (refer to Section 3.5 for details). We compare our results with a simple baseline that was studied in [130]. This baseline approach studied a greedy slack stealing scheduling (**GSS**) approach for energy minimization for an application consisting of inter-dependent sequential tasks. While those dependencies among the tasks were represented by a DAG, the model consists of a single DAG and does not consider recurrent tasks. We can consider that approach for scheduling one DAG. As the work in [130] did not consider parallel task and we consider this work as a baseline to compare with our work, in order to provide a fair comparison it is required to modify the power and system models and the graph model used in [130].

Regarding the power model, first, [130] did not consider the processor static power dissipation.

[1] $\delta(s, (s_1^l + s_2^{l'})^\gamma) = 1$ if $s = (s_1^l + s_2^{l'})^\gamma$, and $= 0$ else-wise.

Second, [130] considered two real processor models, (a) *the Transmeta model* and the (b) *Intel XScale model* (refer to Subsection 2.3 in [130]). Both of them provide a set of voltage/speed levels. But in our model, we have considered continuous frequency scheme. So while executing the GSS algorithm proposed in [130], we use the energy model used in Equation (2.1). We will assume that whenever a processor is introduced in the system it always remains on. We will also consider minimum inter arrival separation (i.e. period) for a DAG. We make these assumptions in order to incorporate the static power consumption according to Equation (3.8).

Regarding the graph model, [130] considered three different kinds of vertices: computation nodes, AND nodes, and OR nodes (refer to Subsection 2.1 in [130]). Here the computation nodes are labeled by two attributes, c_i and a_i, which denotes the maximum and average computation requirement for the corresponding node. AND nodes and OR nodes do not have any such attributes. An AND node can be executed after all of its predecessors finish execution. Similarly, all of its successors can start execution after it finishes execution. But for the OR nodes, it depends on only one of its predecessors. Similarly only one of its successors depends on this node. However, in our work, we have considered only the computation nodes with only one attribute, their worst-case execution time. In order to provide a fair comparison, we change the graph model used in [130] according to ours. So we will consider the DAG where each node will be considered as a computation node. Instead of two, there will be only one attribute c_i^j which denotes maximum computation requirement for the node $\mathcal{N}_i^j$. We also consider the *precedence constraints* among the computation nodes. These evaluation are conducted considering the homogeneous platform.

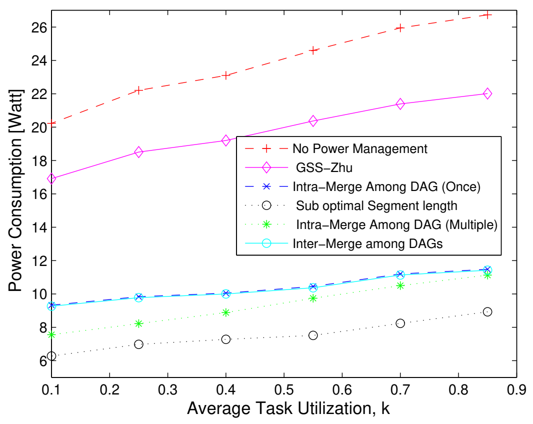
In this subsection, we will evaluate the performance of the technique proposed by us and the technique proposed in [130] based on power consumption. We vary two parameters (i) task periods (utilization) (Subsubsection 3.5.1.1) and (ii) number of nodes in each DAG task (Subsubsection 3.5.1.2) and will use the same set of DAGs. In this subsection, we compare the average power consumption considering the following schemes:

- No Power Management (where every task executes at full speed)
- Greedy Slack Stealing (GSS) algorithm, denoted by *GSS-Zhu*;
- Federated scheduling with intra-DAG processor merging (each processor to be merged only once) (Subsection 3.4.1)
- Federated scheduling with intra-DAG processor merging (each processor can be merged multiple times) (Section 4.1)
- Recalculation of the segment lengths later to the intra-DAG processor merging, where lengths of segments are determined by the proposed convex optimization (Section 4.2)
- Shared scheduling with inter-DAG processor merging (Section 4.3);

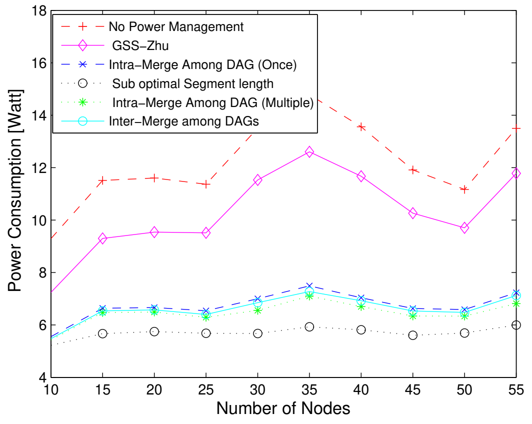
4.4.1 The Effect of Varying Task Periods or Utilization

Here we have compared the performance of intra-DAG processor merging with single and multiple merging, inter-DAG processor merging, and the recalculation of the segment lengths after applying the intra-DAG processor merging by varying the minimum inter-arrival separation for each task as described in Section 3.4. We use the same settings, the number of nodes within each DAG task is 30. We show the average power consumption in Figure 4.3(a).

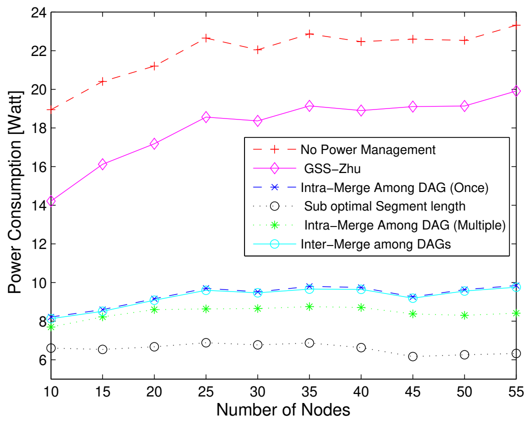
Similar to the phenomenon we have noticed in the previous chapter (Figure 3.7(a)) the average energy consumption is directly proportional to the average task utilization. The results in Figure 4.3(a) indicate that our scheduling algorithm is superior to the GSS. Total energy consumption by the intra-DAG processor merging (only one merge) is significantly less than the GSS, which is further reduced by allowing multiple merging and recalculating the optimal segment length after merging. In particular, the energy consumption by the intra-DAG processor merging (only



(a) Comparison of power consumption with different approaches for DAGs with a fixed number of nodes as 30.



(b) Power consumption comparison with different approaches for tasks with harmonic periods.



(c) Power consumption comparison with different approaches for tasks with arbitrary periods.

Figure 4.3: Power consumption comparisons for task sets for various settings under multiple merging.

one merge) is at least 44.85% less than the GSS. It is easily observable that the improvements to the average power consumption are huge when our convex optimization based methods find a better execution pattern after intra-DAG processor merging (only one merge) and the improvement is larger when the utilization of the task is high. This phenomenon makes sense as the objective function described in Equation (3.8) does not lose its convexity after intra-DAG processor merging.

So, re-applying a gradient-based method for determining sub-optimal segment length assignment (after merging Processors) yields a better result in further reducing the power consumption. On average, it leads to a reduction of the power consumption ranging from 59.41% 63.09%.

4.4.2 Varying Numbers of Nodes in a DAG Task

Now we will show the comparison of the above-mentioned algorithms by varying the number of nodes (from 10 to 55, with an increment of 5). We randomly generate 100 task sets and report the average power consumption over the 100 sets. Again we will consider both harmonic (reported in Figure 4.3(b)) and arbitrary periods (reported in Figure 4.3(c)).

The improvements observed in energy efficiency with our proposed techniques under varying number of nodes are similar to that in the previous set of experiments (varying task utilization with a fixed number of nodes). Figures 4.3(b) and 4.3(c) suggest that both versions (single and multiple) of intra-DAG processor merging performs significantly better than the GSS algorithm. Intra-DAG processor merging results in at least 23.3% (single merge) and 25.05% (multiple merges) lower energy consumption compared to the GSS for the harmonic periods. When considering the arbitrary periods, this reduction becomes 42.22% (single merge) and 45.83% (multiple merges). If the gradient based methods is considered to find a better execution pattern after intra-DAG processor merging (single merge), it brings further improvements in energy consumption. On average, for the harmonic task periods, it leads to a reduction of the power consumption ranging from 27.8% to 52.94% and for the arbitrary task periods the range is 53.55% to 68.22%. All these results indicate that our proposed techniques outperform the GSS algorithm in terms of energy efficiency.

Finally, Figures 4.3(b) and 4.3(c) report that the task sets with harmonic periods result in lower energy consumption compared to the task sets with arbitrary periods. Also, for the DAG task sets, we have observed that there are no significant dependencies between the energy consumption and the number of nodes.

4.5 Conclusion

In Chapter 3, we have proposed a power-sub-optimal scheduler under the condition of federated scheduling and task decomposition. Based on the solution under federated scheduling, we have also presented a greedy heuristic to improve power efficiency further. In this chapter, we improve the performance (w.r.t. energy saving) of the greedy algorithm by allowing multiple merging to any specific processor. Finally, we have discussed the inter-task processor merging, where we select the lightest loaded unmerged processors of each DAG as candidates. These candidates are merged with lightly loaded processors from a different DAG.

CHAPTER 5: ENERGY-EFFICIENT PARALLEL REAL-TIME SCHEDULING ON CLUSTERED MULTI-CORE

Energy-efficiency is a critical requirement for computation-intensive real-time applications on multi-core embedded systems. Multi-core processors enable intra-task parallelism, and in this chapter, we study energy-efficient real-time scheduling of constrained deadline sporadic parallel tasks, where each task is represented as a directed acyclic graph (DAG). We consider a clustered multi-core platform where processors within the same cluster run at the same speed at any given time. A new concept named *speed-profile* is proposed to model per-task and per-cluster energy-consumption variations during run-time to minimize the expected long-term energy consumption. The proposed energy-aware real-time scheduler is implemented upon an *ODROID XU-3* board to evaluate and demonstrate its feasibility and practicality. To complement our system experiments in large-scale, we have also conducted simulations that demonstrate a CPU energy saving of up to 67% through our proposed approach compared to existing methods.

5.1 Introduction

Multi-core processors appear as an enabling platform for embedded systems applications that require real-time guarantees, energy efficiency, and high performance. Intra-task parallelism (a task can be executed on multiple cores simultaneously) enables us to exploit the capability of the multi-core platform, and facilitates a balanced distribution of the tasks among the processors. Such a balanced distribution leads to energy efficiency [100]. Directed Acyclic Graph (DAG) task model [109] is one of the most generalized workload model for representing deterministic intra-

This chapter has been previously published at IEEE Transactions on Parallel and Distributed Systems and available at Bhuiyan, A., Liu, D., Khan, A., Saifullah, A., Guan, N., & Guo, Z. (2020). Energy-efficient parallel real-time scheduling on clustered multi-core. IEEE Transactions on Parallel and Distributed Systems, 31(9), 2097-2111.

task parallelism. Recently, quite some effort has been spent on developing real-time scheduling strategies and schedulability analysis of DAG tasks, few to mention [102, 17, 81, 30, 109, 82, 16].

There are several real-world application that uses the DAG model. For example, the work in [102] studies problems related to scheduling parallel real-time tasks, modeled as DAG, on multiprocessor architectures. In a homogeneous computing environment, a low-complexity compile-time algorithm for scheduling DAG tasks is proposed in [68]. Another example would be systems that control asynchronous devices, such as the local-area network adapters that implement real-time communication protocols.

Since many of those applications are battery-powered, considering energy-efficient approaches for designing such a platform is crucial. Thanks to the fact that modern generation processors support dynamic voltage and frequency scaling (DVFS), where each processor can adjust the voltage and frequency at runtime to minimize power consumption, per-core energy minimization becomes possible during run-time. Despite the hardness of the problem [7], a significant amount of work has considered power minimization for non-parallel tasks on a multi-core platform (refer to [9] for a survey). Regarding parallel tasks, Guo et al. studied energy-efficient real-time scheduling for DAG tasks as an early research effort [67]. They adopted the federated scheduling and task decomposition framework [109] for minimizing system energy consumption via per-core speed modulation. As the only step (that we are aware of) towards energy-aware scheduling of real-time DAG tasks, they targeted an exciting problem and laid some of the foundations of this work. However, the attention of [67] is restricted to implicit deadline tasks with a system model of per-core DVFS.

Unfortunately, per-core DVFS becomes inefficient as it increases the hardware cost [70]. For balancing the energy efficiency and the hardware cost, there is an ongoing trend to group processors into islands, where processors in the same island execute at the same speed. For example, a big.LITTLE platform (e.g., ODROID XU-3 [94]) consists of high performance (but power-hungry) cores integrated into 'big' clusters and low-power cores into 'LITTLE' clusters. Such a platform executes several real-life applications with heavy computational demands (e.g., video streaming [86]) in an energy-efficient manner. Apart from the energy consumption issue, a multi-core platform enables task execution with high-performance demand and tight deadlines, essential for computation-intensive real-time systems, e.g., autonomous vehicles [77]. Such kind of system balances the energy efficiency and hardware cost compared to the traditional (with individual frequency scaling feature) multi-core models. The scheduling problem becomes highly challenging on such platforms because:

(i) The relationship between the execution time, frequency, and the energy consumption is nonlinear, making it highly challenging to minimize energy consumption while guaranteeing real-time correctness, i.e., none of the tasks miss their deadline.

(ii) Existing solution (e.g., [67]) relies on the assumption that each processor can freely adjust its speed. That solution performs poorly as the assumption is no longer valid under a more realistic platform model considered in this paper.

(iii) The speed of a cluster becomes unpredictable when shared by multiple tasks with sporadic release patterns.

In this chapter, we propose a novel technique for energy-efficient scheduling of constrained deadline DAG tasks in a clustered multi-core system. Specifically, we make the following contributions:

• We consider a more practical *cluster-based system model* where the cores must execute at the same speed at any time instant within each cluster.

• To better handle constrained deadlines, one need to capture the gaps between deadlines and upcoming releases, as well as handling sporadic releases. Considering a continuous frequency scheme, we first propose a novel concept of *speed-profile* to present the energy-consumption behavior for each task as well as each cluster, such that they could guide task partitioning in an energy-efficient manner. An efficient *greedy* algorithm is proposed to partition DAG tasks according to the constructed speed-profiles.

• To evaluate the effectiveness of our proposed technique, we implement it on the ODROID XU-3 board, a representative multi-core platform for embedded systems [94]. The experiments report that our approach can save energy consumption by 18% compared to a reference approach. For larger-scale evaluation, we perform simulations using synthetic workloads and compare our technique with two existing baselines [67, 130]. The simulation results demonstrate that our method can reduce energy consumption by up to 66% compared to the existing ones under the cluster-based platform setting.

The rest of the chapter is organized as follows. Section 5.2 discusses related work including a detailed comparison with the approaches presented in Chapter 3 and Chapter 4 Section 5.3 describes the background. Section 5.4, describes the importance of creating a speed-profile for an individual task and the whole cluster. Section 5.5 discusses the approaches to create the speed-profile (considering both the continuous and discrete frequency mode) for each task. In this section, we also propose a greedy algorithm to allocate multiple tasks in the same cluster. Section 5.6 and 5.7 presents the experimental and simulation results. Section 5.9 concludes this chapter.

5.2 Related Work

Much work has been done aimed at energy-efficient scheduling of sequential tasks in a homogeneous multi-core platform (see [9] for a survey). Considering the mixed-criticality task model and varying-speed processors, the works on [92, 20, 21, 65, 26] proposed an approach to handle the energy minimization problem. The work in [41, 39, 85, 6, 40, 91] presented an energy-efficient approach for the heterogeneous platform. Considering the real-time tasks in clustered heterogeneous platforms, the work in [41] studied the partitioned EDF scheduling policy, while [40] proposed an optimal task-core mapping technique that is fully-migrative. Considering the heterogeneous multi-core platform, a two-phase algorithm was proposed by [6]. In the first phase, they proposed a tasks-core allocation approach with the aim of reducing the dynamic energy consumption, while the second phase seeks for a better sleep state to reduce the leakage power consumption. A low overhead, DVFS-cum-DPM enabled energy-aware approach, HEALERS, was proposed by [91]. However, none of them considered the intra-task parallelism. Considering a clustered heterogeneous MPSoC platform, a migrative cluster scheduling approach was proposed by [86]. In this approach, run-time migration (within different cores in the same cluster) for a task is allowed to improve resource utilization. The work in [113] studied the technique to utilize the parallelism in a hard real-time streaming application (represented as a Synchronous Data Flow (SDF) graph) in a clustered heterogeneous platform. Till date, considering both the intra-task parallelization and power minimization has received less attention. A greedy slack stealing algorithm is proposed in [130] that deals with task represented by graphs but did not consider the periodic DAGs. Assuming per-core DVFS, [37] provided the technique to combine DVFS and DPM. Considering the real-time jobs (represented as a DAG) in cloud computing systems and in a heterogeneous multi-core platform, the work in [117, 118] studied a QoS-aware and energy-efficient scheduling strategy. They proposed a scheduling policy that utilizes per-core DVFS. With the aim of improving energy-efficiency in a heterogeneous real-time platform, [115] proposed a combined approach considering the approximate computation and bin packing strategy. [103] investigated the energy awareness for cores that are grouped into blocks, and each block shares the same power supply scaled by DVFS. Benefits of (in terms of power saving) intra-task parallelism is proven theoretically in [100]. Considering the fork-join model, [101] reported an empirical evaluation of the power savings in a real test-bed. Based on level-packing, [124] proposed an energy efficient algorithm for implicit deadline tasks with same arrival time and deadline.

None of these works allows intra-task processor sharing considering the sporadic DAG task model, which is discussed in Chapter 3 and Chapter 4. Compared to the last two chapters, we consider significantly different settings (w.r.t. task model, platform, real-time constraints (deadlines), solution techniques, and the evaluation approach). *First*, in Chapter 3, we have considered a simplified model where only one DAG task executes at a time. In Chapter 4, we improve the energy savings by allowing inter-task processor sharing. However, both of these works assumed that the number of cores are unlimited. *Second*, both of these approaches in Chapter 3 and Chapter 4 assumed per-core speed scaling. However, many of the existing platforms (e.g., ODROID XU-3) do not support such speed scaling—speeds of processors under the same cluster must execute at the same speed. As the number of cores fabricated on a chip increases, per-core speed scaling design is less likely to be supported due to the inefficiency on hardware levels [70]. *Third*, both of these approaches have studied only the implicit deadline tasks and did not consider the *constrained* deadline tasks. Hence, the non-negligible idle gaps between the task deadline and its next release remain un-utilized. *Finally*, the evaluations in these approaches were done based on simulations without any implementation on a real platform.

5.3 Background and Existing Concepts

In this section, we describe some existing concepts and techniques for handling real-time parallel task scheduling, and that constitute an initial step for our proposed work.

Task Decomposition. The well-known task decomposition technique [109] transforms a parallel task τ_i into a set of sequential tasks as demonstrated in Figure 5.1(b). Upon task decomposition, each node $\mathcal{N}_i^l \in \tau_i$ is converted into an individual sub-task with its scheduling window (defined by its own release time and deadline) and execution requirement (c_i^l). The allocation of release time and deadline respect all the dependencies (represented by edges in the DAG). Considering that a task is allowed to execute on an unlimited number of cores, starting from the beginning, a vertical line is drawn at every time instant where a node $\mathcal{N}_i^l$ starts or ends. So the DAG is partitioned into several segments which may contain single/multiple thread(s). Threads assigned to the same segment share equal amount of execution length; e.g., $\mathcal{N}_i^3$, $\mathcal{N}_i^4$, and $\mathcal{N}_i^5$ all have 2-time units assigned to the 3^{rd} segment, as demonstrated in Figure 5.1(b).

Segment Extension. Task decomposition may put unnecessary restriction to a node's deadline, e.g., the decomposition of the DAG in Figure 5.1(a) restricts $\mathcal{N}_i^3$ within the 2^{nd} and 3^{rd} segment. To eliminate such unnecessary restriction and allow $\mathcal{N}_i^3$ to execute in the 4^{th} segment, segment extension should be applied, e.g., the green rectangle for $\mathcal{N}_i^3$ in the 4^{th} segment in Figure 5.1(b).

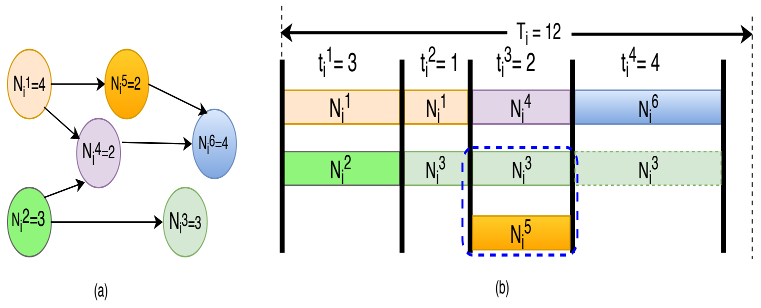


Figure 5.1: (a) A DAG task, τ_i (b) transformed DAG τ_i after applying task decomposition. Both of them are adopted from [67].

Intra-Task Processor Merging. After applying task decomposition and segment extension upon a DAG task τ_i, some of these cores (where τ_i is allocated) can be very lightly loaded. Those core cause massive leakage power consumption in the long run and should be avoided when necessary. Intra-task merging [67] seeks to merge those cores to gain overall energy efficiency by reducing the total number of active cores. For example, in Figure 5.1(b), the third core (executing $\mathcal{N}_i^5$) is lightly loaded, and thus it is better to merge all the execution into the second core and shut it off completely. Such a reduction on the number of active cores minimizes leakage power consumption as well as the total number of clusters.

5.4 Speed-Profile for Task and Cluster

This section discusses how different tasks share a cluster where all processors in a cluster execute at the same speed. When multiple tasks share a cluster, they may not align well due to sporadic releases and different periods. In a cluster-based platform, the processor having the maximum speed dominates the others in the same cluster. Hence, existing energy-saving techniques may perform poorly in a cluster-based platform. To tackle this problem, we propose a new concept called *speed-profile*. We provide the definition of speed-profile and its motivation in Subsection 5.4.1. In Subsection 5.4.2, we describe how speed-profiles are handled when two tasks are partitioned into the same cluster.

5.4.1 Speed-Profile for Each DAG

Interesting energy-saving techniques (e.g., segment extension) have been proposed in [67] for the implicit deadline tasks. For the constrained deadline tasks, this technique becomes incompetent because of the non-negligible idle gaps between the task deadline and its next release. For example, consider the task τ_i in Figure 5.1(b) with $D_i = 10$ and $T_i = 12$. Segment extension can stretch $\mathcal{N}_i^3$ to the end of the 4^{th} segment but cannot utilize the idle time of 2 units. Besides, the sub-optimal solution provided in [67] becomes *non-convex* (in a convex function, we can find the global maximum or minimum, for some variables of this function, which does not hold for a non-convex function) in a cluster-based platform (see Lemma 7).

Lemma 7. *In a cluster-based platform, the convex optimization problem constructed in Lemma 3* *(in Chapter 3) becomes non-convex.*

Proof. The following set of constraints ensure the real-time correctness for each node $\mathcal{N}_i^l \in \tau_i$, i.e., $\mathcal{N}_i^l$ receives enough time to finish execution within its scheduling window.

$$\forall l : \mathcal{N}_i^l \in \tau_i :: \sum_{j=b_i^l}^{d_i^l} t_j^c s_{i,j} \geq c_i^{\mathcal{N}_i^l}. \tag{5.1}$$

We introduce the following inequalities to bound the total length for all segments in task τ_i:

$$\sum_{j=1}^{Z} t_j^c \leq T_i. \tag{5.2}$$

Any value of execution speed and segment length ensures real-time correctness if Equation (5.1) and (5.2) are respected. However, the in Lemma 3, we have considered that the execution speed of a node, $\mathcal{N}_i^l$, is *constant* within its scheduling window (from b_i^l to d_i^l), and can be represented by a function of nodes execution requirement and its scheduling window. Besides, we have considered that a single DAG executes at a time, and, hence the execution speed of a node is not affected by the execution speed of other nodes (of other tasks). In this work, we consider the cluster-based platform, and the execution speed of a node depends on the execution speed of other nodes (of other tasks) in the same cluster. As a result, we cannot express the execution speed of a node as a function of its execution requirement, resulting in quadratic inequality constraints (Equation (5.1)). This makes the optimization problem *non-convex*. □

Due to the characteristics of a clustered platform, at each instant, all cores in a cluster must execute at the speed of the fastest one. If these tasks are not well aligned (concerning their execution speed), the cluster as a whole will perform poorly (w.r.t. energy efficiency). Assigning tasks with similar speed shape on the same cluster may not be an energy efficient option (due to their sporadic releases pattern). Figure 5.2 and Example 8 demonstrates one such scenario.

Example 8. *In this example, we describe how the sporadic arrival pattern of a task influences the energy efficiency of the whole cluster. Consider two tasks τ_1 and τ_2 with the predefined necessary speed of execution on two processors each, to be partitioned on to the same cluster (of four processors). In such a case, the resultant speed pattern (τ_{12}) of the cluster may vary for their (τ_1 and τ_2) different release offsets. This is because, the processors (of the same cluster) must run at the maximum/larger of the two individual speeds at each instant (to satisfy platform model restrictions while guaranteeing the correctness). Figure 5.2(a) shows the synchronous release case, where the whole cluster could run at 0 speed between [3,4) and [7,8). While Figure 5.2(b) shows the scenario when τ_1's initial release is delayed by one-time unit, where the whole cluster will need to run at a higher speed (of 0.8) most (75%) of the time and thus consumes more energy.*

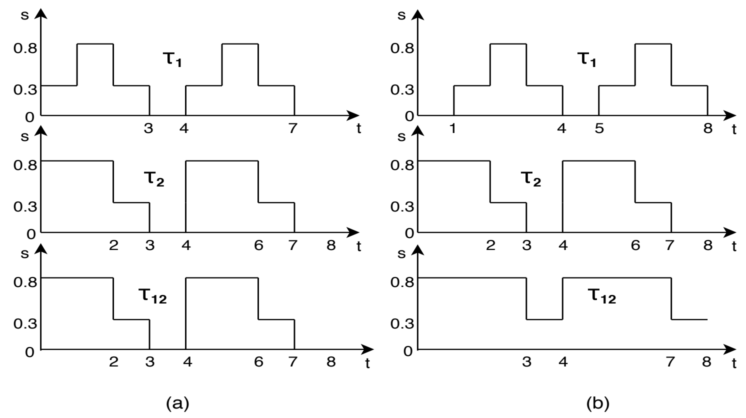


Figure 5.2: Impact of different release offset when multiple tasks share the same cluster.

In this example, from τ_2's perspective, direct energy reduction with existing per-task WCET based techniques may not help much, as it may be another task dominating the speed of the whole cluster most of the time. The critical observation is that, due to the extra restriction of the more realistic platform model, the speed of a cluster is determined by the heavier DAG running on it, as well as how synchronous are the releases, which could be entirely random. Moreover, even a task finishes *its execution early (say, τ_2 requires no execution over [5,7)), we may not be able to reduce the cluster speed at all.*

To address this issue, we propose a novel concept of speed-profile to capture the energy consumption behavior of all possible alignment scenarios.

Definition 1. *The **Speed-profile** of a task describes the percentage/likelihood of all possible speeds that the task may execute at over a period. It is a random variable $\mathcal{S}$ with an associated probability function (PF) $f_{\mathcal{S}}(s) = \mathbb{P}(\mathcal{S} = s)$, where s is a speed from the finite set of possible speeds, and $f_{\mathcal{S}}(s)$ represents the portions of the time (likelihoods) when it is running at speed s.*

Example 9. *Let us consider a task τ_i with $T_i = 15$ executing at a speed of 0.6 for 5 time units (not necessarily to be continual), and at a speed of 0.5 for the rest of the time. The speed-profile of the task is thus $\mathcal{S}_i = \begin{pmatrix} 0.6 & 0.5 \\ 5/15 & 10/15 \end{pmatrix} = \begin{pmatrix} 0.6 & 0.5 \\ 0.33 & 0.67 \end{pmatrix}$. At any specific time, t, there is about 33% probability that the cores are running at the speed of 0.6 unit and about 67% probability that the cores are running at the speed of 0.5 unit.*

It is evident that from another task's point of view, the speed-profile provides probabilistic information on how the task of interest would restrict the lower bound to the speed of the cluster over time. As the alignment of releases between any two tasks is unknown, we assume in the future analysis that any phase difference is of equal chance over the long run.

Remark 6. *The speed-profile $\mathcal{S}_i$ of a given task τ_i remains the same for an initial phase (release offset) $\phi_i \geq 0$. Regarding inter-task combinations, we assume uniform distribution for the phase of any task; i.e., $\phi_i \sim U[0, T_i)$.*

Subsection 5.5.1 details the calculation for task speed-profile. Here, we describe the calculation of the cluster speed-profile when two tasks are combined on to the same cluster.

5.4.2 Speed-Profile for the Cluster Containing Multiple DAGs

As stated earlier, the property of the clustered platform and sporadic arrival pattern of a task makes the exact speed of the cluster unpredictable at a specific time instant (see Figure 5.2 and Example 8). As a result, when two tasks τ_i and τ_j (with speed-profiles) are being considered allocating to the same cluster, we need to construct the merged speed-profile of the cluster (executing them both). To perform such calculation, we introduce a special $\odot$ operator that takes the maximum of the two profiles on a probability basis[1].

Definition 2. *The special operator $\odot$ operates on two (or more) random variables $\mathcal{X}$ and $\mathcal{Y}$. During this operation, each entry $\mathcal{X}_i \in \mathcal{X}$ is compared with each entry $\mathcal{Y}_j \in \mathcal{Y}$ and the value $\mathcal{Z}_{ij}$ is calculated as $\mathcal{Z}_{ij} = max(\mathcal{X}_i, \mathcal{Y}_j)$, with a combined (multiplied) probability. If there are multiple occurrences of an entry, all of them are merged into a single entry, and their associated probability are summed together.*

Example 10. *Let* $\mathcal{S}_i = \begin{pmatrix} 6 & 5 \\ 0.4 & 0.6 \end{pmatrix}$, $\mathcal{S}_j = \begin{pmatrix} 6 & 2 \\ 0.4 & 0.6 \end{pmatrix}$. *Then* $\mathcal{S}_i \odot \mathcal{S}_j = \begin{pmatrix} 6 & 6 & 6 & 5 \\ 0.16 & 0.24 & 0.24 & 0.36 \end{pmatrix} = \begin{pmatrix} 6 & 5 \\ 0.64 & 0.36 \end{pmatrix}$.

Note that we allocate two different DAGs (with same/different periods) to the same cluster. The speed-profile indicates how long a DAG executes at different speeds within its deadline, i.e., the probability that a DAG executes at a specific speed. The task's period becomes irrelevant as speed-profile is a probability-based measurement. Once τ_i and τ_j are allocated to the same cluster, we use $\mathcal{S}_{ij}$ to denote the speed-profile of the cluster (see Example 10).

In summary, energy minimization in a cluster-based platform is challenging because of sporadic

[1] Although the appearance of the proposed operator is identical to [89], the calculation is quite different. This is due to the "larger value dominating" nature of the platform model considered in this paper.

release pattern and the idle gaps between a task deadline and its period. To tackle these problems, we have introduced the concept of speed-profile for both an individual task and a cluster where multiple tasks can be allocated.

5.5 Task Partitioning Algorithm

The ultimate goal of the paper is to partition all DAGs into clusters, such that overall platform energy consumption is minimized. Recall that on a clustered multiprocessor platform, at a given instant, all processors in the same cluster must execute at the same speed. Due to this property of a cluster-based platform, if two tasks that are not well-aligned (in terms of execution speed) are allocated to the same cluster, it will result in reduced energy efficiency. So, we have proposed the concept of speed-profiles (refer to Section 5.4) which is a tool to measure the potential long-term energy saving of a cluster when partitioning any pair of DAGs into this cluster. So far we have discussed the importance of the concept of speed-profile but did not mention how to create them given a DAG task, which is the focus on Subsection 5.5.1. In Subsection 5.5.2, we describe the task-to-cluster partitioning algorithm.

5.5.1 Creating the Speed-Profile of a Task

Given a DAG task τ_i, we provide two approaches to create the speed-profile $\mathcal{S}_i$.

Approach A: Considering the Maximum Speed from all the Cores. Upon applying the task decomposition, segment extension, and intra-task processor merging techniques (Section 5.3), some vital information (e.g., the speed of a core at a specific time and number of cores required) becomes available. This information plays a role to calculating the speed-profile S_i of task τ_i. At any time instant t, we consider the maximum speed from all the cores available. It ensures the sufficient speed so that even the heaviest node can finish execution within its scheduling window (defined after task decomposition). We consider constrained deadline (i.e., $D_i \leq T_i$), so the task must have to finish by D_i and rest of the time is an idle slot. For each segment $j \in \tau_i$, (summation of the length of these segments is equal to D_i), we create a pair $(s_{i,j}, p_{i,j})$. For the j^{th} segment, $s_{i,j}$ and $p_{i,j}$ respectively denote the maximum execution speed and the probability that the cluster will run at this speed. Let, M cores are allocated to τ_i. At j^{th} segment, we calculate $s_{i,j}$ and $p_{i,j}$ as follows:

$$s_{i,j} = \max_{k \leq M}\{s_{i,j,k}\}, p_{i,j} = \frac{t_j^c}{T_i}.$$

Here, $s_{i,j,k}$ denotes the speed of k^{th} core at j^{th} segment and t_j^c is the length of j^{th} segment. The speed-profile $\mathcal{S}_i$ will be:

$$\mathcal{S}_i = \begin{pmatrix} s_{i,1} & s_{i,2} & \cdots & s_{i,z} & 0 \\ p_{i,1} & p_{i,2} & \cdots & p_{i,z} & (T_i - D_i)/T_i \end{pmatrix}.$$

The last pair reflects the fact that the core remains idle for the $(T_i - D_i)$ time units at the end of each period.

Example 11. *Consider a task τ_i with $T_i = 15$, $D_i = 12$ and $C_i = 6.5$. Let, the task is partitioned into three segments of length 5, 7 and 3 time units respectively, where the processor is executing at a (maximum) speed of 0.6 in the first segment, speed of 0.5 in the second segment, and remain idle in the third segment. The speed-profile is:*

$$\mathcal{S}_i = \begin{pmatrix} 0.6 & 0.5 & 0 \\ 0.33 & 0.47 & 0.2 \end{pmatrix}$$

Note that, if a cluster contains a single task τ_i, then $\mathcal{S}_i$ also represents the cluster speed-profile. If τ_i and τ_j (or more tasks) are executing on the same cluster, then the technique described in

Subsection 5.4.2 needs to be applied before making any choices. The greedy choosing approach for task partition is detailed in Subsection 5.5.2.

Approach B: A Single Speed Throughout. Theorem 2 of Chapter 3 shares a valuable insight: *The total energy consumption (assuming processor remains on) is minimized in any scheduling window when execution speed remains uniform (the same) throughout the interval.* Motivated by it[2], we propose another approach of selecting a single speed for a DAG task (job) during the whole duration from its release until its deadline.

In this approach, we consider the maximum workload (or the execution requirement) from all the cores available and determine the aggregated workload. Upon dividing the aggregated workload by the deadline, we get the desired single speed. Let M cores be allocated to task τ_i. At j^{th} segment, the execution requirement of the k^{th} core is denoted by $w_{i,j,k}$, which is calculated as follows:

$$w_{i,j,k} = s_{i,j,k} \times t_j^c.$$

We determine the maximum execution requirement as follows:

$$w_{i,j} = \max_{k \leq M} \{w_{i,j,k}\}.$$

Let Z denotes the total number of segments in τ_i. The maximum total workload w_i and the desired single speed s_i is calculated using the following equations:

$$w_i = \sum_{j=1}^{Z} w_{i,j}, \quad s_i = \frac{w_i}{D_i}. \tag{5.3}$$

[2]Theorem 2 considered that the speed remains constant within a scheduling slot for each processor. Also, they assumed per core speed scaling and calculated the speed within each scheduling slot through a convex optimization method. This paper considers the clustered platform where the objective function becomes *non-convex* (see Lemma 7) and thus the existing approach is inefficient.

Other than the idle pair $(0, (T_i - D_i)/T_i)$, we consider a single speed throughout the deadline so only a single pair (s_i, p_i) is required, where $s_i = w_i/D_i$ and $p_i = D_i/T_i$.

Example 12. *Consider the task described in Example 11 ($T_i = 15$, $D_i = 12$ and $C_i = 6.5$). It must finish 6.5 unit of workloads within 12-time units. Using this approach its speed-profile is:*

$$\mathcal{S}_i = \begin{pmatrix} 0.54 & 0 \\ 0.8 & 0.2 \end{pmatrix}.$$

Lemma 8. *If a task τ_i executes according to the speed-profile $\mathcal{S}_i$, it guarantees real-time correctness.*

Proof. We have shown in Chapter 3 that the following constraint guarantees the real-time correctness:

$$\forall l : \mathcal{N}_i^l \in \tau_i :: \sum_{k=b_i^l}^{d_i^l} t_k^c S_k^{\mathcal{M}_i^l} \geq c_i^{\mathcal{N}_i^l}. \tag{5.4}$$

Here, b_i^l and d_i^l denotes the release time and deadline of $\mathcal{N}_i^l$, $\mathcal{M}_i^l$ denotes the node-to-processor mapping and $S_k^{\mathcal{M}_i^l}$ is the speed of the processor (where $\mathcal{N}_i^l$ is allocated) at k^{th} segment. Here, at any time instant t, we choose either the maximum speed from all the cores running on the same cluster (Approach A) or a single speed that can guarantee the maximum execution requirement for the whole duration up to τ_i's deadline (Approach B). So, at any time instant, the cluster speed is larger or equals to the speed of any individual core. Considering Equation (5.1) and (5.4) we can deduce that:

$$\forall l : \mathcal{N}_i^l \in \tau_i :: \sum_{k=b_i^l}^{d_i^l} t_k^c s_{i,k} \geq \sum_{k=b_i^l}^{d_i^l} t_k^c S_k^{\mathcal{M}_i^l} \geq c_i^{\mathcal{N}_i^l}.$$

So, we conclude that Executing a task with speed according to the speed-profile $\mathcal{S}_i$ guarantees real-time correctness. □

An Efficient Approach for Implicit Deadline System. By adopting simple modification in Equation (5.3), it is possible to apply the process mentioned above for the implicit deadline tasks also. The workload w_i should be divided by the period instead of the deadline. We consider the same speed through the task period, so only a single pair (s_i, p_i) is required, where $s_i = w_i/T_i$ and $p_i = 1$.

Example 13. *Now we create the speed-profile for the task described in Example 11 and 12 considering implicit deadline. So it has $T_i = D_i = 15$ and $C_i = 6.5$. Let's assume that it is executed at a speed of 0.6 for 5-time units, at a speed of 0.35 for 10-time units. According to Approach A, the speed-profile is:*

$$\mathcal{S}_i = \begin{pmatrix} 0.6 & 0.35 \\ 0.33 & 0.67 \end{pmatrix},$$

and according to Approach B, the speed-profile is:

$$\mathcal{S}_i = \begin{pmatrix} 0.43 \\ 1 \end{pmatrix}.$$

5.5.2 *Task Partition: Greedy Merging with Speed-Profiles*

We are now equipped with tools (speed-profiles) to measure the potential long-term energy saving of a cluster when partitioning any pair of DAG tasks into it. This subsection describes the scheme for selecting pair by pair so that the total number of clusters can be significantly smaller than the total number of tasks.

To select a (task) pair that will share the same cluster, we greedily choose the pair that provides maximum power saving, as depicted in Algorithm 2. Note that we allow the pairing of two DAGs that are not merged previously. Also, if any task uses more cores than what is available in a cluster, that task cannot be merged with that cluster. Algorithm 2 creates two empty lists $\bar{\mathcal{S}}$ and $\tilde{\mathcal{S}}$ that will

Algorithm 2: Greedy Merging

```
1: **Input:** Task-set $\tau$, with speed-profile $\mathcal{S}_i$ (computed using approach A or approach B) for
   each task
2: **Output:** Speed-profile $\tilde{\mathcal{S}}$ (with processor power saving).
3: $\bar{\mathcal{S}}, \tilde{\mathcal{S}} \leftarrow \emptyset$        $\triangleright$ All the possible/selected speed-profiles;
4: **for** $i = 1$ to $n$ **do**
5:     **for** $j = i + 1$ to $n$ **do**
6:         $\mathcal{S}_{ij} \leftarrow \mathcal{S}_i \odot \mathcal{S}_j$; $\bar{\mathcal{S}} \leftarrow \bar{\mathcal{S}} \cup \mathcal{S}_{ij}$;
7:     **end for**
8: **end for**
9: **while** $\exists \mathcal{S}_{xy} \in \bar{\mathcal{S}}$ and $\mathcal{S}_{xy}$ provides non-zero power saving **do**
10:    $\mathcal{S}_{xy} \leftarrow$ the pair from $\bar{\mathcal{S}}$ with maximum power saving;
11:    $\tilde{\mathcal{S}} \leftarrow \tilde{\mathcal{S}} \cup \mathcal{S}_{xy}$
12:    **for** $k = 1$ to $n$ **do**
13:        $\bar{\mathcal{S}} \leftarrow \bar{\mathcal{S}} - \mathcal{S}_{kx} - \mathcal{S}_{xk} - \mathcal{S}_{ky} - \mathcal{S}_{yk}$;
14:    **end for**
15: **end while**
16: **return** $\tilde{\mathcal{S}}$.
```

contain all the possible and selected speed-profiles (Line 3). Lines 4–8, calculate all the possible speed-profiles and insert them into $\bar{\mathcal{S}}$. We greedily select a pair of DAGs that provide the maximum power saving (calculated using Equation (6.1) and Equation (10) from [67]) and update the list $\bar{\mathcal{S}}$ by removing the pair from any further merging (Lines 9–15). The list $\tilde{\mathcal{S}}$ is also updated by adding the selected pair (Line 11). We conclude by returning the updated list $\tilde{\mathcal{S}}$ (Line 16).

Theorem 9. *Executing a task with a speed according to the cluster speed-profile guarantees real-time correctness.*

Proof. We have shown in Lemma 8 that a task τ_i will not miss the deadline if executed according to its speed-profile $\mathcal{S}_i$. If τ_i share a cluster with another task τ_j and executes according to the merged (i.e., cluster) speed-profile $\mathcal{S}_{ij}$, then it still guarantees the real-time correctness, because $\mathcal{S}_{ij} \geq \mathcal{S}_i$ holds at any time instant. $\square$

Remark 7. *For n tasks, the time complexity to generate all possible speed-profiles, $\bar{\mathcal{S}}$, is $O(n^2 Z)$, where Z is the maximum number of segments of all DAGs in the set after decomposition (related to the structure and size of the DAGs). Algorithm 2 greedily choose a speed-profile by iterating through $\mathcal{S}$ and then update, which takes $O(n^2)$ time as well. Thus the total complexity of the proposed method is $O(n^2)$.*

In summary, we have proposed two methods (Subsection 5.5.1) to create the speed-profile for a constrained-deadline DAG. We also show that if a task executes according to the speed-profile, it ensures real-time correctness. According to the techniques provided in Section 5.4, we could evaluate and compare all potential pairs of the combination by calculating the cluster speed-profile after merging. Finally, Subsection 5.5.2 discussed how to use these speed-profiles to find suitable partners to share a cluster greedily.

5.6 System Experiments

In this section, we present experimental results conducted on an ODROID XU-3 board. The platform runs on Ubuntu 16.04 LTS with Linux kernel 3.10.105. It is fabricated with Samsung Exynos5422 Octa-core SoC, consisting of two quad-core clusters, one 'LITTLE' cluster with four energy-efficient ARM Cortex-A7 and one 'big' cluster with four performance-efficient ARM Cortex-A15. Four TI INA231 power sensors are integrated onto the board to provide real-time power monitoring for the A-7 and A-15 clusters, GPU, and DRAM. An energy monitoring script, `emoxu3` [50], is used to log energy consumption of the workloads.

DAG Generation. In this experiment, we generate two task sets each with 300 DAGs, and use the widely used Erdos-Renyi [42] method to generate a DAG. We tune a parameter p, that denotes the probability of having an edge between two nodes. In this experiment, we set p to 0.25 generate DAGs with an uncomplicated structure. If a disconnected DAG is generated, we add the fewest number of edges to make it connected. For experimentation, we have considered arbitrary task periods, and it is determined using Gamma distribution [58]. We set the periods with $T_i = L_i + 2(C_i/m)(1+\Gamma(2,1)/4)$ [109]. Here, L_i is the critical path length of τ_i, calculated according to the definition of L_i (refer to Section 2.1.1).

After generating the topology of each DAG of a set, we partition them into two subsets according to the proposed approach, one to the "big" and the other one to the "LITTLE" cluster, and measure the energy consumption over the hyper-period of all DAGs. We use `rt-app` [107] to emulate the workload for each node. rt-app simulates a real-time periodic load and utilizes the POSIX threads model to call and execute threads. For each thread, an execution time needs to be assigned. In this experiment, for each node, we randomly select an execution time ranged between $[300ms, 700ms]$. rt-app itself has a latency that varies randomly between $13 - 150ms$ per thread. Therefore, we add the maximum latency of rt-app, i.e., 150*ms*, to the execution time of each thread from an analytical point of view.

DAG Scheduling. We use the Linux built-in real-time scheduler `sched_FIFO` to schedule the DAGs. Compared to the other system tasks, DAGs are assigned with higher priorities so that they can execute without interference. Our approach is also applicable to other preemptive schedulers which feature the work-conserving property.

Frequency Scaling. According to the frequency/speed-profile (Section 5.5), we use `cpufreq-set` program (from `cpufrequtils` package) to change the system's frequency online. We use the ODROID XU-3 board, where scaling-down (up) the frequency of the big cluster takes at most $60\ (40)ms$, respectively. On the LITTLE cluster, both the operation takes at most $15ms$. Due to

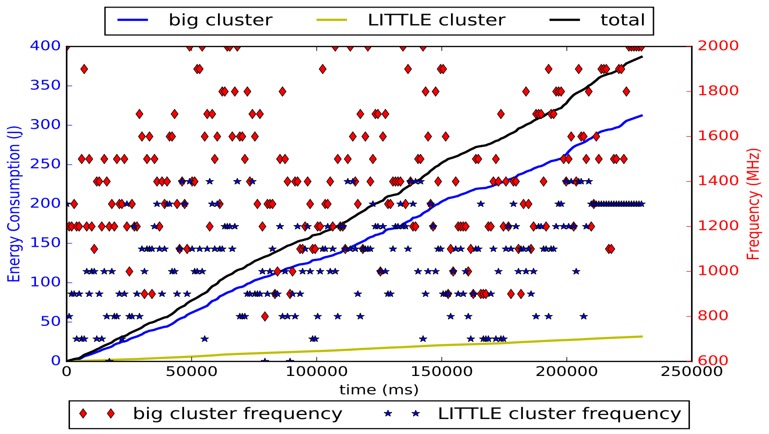


Figure 5.3: The energy consumption and the frequency variation of our proposed approach on ODROID XU-3.

this delay, the hyper-period of all DAGs becomes large (230s, in this experiment). We detail the reasons behind this delay in Section 5.8.2.

The Reference Approach. Since no work has studied the same problem considered in this paper, we do not have a direct baseline for comparison. So, we propose a reference approach based on the studies for energy-efficient scheduling of sequential tasks [38]. They assigned an operational frequency to each task, and at run-time, schedule them according to their frequency. In this reference approach, we compute an operational frequency for each DAG. This frequency stretches out execution length of these DAGs as much as possible without violating their deadlines. As stated earlier, the reference approach executes the DAGs with the same partition, but without the merging techniques proposed in Section 5.5.

Results. The experimental results are plotted in Figure 5.3 and 5.4. In these figures, we show (i) the energy consumption over the hyper-period (230s), where the three lines show the energy consumption of the big and LITTLE cluster, and the total system; and (ii) frequency variation during the run-time, where the diamond and star marks denote the operational frequency of the big

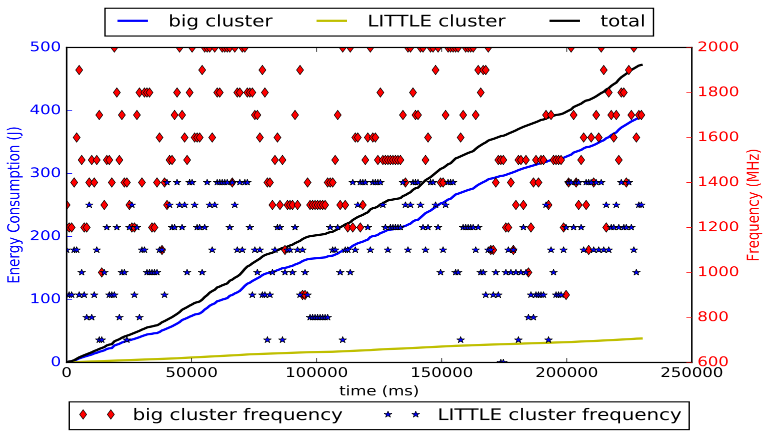


Figure 5.4: The energy consumption and the frequency variation of the reference approach on ODROID XU-3.

Table 5.1: Summary of experimental results.

	Ours (J)	Ref (J)	Energy Saving $(\%)$
big cluster	312	389	20
LITTLE cluster	32	38	16
Total	387	472	18

and the LITTLE cluster at a specific time instant, respectively. Note that the GPU and DRAM also contribute the energy consumption of the total system. Hence, the total energy consumption is a bit higher than the summation of the contribution of the big and the LITTLE cluster, but it is observed that there is a negligible difference for the energy consumption of GPU and DRAM between the two approaches. Besides, it is worth noticing that this energy consumption also accounts for energy consumption of the operating system.

Table 5.1 summarizes the comparison of the experimental results, where the energy consumption of the two clusters and the total system is presented, and the energy saving from our approach is given. As can be seen, our approach consumes 312J and 32J on the big and the LITTLE cluster, respectively. Comparing to the reference approach, we save energy consumption by 20% and 16%.

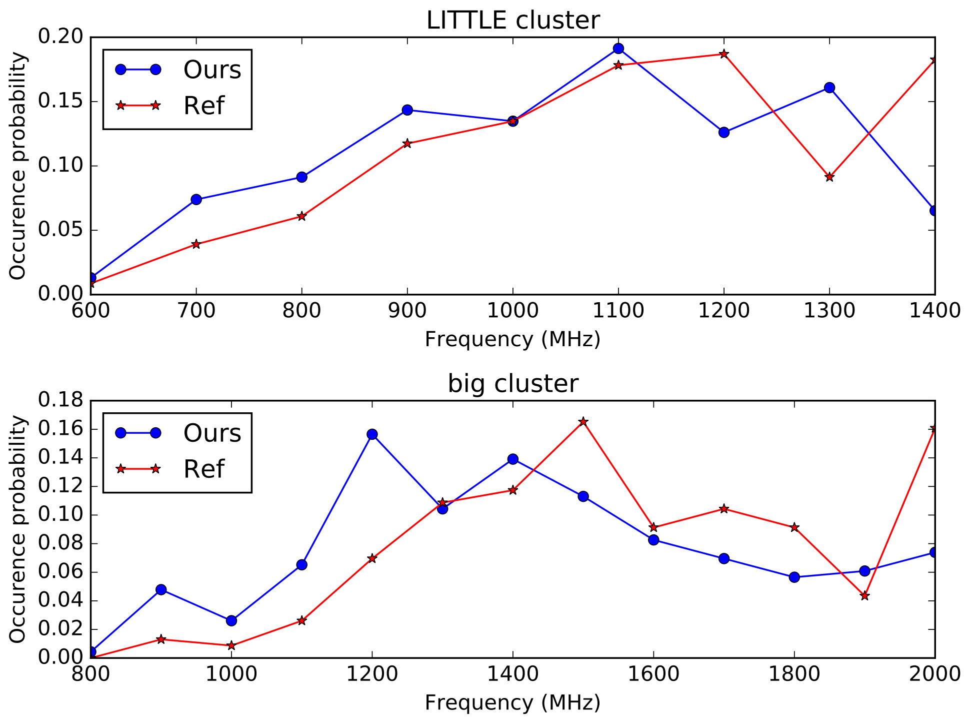


Figure 5.5: Frequency occurrence probabilities.

In total, our approach saves energy consumption by 18%.

The result can be justified as the reference approach changes the frequency for each DAG, while ours have a fine-grained frequency adjustment at each segment (Section 5.5.1), and could scale down the frequency if required. Figure 5.5 presents the frequency occurrence probability of two clusters which is recorded per second by emoxu3. We observe that within the same time interval the reference approach has a higher probability to execute at a higher frequency, while our approach is more likely to execute at the lower frequencies, thus reducing the energy consumption.

Remark 8. *Each heavy DAG ($C_i > T_i$) needs two or more cores while executing and the ODROID XU-3 board contains four cores per cluster. So, in this experimental setup, we can not execute more than four heavy DAGs at a time. Such a restriction is not applicable to the light DAGs ($C_i \leq T_i$). We also consider that a heavy DAG cannot be allocated in multiple clusters.*

5.7 Simulations

For large-scale evaluation, we perform simulations and compare the results with existing baselines. We generate DAGs using the Erdos-Renyi method (Section 5.6). We consider two types of task periods; *(a) harmonic periods*, where the task period T_i is enforced to be an integral power of 2. We define T_i as $T_i = 2^\alpha$, where α is the minimum value such that $2^\alpha \geq L_i$, where L_i is the critical path length of τ_i *(b) arbitrary periods*, T_i is determined using Gamma distribution (Section 5.6).

We compare our approaches with some existing baselines studied in [67, 130, 37]. Total power consumption by our approach and by these baselines are calculated using Equation (6.1). As mentioned earlier, [67] considered per-core DVFS, i.e., each core individually is an island of the cluster-based platform. For a fair comparison, according to the scheduling policy of [67], when a task is allocated on some cores at any time instant t, we choose the maximum speed among all these cores. We consider [67] as a baseline because that work is closely related to ours. Although they have considered per-core DVFS and restrict their attention only to implicit deadline tasks, the task and the power model are same. Besides, although this work and [67] propose different approaches to power saving, the initial (preparation) steps of both approaches are based on commonly known techniques like task decomposition, task merging, etc.

The work in [130] studied a greedy slack stealing (GSS) scheduling approach considering inter-dependent sequential tasks. It considered the DAG model to represent dependencies among the tasks. In GSS, the *slack* (unused time in actual computation requirement of a task) is reclaimed by one task by shifting others towards the deadline. They did not consider repetitive tasks; hence it can be regarded as scheduling a single task. Besides this, the power and graph model used in [130] is different from ours. To ensure a fair comparison, we execute the GSS algorithm using the power model in Equation (2.1) and assume that once introduced in the system; a processor remains active. We also consider a minimum inter-arrival separation for a DAG. That work considered three different kinds of nodes: *AND, OR*, and *Computation* nodes (Subsection 2.1 in [130]). A computation node has both the maximum and average computation requirement. To comply with our work where the focus in energy reduction while guaranteeing worst-case temporal correctness, we execute the GSS algorithm considering only the computation nodes with their maximum computation requirement. We made all the changes in order to provide a fair comparison. Despite these differences, we chose [130] as a baseline because they studied a *GSS* approach for energy minimization. They considered the inter-dependent sequential tasks and their dependencies was represented by a DAG, which is similar to our task model. We compare power consumption by varying two parameters for each task: *task periods (utilization)* and *the number of nodes*. We randomly generate 25 sets of DAG tasks and compare the average power consumption.

Notations of Referenced Approaches. For the task partitioning step, either we randomly choose any two and allocate them to the same cluster, or greedily choose the ones with lowest speed as proposed. Regarding speed-profile calculation, there are also two options (Approaches A and B in Section 5.5.1). Combining these options in two steps lead to four baselines: *MaxSpeed_Greedy*, *SingleSpeed_Greedy*, *MaxSpeed_Random*, *SingleSpeed_Random*. Also, three baselines mentioned above are included for comparison:

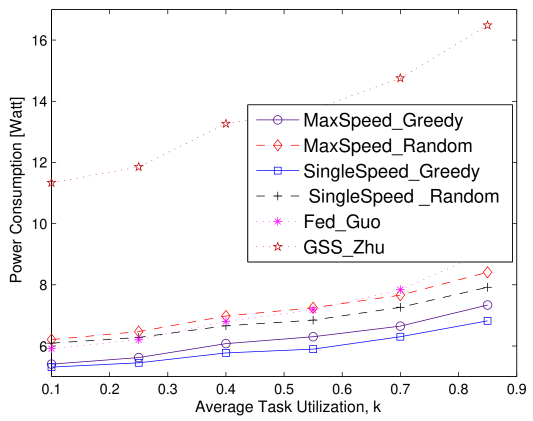
• Federated scheduling with intra-task processor merging [67], denoted by *Fed_Guo*;

• GSS algorithm [130], denoted by *GSS_Zhu*.

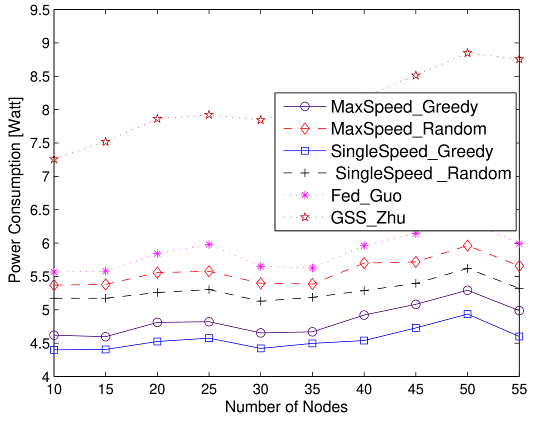
5.7.1 Identical Heterogeneous Platform with a Continuous Frequency Scheme

In this section, we report the power consumption comparison considering the identical heterogeneous platform and a continuous frequency scheme. In this platform, both the ”big” and the ”LITTLE” clusters share the same power model as described in Equation (2.1).

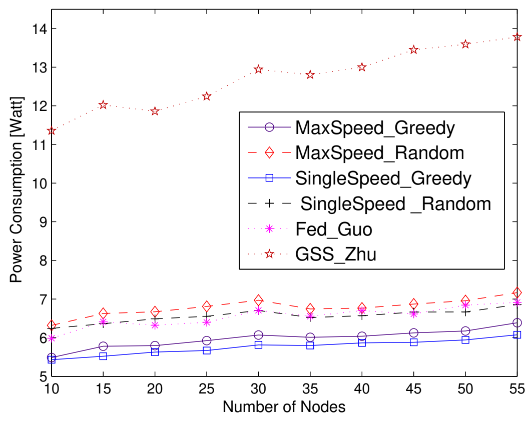
5.7.1.1 Constrained Deadline Task



(a) Comparison of power consumption with different approaches for DAGs with a fixed number of nodes as 30.



(b) Average power consumption comparison with different approaches for tasks with harmonic periods.



(c) Average power consumption comparison with different approaches for tasks with arbitrary periods.

Figure 5.6: Power consumption comparison between different approaches for the **constrained** deadline tasks considering a **continuous** frequency scheme on the **identical** heterogeneous platform

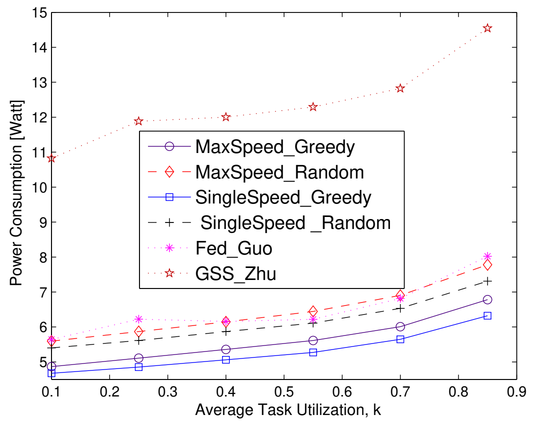
In this Section, we consider the constrained deadline tasks and report their average power consumption under varying task period (or utilization) and the number of nodes.

Effect of Varying Task Periods (utilization). Here, the number of nodes is fixed to 30. We vary the period in a range ($L_i \leq T_i \leq C_i$). The parameter L_i and C_i are measured once the DAG is generated according to the technique described in Section 5.6. We also use the following equation (similar to Chapter 3) to ensure that the value of T_i satisfies the range ($L_i \leq T_i \leq C_i$).

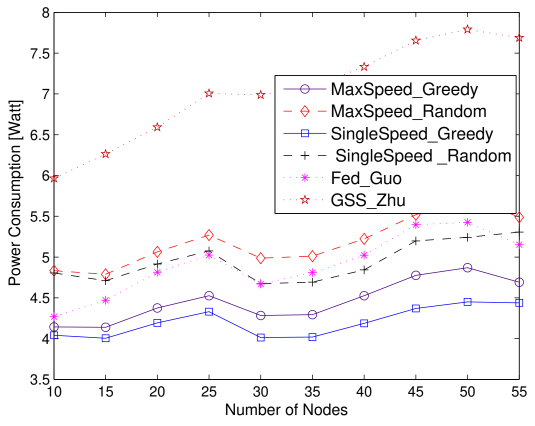
$$T_i = L_i + (1 - k)(C_i - L_i) \tag{5.5}$$

Here, $k \in$ [0, 1] is task utilization. As we are considering the constrained deadline tasks, D_i is randomly picked from the range ($L_i \leq D_i \leq T_i$). We observe that the average energy consumption is directly proportional to the average task utilization. Figure 5.6(a) shows that *SingleSpeed_Greedy* approach outperforms the others and leads to a power savings of at least 16.67% and 56.24% compared to the *Fed_Guo* and *GSS_Zhu* approaches.

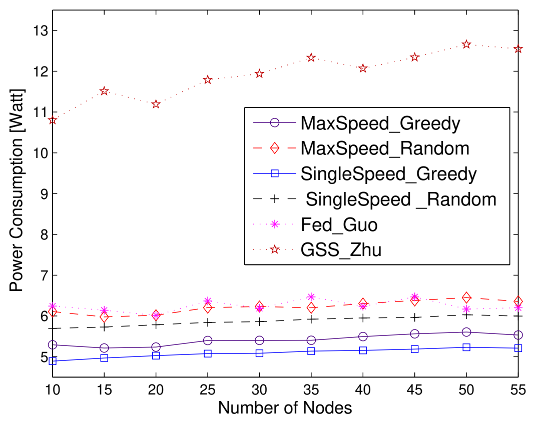
Effect of Varying the Numbers of Nodes. Here we vary the number of nodes (T_i remains fixed) and report the average power consumption. We consider both harmonic and arbitrary periods (reported in Figures 5.6(b) and 5.6(c)). For both of these settings, we randomly generate 100 tasks; and vary the number of nodes in each task between 10 and 55. Compared to the previous set of experiments (varying task utilization with a fixed number of nodes), we observe similar improvements in power consumption, i.e., choosing a single speed over the whole deadline leads to more power savings. Especially, when considering harmonic task periods the *SingleSpeed_Greedy* approach uses on average 22.25% and 43.56% less power compared to *Fed_Guo* and *GSS_Zhu* approaches, respectively. If we consider arbitrary task periods, the savings become 12.39% and 54.57%, respectively.



(a) Comparison of power consumption with different approaches for DAGs with a fixed number of nodes as 30.



(b) Average power consumption comparison with different approaches for tasks with harmonic periods.



(c) Average power consumption comparison with different approaches for tasks with arbitrary periods.

Figure 5.7: Power consumption comparison between different approaches for the **implicit** deadline tasks considering a **continuous** frequency scheme on the **identical** heterogeneous platform

5.7.1.2 Implicit Deadline Task.

Now we consider the Implicit deadline tasks and show their average power consumption by changing two parameters: task period (or utilization) and the number of nodes.

Effect of Varying Task Periods (Utilization). In this experiment, we fix the number of nodes to 30. We report the average power consumption in Figure 5.7(a). Similar to the phenomenon we observed in the last experiment, average energy consumption for implicit deadline tasks is directly proportional to the average task utilization. Figure 5.7(a) also shows that adopting the *SingleSpeed_Greedy* approach results in reduced power consumption. On average, the *SingleSpeed_Greedy* approach leads to a power saving of at least 18.44% and 57.3% compared to *Fed_Guo* and *GSS_Zhu* approaches, respectively.

Effect of Varying the Numbers of Nodes. Now we measure the average power consumption by varying the number of nodes with fixed T_i. We randomly generate 100 tasks with harmonic and arbitrary deadline. We report the average power consumption in Figure 5.7(b) (or Figure 5.7(c)) for harmonic (or arbitrary) deadline tasks. Again, we observe similar improvements in power consumption, i.e., choosing a single speed over the whole deadline outperforms other approaches. Instead of period, we use the term deadline because we are considering constrained deadline tasks. Specifically, under harmonic task periods, the *SingleSpeed_Greedy* incurs 14.05% and 40% less power on average compared to *Fed_Guo* and *GSS_Zhu*; under arbitrary task periods, the savings potential are 18.39% and 57.27%, respectively.

5.8 Discussions: Assumptions and Applicability

In this section we discuss the assumptions adopted in this paper. Subsection 5.8.1 discusses the validity of these assumptions, their impacts and potential solutions to overcome these impacts. Then, in Subsection 5.8.2 we detail the reasons behind the measurement overheads and the applicability of our proposed approaches.

5.8.1 Assumptions Behind the Power Model

Components Behind the Overall Power Consumption. While some other factors such as cache miss, bus accesses, context-switches, and I/O usage also affect the power consumption, CPU power consumption is one of the major contributors to the overall power consumption. Power consumption may largely be dominated by any of these factors depending on the application/benchmark (e.g., power consumption is dominated by the radio/network in some communication-oriented applications [47]). In this work, we target to minimize the CPU power consumption only. While minimizing the CPU power consumption, our approach does not increase the power consumption that is influenced by other factors, as our technique does not introduce additional existing DAG schedulers (DAG decomposition based). It would build the foundation for more complicated analysis that considers all other factors of overall power consumption in the future.

Dynamic Power Management. DPM explores idle slot of a processor and puts the processor to a low power mode to reduce the static power consumption. Switching to low power mode (and backward) incurs additional energy consumption and is beneficial only when the idle slot is longer than a threshold, known as the *break-even time (BET)* [3][37]. In this paper, the available idle slot may not be longer than the BET for two reasons. First, we focus on clustered multiprocessor platform, where processors within each cluster must execute at the same speed, i.e., the maximum speed necessary for the demand on each processor at a given instant. As a result, unless all processors within a cluster are idle, the cluster cannot be switched into any sleep mode. For sporadic releases, idle slots of each processor are unlikely to be synchronized. Moreover, cluster-wide idle slots tend to be relatively short. Second, while executing a task, a uniform execution speed significantly reduces the overall energy consumption (Theorem 2), which is the goal of our proposed approaches—this leads to further reduction of idle slots. For example, a study using Intel Strong ARM SA-1100 processor has shown a transition time of 160*ms* to switch from sleep mode back

[3]BET is the minimum duration for the processor to stay at the sleep mode.

to run mode [128], which can be larger than many task periods in avionics. As triggering mode switches becomes more energy consuming in general and may overwhelm the gain in energy savings, DPM is considered out of the scope of this work. DPM could be a valid option under certain scenario, and we leave the further exploration along this direction as future work.

5.8.2 A Note on the Overhead Delay

We have mentioned that the scaling-down (up) the frequency of the big cluster takes at most $60\ (40) ms$, while both these operations take at most $15ms$ on the LITTLE cluster (see Section 5.6). we use cpufreq-set module to change the system's frequency, and this module accounts for microsecond-scale transition delay (usually 100-200μs), which is typically incorporated into the WCET. In our case, the delay is much higher because (i) we used a Python script to measure the delay; and (ii) there is some user-level delay caused by I/O operations and file logging, e.g., time-stamp storage before and after each run. Time-stamp storage detects the arrival and completion of nodes which could be avoided when one does not need to track system behavior in a precise manner (which is the normal scenario). Considering the potential overhead issue, in Subsection 5.5.1, we proposed Approach-B, where a task executes at a single speed (so, there is no frequency changing overhead) for the whole duration from its release to the deadline. Experimental study (Section 5.7) also shows excellent performance of such approach when WCETs of sub-jobs are short. Note that, we can not entirely avoid the speed changing overhead for Approach-A in Subsection 5.5.1. However, we can reduce the number of frequency changes by partitioning the tasks (into a cluster) according to Algorithm 2. Thus, an efficient partitioning can reduce the frequency changing overhead.

5.9 Conclusion

In this chapter, we have studied real-time scheduling of a set of implicit and constrained deadline sporadic DAG tasks. We schedule these tasks on the cluster-based multi-core platforms with the goal of minimizing the CPU power consumption. In a clustered multi-core platform, the cores within the same cluster run at the same speed at any given time. Such design better balances energy efficiency and hardware cost and appears in many systems. However, from the resource management point of view, this additional restriction leads to new challenges. By leveraging a new concept, i.e., *speed-profile*, which models energy consumption variations during run-time, we can conduct scheduling and task-to-cluster partitioning while minimizing the expected overall long-term CPU energy consumption.

We have presented the experimental result performed on an ODROID XU-3 board to demonstrate its feasibility and practicality. We have also presented our system experiments on a larger scale through realistic simulations that demonstrate an energy saving of up to 57% through our proposed approach compared to existing methods.

CHAPTER 6: ENERGY-EFFICIENT PARALLEL REAL-TIME SCHEDULING ON CLUSTERED MULTI-CORE: ADAPTING THE FREQUENCY DISCRETIZATION AND PLATFORM HETEROGENEITY

Chapter 5 has studied real-time scheduling of a set of implicit and constrained deadline sporadic DAG tasks on an identical multiprocessor platform. Assuming a continuous frequency scheme, we schedule these tasks on the cluster-based multi-core platforms to minimize the CPU power consumption. This chapter modifies our energy-efficient algorithm to adopt the more realistic discrete frequency scheme on a uniform multiprocessor platform. In section 6.1, we discuss the approach to discretize the speed profile. In section 6.2, we discuss the platform heterogeneity, and in Section 6.3 we demonstrate the efficiency of our approach via experimental results. Section 6.4 concludes the chapter.

6.1 Discretization of the Speed-Profile

In Section 5.5.1, we have described two approaches to create the speed-profile for an individual task. While creating the speed-profiles, those approaches assume a continuous frequency scheme. However, the discrete frequency mode is more practical, because a real platform supports only a set of frequencies. Now, we describe the technique to discretize all the speeds available in a speed-profile (assuming that the speed-profile is already created).

Suppose, we execute a task τ_i (and its speed-profile is $\mathcal{S}_i$) in a real-platform, and this platform supports only those speeds available on a speed-set $\mathcal{Z}$. Note that the content of $\mathcal{Z}$ is dependent

This chapter has been previously published at IEEE Transactions on Parallel and Distributed Systems and available at Bhuiyan, A., Liu, D., Khan, A., Saifullah, A., Guan, N., & Guo, Z. (2020). Energy-efficient parallel real-time scheduling on clustered multi-core. IEEE Transactions on Parallel and Distributed Systems, 31(9), 2097-2111.

on the platform. For example, ODROID XU-3 supports a frequency range of 200-1400 MHz (LITTLE cluster) and 200-2000 MHz (big cluster) with scale steps of 100 MHz). Now, for each entry $s_{i,j} \in \mathcal{S}_i$, we find the minimum speed $\mathcal{Z}_k \in \mathcal{Z}$, where $\mathcal{Z}_k \geq s_{i,j}$. Once, we find an appropriate $\mathcal{Z}_k$; we set the value of $s_{i,j}$ as $s_{i,j} = \mathcal{Z}_k$.

Example 14. *Consider a task τ_i with the same speed-profile from Example 11. Let us assume that we will execute τ_i in a platform where $\mathcal{Z} = \{0, 0.2, 0.4, 0.55, 0.75,$ and $1\}$, i.e., this platform supports only six discrete speeds, and all the speeds are normalized w.r.t. the maximum speed supported by this platform. Considering the speed-profile $\mathcal{S}_i$ (from Example 11) and the speed-set $\mathcal{Z}$, we find that:*
(a) $s_{i,1} \leq \{\mathcal{Z}_5$ and $\mathcal{Z}_6\}$
(b) $s_{i,2} \leq \{\mathcal{Z}_4, \mathcal{Z}_5$ and $\mathcal{Z}_6\}$, and
(c) $s_{i,3} \leq \{\mathcal{Z}_1, \mathcal{Z}_2, \mathcal{Z}_3, \mathcal{Z}_4, \mathcal{Z}_5$ and $\mathcal{Z}_6\}$.
Now, we choose the minimum $\mathcal{Z}_k \in \mathcal{Z}$ such that $\mathcal{Z}_k \geq s_{i,j}$. So, we assign $\mathcal{Z}_5$ to $s_{i,1}$, $\mathcal{Z}_4$ to $s_{i,2}$, and $\mathcal{Z}_1$ to $s_{i,3}$. Now, the updated (i.e., discretized) speed-profile becomes

$$\mathcal{S}_i = \begin{pmatrix} 0.75 & 0.55 & 0 \\ 0.33 & 0.47 & 0.2 \end{pmatrix}.$$

Theorem 10. *When a task executes with its discretized speed-profile, it guarantees that the task will not miss the deadline.*

Proof. We have shown in Lemma 8 that a task τ_i will not miss the deadline if executed according to its speed-profile $\mathcal{S}_i$. If we discretize τ_i's speed-profile and execute τ_i according to this speed-profile, then the task still guarantees the real-time correctness. This is because any speed $s_{i,j}$ of the discretized speed-profile is greater than or equal to $s_{i,j}$ when it was continuous. □

6.2 Handling Platform Heterogeneity

In this section, we discuss a specific type of multi-core platform with diverse computing abilities: heterogeneous multi-core platform. We first discuss different types of heterogeneous platforms, and then explain how our proposed techniques can be extended to handle heterogeneity. In a heterogeneous platform, different cores have different computational capabilities. In terms of speed, Funk defined a widely-accepted classification of the heterogeneous platform [57] as follows, where the speed of the processor denotes the work completed (while executing a task) in a single-time unit by this processor.

(i) *Identical multiprocessors*: On Identical multiprocessors, all tasks are executed at the same speed on any processor;

(ii) *Uniform multiprocessors*: On Uniform multiprocessors, all the tasks execute at the same speed if allocated on the same processor, but at a different speed on different processors. So, the execution speed of a task depends on the processor where the task is allocated.

(iii) *Unrelated multiprocessors*: On Unrelated multiprocessors, execution speeds of different tasks may vary on the same processor, i.e., a task's execution speed depends on both the task itself and the processor where it is allocated.

In a heterogeneous platform, each core is designed with a different computational capability, and an efficient task-to-core mapping improves the system resource efficiency. In the context of energy efficiency, two major directions have been mentioned in [6] for any heterogeneous platform:
(i) Find an appropriate core/cluster for task mapping to reduce the overall power consumption of the whole platform.
(ii) Deploy energy-aware scheduling techniques on each core/cluster to reduce power consumption.
Our proposed approach covers both directions. First, we use speed profile to identify efficient

core/cluster to task mapping and then try to reduce the overall cluster speed as much as possible. It works for an identical heterogeneous platform (a.k.a., homogeneous multiprocessor) as task-to-core mapping does not impact energy consumption much.

Table 6.1: Estimated parameters for different cluster of an ODROID XU-3 board.

Cluster Type	$\beta(W)$	$\alpha(W/MHz^{\gamma})$	γ
big	0.155	3.03×10^{-9}	2.621
LITTLE	0.028	2.62×10^{-9}	2.12

Now, we extend our approach to apply to the uniform heterogeneous platform by modifying the parameters in the power model in Equation (2.1), i.e., setting different $\alpha, \beta,$ and γ values for the 'big' and 'LITTLE' cluster. Under such consideration, different clusters no longer share the same power model, and the same task may have different execution requirements on different clusters. We report the estimated values of $\alpha, \beta,$ and γ in Table 6.1. These parameters in Table 6.1 are adopted from [86]. The work in [86] estimated these parameters for the ODROID XU-3 board using the real power measurements along with a curve fitting method. They have also assumed that there is another contributor to the total power consumption of a cluster, i.e., the ”uncore” power consumption (reported in Table 6.2 which is also adopted from [86]). The ”uncore” power consumption introduced in the system from some components other than a processor, e.g., a shared cache. Similar to the dynamic power consumption, the ”uncore” power consumption also depends on the processor frequency. However, unlike the dynamic power consumption, there is always some ”uncore” power consumption as long as the cluster remains on (even if there is no workload on a processor).

Considering all the parameters from Table 6.1 and Table 6.2, we bring the following modification

Table 6.2: The "uncore" power consumption for different cluster of an ODROID XU-3 board.

Freq(GHz)	2	1.8	1.6	1.4	1.2	1.0
big cluster(W)	0.8	0.528	0.39	0.309	0.244	0.182
Freq(GHz)	1.4	1.2	1.0	0.8	0.6	0.4
LITTLE cluster(W)	0.04	0.04	0.04	0.04	0.04	0.04

in Equation (2.1):

$$P(s) = N_p\beta + \alpha s^\gamma + P_s(f), \tag{6.1}$$

N_p denotes the number of cores per cluster, and $P_s(f)$ denotes the "uncore" power consumption. We have a different power model for the "big" and the "LITTLE" cluster, but we still don't know what the basis of assigning a task to a cluster is. Recall that, while creating the speed-profile, some vital information (e.g., the speed of a core at a specific time) were known to us (Subsection 5.5.1). If the execution speed of a task is greater than a certain threshold at any point from its release to its deadline, then we assign this task to the big cluster. Else, we assign this task to the LITTLE cluster. For the platform we consider (ODROID XU-3), we set the threshold to 0.7. It is the ratio of the maximum speed supported by the big cluster and the LITTLE cluster (see Table 6.2).

6.3 Simulation Study

In this section, we perform simulations and compare the results with existing baselines. We follow the same DAG task generation method that uses the Erdos-Renyi method (Section 5.6). Likewise the previous chapters, we compare power consumption by varying the following two parameters for each task: *task periods (utilization)* and *the number of nodes*. Recall that, for the task partitioning step, we have two options. We randomly choose any two tasks and allocate them to the same cluster, or greedily choose the ones with the lowest speed as proposed. Besides, there are two options for speed-profile calculation (Approaches A and B in Section 5.5.1). We combine these options in two steps lead to four baselines: *MaxSpeed_Greedy*, *SingleSpeed_Greedy*, *MaxSpeed_Random*, *SingleSpeed_Random*. In addition, we consider the following three baselines for comparison:

- Federated scheduling with intra-task processor merging [67], denoted by *Fed_Guo*;
- GSS algorithm [130], denoted by *GSS_Zhu*.
- DVFS and DPM combination [37], denoted by *com_Chen*.

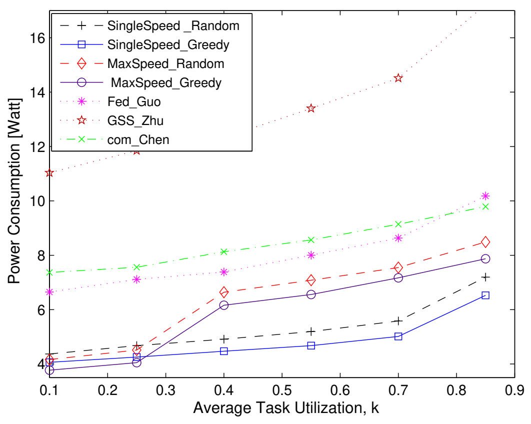
6.3.1 Uniform Heterogeneous Platform with a Continuous Frequency Scheme

In this section, considering the uniform heterogeneous platform and a continuous frequency scheme, we report the power consumption comparison for different approaches mentioned earlier. Under such a platform, different clusters no longer share the same power model and we use the power model described in Equation (6.1).

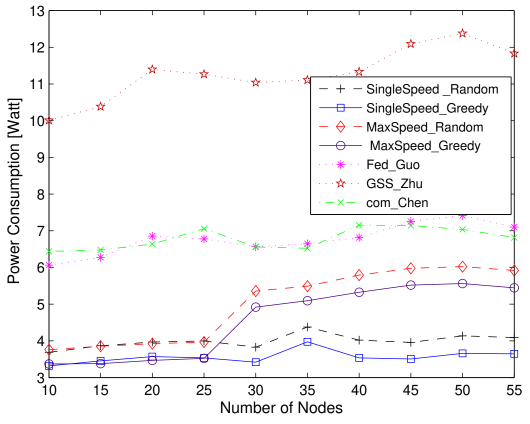
6.3.1.1 Constrained Deadline Task

Here, we report the power consumption under the scheme for constrained deadline tasks. We evaluate the efficiency of our proposed method by changing two parameters; task period (utilization) and the number of nodes in the task.

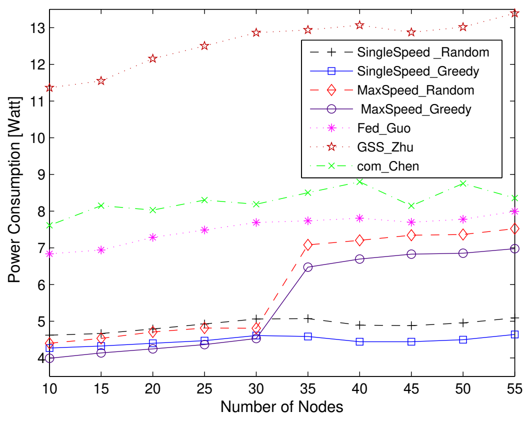
Effect of Varying Task Periods (Utilization). Here we control the average task utilization through varying the task period. In order to make the task schedulable, the critical path length L_i of task τ_i should not exceed its deadline D_i. We calculate the task deadline D_i using Equation (5.5). The results are presented in Figure 6.1(a). The results indicate a proportional relationship between the average power consumption and average task utilization. It happens because a higher



(a) Comparison of power consumption with different approaches for DAGs with a fixed number of nodes as 30.



(b) Average power consumption comparison with different approaches for tasks with harmonic periods.



(c) Average power consumption comparison with different approaches for tasks with arbitrary periods.

Figure 6.1: Power consumption comparison between different approaches for the **constrained** deadline tasks considering a **continuous** frequency scheme on the **uniform** heterogeneous platform

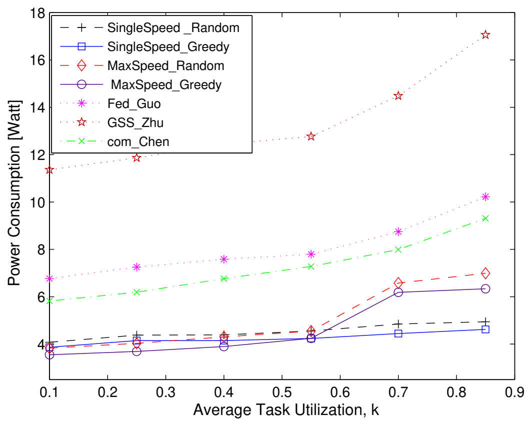
task utilization imposes tighter real-time restrictions. It restricts (refer to Figure 5.1(b)) the space for the segment length optimization. Figure 6.1(a) shows that *SingleSpeed_Greedy* approach performs better for a higher utilization value. On average, the *SingleSpeed_Greedy* approach leads to a power saving of at least 30.23% and 60.2% compared to *Fed_Guo* and *GSS_Zhu* approaches, respectively. In *SingleSpeed_Greedy* approach, a task executes with a single speed throughout the deadline. During the task partitioning step, a suitable partner (with similar speed-profile) leads to energy efficiency. However, for the other approaches task speed may vary throughout the deadline. In that case, evil alignment and a significant variation in the speed may reduce energy efficiency (see Figure 5.2 and Example 8).

Effect of Varying the Numbers of Nodes. Now we vary the number of nodes (10 to 55) (T_i is fixed) and report the average power consumption. We report the average power consumption for harmonic deadline tasks in Figure 6.1(b) and arbitrary deadline tasks in Figure 6.1(c). We observe that the power consumption pattern does not change that much, i.e., SingleSpeed_Greedy approach outperforms other approaches especially when the number of nodes (in each DAG) are high, 35 or higher. Specifically, under harmonic task periods, the SingleSpeed_Greedy incurs 40.19% and 65.9% less power on average compared to Fed_Guo and GSS_Zhu; under arbitrary task periods, the savings potential are 33.43% and 61.96%, respectively.

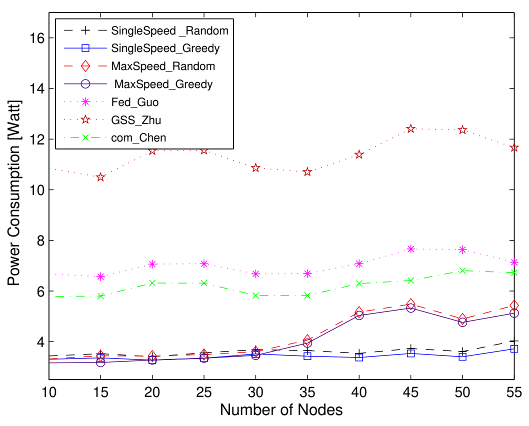
6.3.1.2 Implicit Deadline Task

Effect of Varying Task Periods (Utilization). Using previous setup (Section 6.3.1.1), We observe that the average energy consumption is directly proportional to the average task utilization. Figure 6.2(a) shows that *SingleSpeed_Greedy* approach performs better for a higher utilization value and on average, saves at least 35.21% and 62.52% compared to *Fed_Guo* and *GSS_Zhu* approaches, respectively.

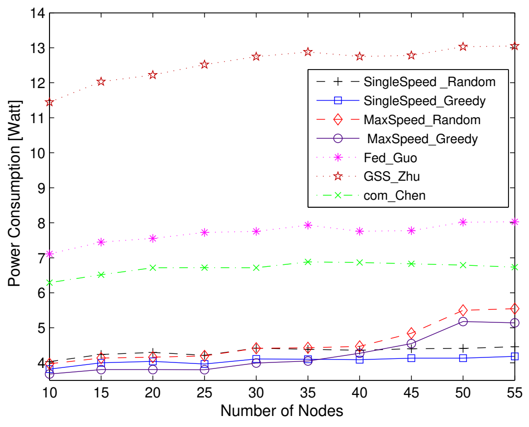
Effect of Varying the Numbers of Nodes. Figure 6.2(b) and 6.2(c) report the average power consumption for the harmonic and arbitrary deadline tasks, respectively. We observe that the SingleSpeed_Greedy approach outperforms other approaches when the number of nodes (in each DAG) are high. Under harmonic task periods, the SingleSpeed_Greedy incurs 44.84% and 67.55% less power on average compared to Fed_Guo and GSS_Zhu; under arbitrary task periods, the savings potential are 42.33% and 67.19%, respectively.



(a) Comparison of power consumption with different approaches for DAGs with a fixed number of nodes as 30.



(b) Average power consumption comparison with different approaches for tasks with harmonic periods.

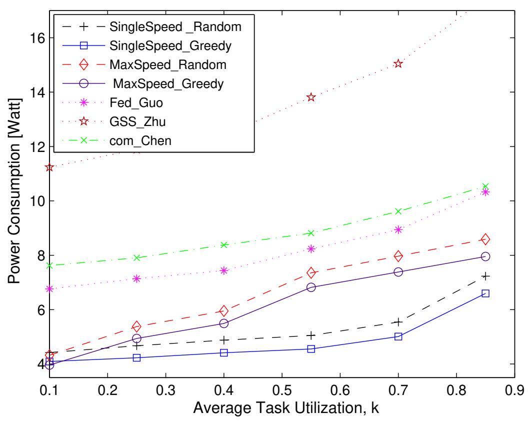


(c) Average power consumption comparison with different approaches for tasks with arbitrary periods.

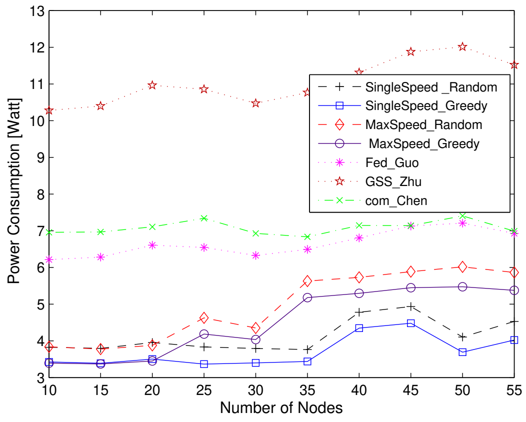
Figure 6.2: Power consumption comparison between different approaches for the **implicit** deadline tasks considering a **continuous** frequency scheme on the **uniform** heterogeneous platform

6.3.2 Uniform Heterogeneous Platform With a Discrete Frequency Scheme

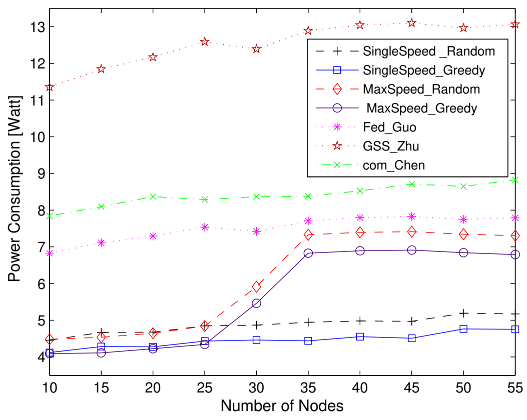
In this section, we report the power consumption comparison for the (previously mentioned) approaches considering the uniform heterogeneous platform and a discrete frequency scheme. Under such a platform, we discretize the frequency using the technique described in Section 6.1.



(a) Comparison of power consumption with different approaches for DAGs with a fixed number of nodes as 30.



(b) Average power consumption comparison with different approaches for tasks with harmonic periods.



(c) Average power consumption comparison with different approaches for tasks with arbitrary periods.

Figure 6.3: Power consumption comparison between different approaches for the **constrained** deadline tasks considering a **discrete** frequency scheme on the **uniform** heterogeneous platform

6.3.2.1 Constrained Deadline Task

Here, we consider the constrained deadline tasks and report their average power consumption by changing two parameters: task period (or utilization) and the number of nodes.

Effect of Varying Task Periods (Utilization). Similar to the Figure 6.1(a), and 6.2(a), we observe that the (i) average energy consumption is directly proportional to the average task utilization. (ii)

SingleSpeed_Greedy approach consumes less power than other approaches (see Figure 6.3(a)).

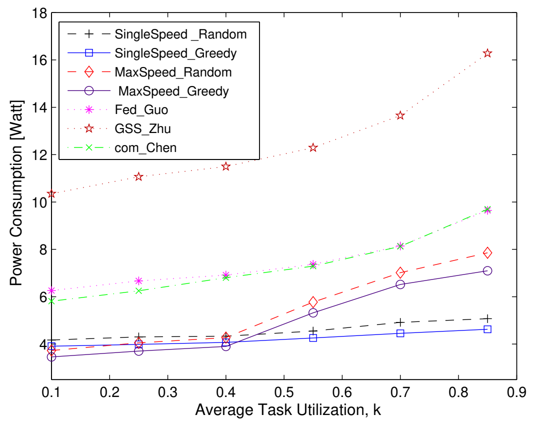
Effect of Varying the Numbers of Nodes. We vary the number of nodes (10 to 55) and report the average power consumption for harmonic (arbitrary) deadline tasks in Figure 6.3(b) (Figure 6.3(c)). Similar to the Figure 6.1(b), 6.1(c), 6.2(b) and 6.2(c), we observe that the (i) Performance of *SingleSpeed_Random, SingleSpeed_Greedy, MaxSpeed_Greedy,* and *MaxSpeed_Random* does not vary that much for a small number of nodes (typically 10 to 25) per DAG. (ii) *SingleSpeed_Greedy* approach performs better (i.e., consume less power) than other approaches when the number of nodes per DAG is high.

6.3.2.2 Implicit Deadline Task

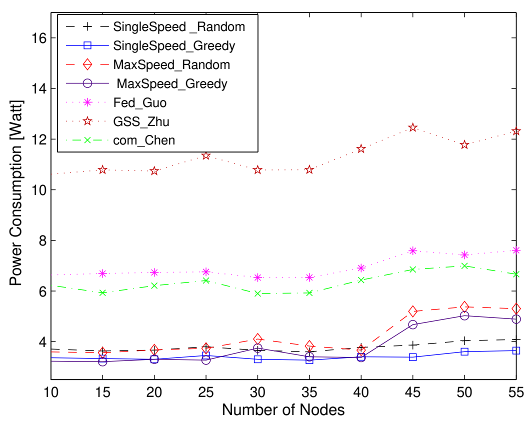
Now, we report the average power consumption using the same setup as described in Section 6.3.2.1, i.e., (a) for a fixed number of nodes (30) per task, change their utilization value, and (b) vary the number of nodes (10 to 55) per task, while keeping T_i fixed. We report the average power consumption in Figure 6.4(a), 6.4(b), and 6.4(c). From this figure, we observe that the (i) Performance of *SingleSpeed_Random, SingleSpeed_Greedy, MaxSpeed_Greedy,* and *MaxSpeed_Random* does not vary that much for a smaller task utilization or when the number of nodes per DAG is small (typically 10 to 35). (ii) *SingleSpeed_Greedy* approach performs better (i.e., consume less power) compared to the other approaches when the number of nodes per DAG is high.

6.4 Conclusion

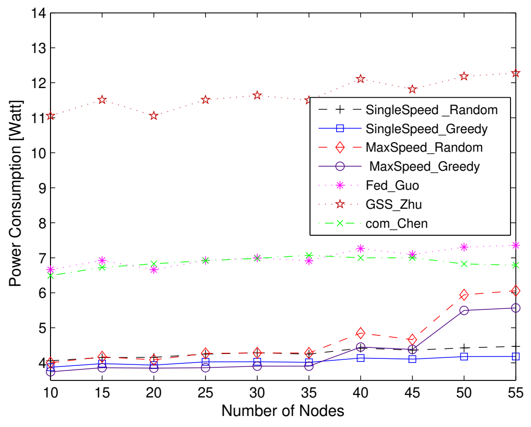
In this chapter, we extend our approach (proposed in Chapter 5) to adopt the more realistic discrete frequency scheme on a uniform multiprocessor platform. We present the steps to discretize the speed profile. Besides, we present a detailed discussion on how we can adapt to the platform



(a) Comparison of power consumption with different approaches for DAGs with a fixed number of nodes as 30.



(b) Average power consumption comparison with different approaches for tasks with harmonic periods.



(c) Average power consumption comparison with different approaches for tasks with arbitrary periods.

Figure 6.4: Power consumption comparison between different approaches for the **implicit** deadline tasks considering a **discrete** frequency scheme on the **uniform** heterogeneous platform.

heterogeneity. Finally, we report the efficiency of our approach via extensive experimental results.

CHAPTER 7: MIXED-CRITICALITY REAL-TIME SCHEDULING OF GANG TASK SYSTEMS

In the previous chapters, we have proposed energy-efficient scheduling approaches that assume each task will execute up to their WCET. This assumption is pessimistic as the task execution pattern may show significant variability (w.r.t. time). Designing a system with such an assumption may lead to system over-provisioning and excessive power/energy consumption. Mixed-Criticality (MC) framework was proposed to efficiently utilize the non-negligible gap between the WCET and the actual execution time, leading to a minimized resource over-provisioning. Different software components with varying levels of criticality are integrated into a common platform in an MC setup.

Mixed-criticality (MC) scheduling of sequential tasks (with no intra-task parallelism) has been well-explored by the real-time systems community. MC scheduling of parallel tasks is highly challenging due to the requirement of various assurances under different criticality levels. However, till date, there has been little progress on MC scheduling of parallel tasks, and these works have restricted their attention mostly on the DAG task model. However, other well-known representatives of the parallel task model, i.e., the gang task model, represent an efficient mode-based parallel processing scheme with many potential applications. This model suits various applications that use parallelism, some of which are implemented using the message-passing approach and tools like MPI. This chapter addresses the MC scheduling of parallel tasks of gang model that allows workloads to execute on multiple cores simultaneously and the change to the degree of parallelism of a task upon a mode switch. To schedule such task sets, we propose a new technique

This chapter has been previously published at Springer Real-Time Systems and available at Bhuiyan, A., Yang, K., Arefin, S., Saifullah, A., Guan, N., & Guo, Z. (2021). Mixed-criticality real-time scheduling of gang task systems. Real-Time Systems, 1-34.

GEDF-VD, which integrates Global Earliest Deadline First (GEDF) and Earliest Deadline First with Virtual Deadline (EDF-VD). We prove the correctness of GEDF-VD and provide a detailed quantitative evaluation in terms of speedup bound in both the MC and the non-MC cases. Experiments on randomly generated gang task sets are conducted to validate our theoretical findings and to demonstrate the effectiveness of the proposed approach.

7.1 Introduction

Due to size, weight, and power considerations, there is a trend that multiple tasks with different criticality levels (that are subject to varying degrees of assurance/verification) share a computing platform [122]. This type of system is commonly known as a *mixed-criticality (MC) system*, where each task can be associated with various execution budgets. During normal operation, all tasks are scheduled according to their typical execution budget. However, some critical tasks may exceed their typical budget and need more resources to finish their execution. Suppose the available resources are not sufficient in these scenarios. In that case, the less critical task will be sacrificed to free up the resources for accommodating the additional computational requirements requested by the more critical ones.

Take an avionics software standard as an example, where the ground control subsystems are more safety-critical than ground communication and light controls. During the incident of emergency (e.g., an accident), it is more important to execute the safety-critical components rather than the other components. On the other hand, in normal condition, all these components are required to perform smoothly (for more details, refer to the Table 1.1 of [79], which demonstrates the RTCA DO-178B avionics software standard). MC scheduling has received considerable attention (refer to [32] for a thorough and updated survey) as it brings significant improvements in resource efficiency.

Note that safety-critical MC systems have tight correctness requirements. These requirements can be verified by two related but orthogonal perspectives: *a priori verification* and *run-time robustness* [12]. Before run-time, a priori verification determines whether a system will behave correctly (or not) during execution, while run-time robustness deals with unexpected system behavior at run-time. There are some debates regarding MC's applicability to run-time robustness [52, 53]. Although an MC system has imitation from the perspective of a priori verification (which is our work scope), these criticisms usually do not hinder the applicability of an MC system for its designed scope of a priori verification [12].

Parallel Computing Workloads. Recent advances in parallel computing allow executing a single piece of code simultaneously on multiple computing units or a set of threads to execute on multiple processors concurrently. Such design provides a much better capability of exploiting the benefits provided by modern platforms. As a result, there is an urgent need in handling workload models that allow intra-task parallelism (i.e., parallel tasks). Parallel computing systems perform a large number of computations and often need to interact with their surroundings under real-time constraints, e.g., arms system (RADAR). In these types of applications, a lot of processors co-operate with each other, and these communications are timing critical. It is necessary for a system to have both high performance and predictability; i.e., efficient control that minimizes the introduced overhead, while responding to external events (coming through sensors) in real-time. In this work, we consider the gang scheduling, where all threads of a task are grouped into a gang, and during execution, the whole group is concurrently scheduled on distinct cores. The gang task model is a practical, widely used, and representative workload model for intra-task parallelism [3, 46, 61, 76]. Also, the gang task model is supported by some widely used parallel computing programming standards (e.g., OpenACC [95]), which is commonly used in graphics processing unit (GPU).

Existing Work. The real-time systems and parallel computing communities have given considerable attention towards these two directions: *MC scheduling* and *scheduling of parallel tasks*.

These two emerging trends bring in some critical and exciting problems, and there is an emerging need in integrating those two trends. There has been extensive research on the (a) MC scheduling of sequential (i.e., non-parallel) tasks (refer to the recent survey in [33, 32]) and (b) scheduling of parallel tasks with a single-criticality level [76, 61, 46, 4, 30, 82]. Till date, very few efforts [87, 11, 83] have been made towards the combined problem of MC scheduling of parallel tasks. To our knowledge, none of these efforts has considered mixed-criticality gang task scheduling on multi-core platforms.

Motivation Behind This Work. Multi-core platform enables applications that require better energy efficiency, higher performance, and real-time guarantees. The notion of MC systems with the intra-task parallelism stems from many current trends. For example, the number of cores fabricated on a chip is increasing rapidly. Besides, the computational demand for an individual task (with stringent timing requirements) is rising, which makes it essential to consider the intra-task parallelism. Furthermore, when safety-critical and non-safety-critical tasks share a common computational platform, there is an increasing demand to integrate functionality with different criticality levels. Such demand promotes the idea of MC scheduling, i.e., combining various functionalities of varying criticality levels onto the same computing platform.

In this work, we study the mixed-criticality gang task scheduling. The gang task model has many promising applications, e.g., fault-tolerant systems. A fault-tolerant system often follows the mixed-criticality model [32]. If a fault is identified in such a system, it is recovered via various recovery techniques, e.g., exception handling, recovery blocks, and task replication. Some extra work has to be undertaken upon identifying a fault, which leads to the abandonment or delay of some less critical works. The impact of a fault in such a system ranges from no visible effect to an entire system crash. To overcome these faults, the *ASTEROID* project is proposed (a cross-layer fault-tolerance solution for the mixed-criticality platform) that detects errors and recovers the system in different software and hardware layers [45, 51]. Considering such a cross-layer platform, a recent work in [104] proposed a replica-aware co-scheduling (with strict priority preemptive protocols) for a mixed-criticality system that improves the system performance. They have considered the replicas as a gang that is activated concurrently on multiple cores.

Challenges. In gang task model, a task cannot start execution until the number of available cores is no less than what is required by it (i.e., a task's *degree of parallelism*). This simple constraint adds a huge restriction on real-time schedulability and makes the problem highly challenging. We are aware of only one known correct schedulability analysis [46] under Global Earliest Deadline First (GEDF) for gang tasks. Besides, integrating MC in gang scheduling scheme adds additional challenges due to the dual notion of correctness. In the normal mode, a task may have a utilization less than 1, while in the critical mode, the utilization could be much higher than one [46]. Such a change in the utilization adds significant complexity in speedup bound analysis [13] . For example, in the speedup bound analysis for MC scheduling of ordinary sporadic tasks, an individual task's utilization is at most the processor's speed is a straight forward and necessary feasibility condition. At the same time, it no longer holds for the gang tasks. Besides, the scheduler does not know the exact behavior of each task before run-time (non-clairvoyant). Hence, the scheduler must be able to detect the critical condition early enough to allocate more resources to the more critical tasks to handle this drastic change and still be able to meet the deadlines.

This research. In this chapter, we study the real-time scheduling of MC gang tasks on identical multi-core platforms [1]. We propose the first scheduling algorithm GEDF-VD (GEDF with Virtual Deadline) for MC gang tasks. Our approach leverages the synthesis of uniprocessor scheduling techniques such as EDF-VD [14] as well as *GEDF* [46] that was designed for non-MC gang tasks. Specifically, we make the following contributions:

[1]On an identical multi-core platform, all tasks are executed at the same speed on any processor. Refer to Section 6.2 for details.

- We generalize the gang task model to the MC context by incorporating extensions on both the execution time and the degree of parallelism dimensions.
- We propose a scheduling algorithm called GEDF-VD for the generalized model.
- We formally prove its correctness of GEDF-VD (in LO and HI-criticality mode) through a utilization based schedulability test.
- Extensive simulations under randomly generated task sets are conducted to demonstrate the real-time performance and effectiveness of the proposed algorithm in terms of acceptance ratio, which is defined as the ratio of the number of schedulable task sets over the total number of task sets.

Organization. The remainder of this paper is organized as follows. Section 7.2 discusses related prior work. Section 7.3 describes the task model, notations, and preliminaries. Section 7.4 provides a detailed description of our scheduling algorithm and prove its correctness. Section 7.5 derives the speedup bounds for the non-MC and MC platform, under GEDF and GEDF-VD scheduling algorithms, respectively. Simulation results are presented in Section 7.6. Section 7.7 concludes this paper and points out future research directions.

7.2 Related Work

Since Vestal's proposal [122] of MC workload model, much work has focused on scheduling MC tasks (refer to Burns et al. [32] for a survey). For uniprocessor platforms, many algorithms were proposed based on both fixed priority (e.g., Li et al. [80], Baruah et al. [15]) and dynamic priority scheduling(e.g., Easwaran et al. [48]). The work in [26, 25] proposed the precise scheduling policy, where all LO-criticality tasks receive a full-service guarantee even after a mode switch. Numerous MC scheduling algorithms were proposed for multiprocessor platforms [78, 19, 120, 5, 121]. Considering the multiprocessor platforms, Lee et al. [78] and Baruah et al. [19] proposed fluid-based MC models, and a semi-partitioned based scheme is proposed by Awan et al. [5].

Considering different parallel tasks models (e.g., synchronous task model [4], DAG model [30, 82, 67, 24, 23, 66] and gang models [76, 61, 46]) there have been a number of works that have provided the energy efficiency technique, schedulability analysis, and the speedup bound (i.e., resource augmentation bound) for various scheduling strategies. For synchronous tasks under GEDF scheduling, Andersson et al. [4] proved a resource augmentation bound of 2 with constrained deadlines tasks. Considering DAG tasks (with arbitrary deadlines) under GEDF, Li et al. [81] and Bonifaci et al. [30] simultaneously proved a resource augmentation bound of 2. Bonifaci et al. [30] also showed the bound to be 3 under global rate-monotonic scheduling. For implicit deadline DAG tasks under federated scheduling, a resource augmentation bound of 2 is showed by Li et al. [82].

Gang scheduling and Coscheduling was initially introduced to perform parallel processing with fine-grained interactions efficiently [54, 97, 59]. Both of these approaches allocate resources to the threads of the same task concurrently. However, gang scheduling imposes a strict requirement of executing all threads of the same task simultaneously. In contrast, in coscheduling, some threads may not execute concurrently with the rest of the threads in the same task. Some recent work used this concept to execute the parallel workload in cloud computing [116] and extended to incorporate hard real-time tasks [61]. The work in [61] also has proposed a DP-Fair based scheduling of periodic gang tasks and proved a speedup bound which is no larger than $(2 - 1/m)$.

A recent work by Alahmad et al. [3] proposed the isochronous scheduling, which has some similarity to the traditional gang scheduling. Unlike the gang scheduling, the isochronous model assumed that the job versions might not be compatible with all the processors available. Although the work in [3] has some connections to the MC task model, they did not explicitly concern different criticality levels of a task. It aims to achieve higher design assurance levels by using adequate monitoring and improving mechanisms. In contrast, the MC scheduling that we propose focuses on providing service guarantee only to the high criticality jobs where computational resources are not adequate. Kato et al. [76] introduced gang task scheduling based on global EDF. Dong et al. [46] proposed a schedulability analysis based on lag-based reasoning. Few other works, e.g., Goossens et al. [60], provided schedulability tests for fixed task-priority scheduling of real-time periodic gang tasks.

Although a good number of works studied MC scheduling and parallel tasks scheduling individually, very few works studied the scheduling of MC parallel tasks [87, 11, 83, 104]. Rambo et al. [104] proposed a replica-aware co-scheduling approach (that is a combination of strict priority preemptive (SPP) policy and gang scheduling policy) for mixed-critical systems. Baruah et al. [11] and Li et al. [83] proposed the MC scheduling of DAG models, while Liu et al. [87] proposed the MC scheduling of the synchronous task model. Unlike these works, we consider the gang task model, where a task cannot execute if the number of available cores is less than its degree of parallelism. This constraint makes the scheduling problem highly challenging.

7.3 Dual-Criticality Gang Task Model

In this chapter, we discuss the scheduling of a sporadic MC *implicit deadline* (i.e., the period of a task is equal to its deadline) gang task set $\tau = \{\tau_1, \ldots, \tau_n\}$ on M identical cores. In this model, each task generates an infinite number of MC gang jobs (the j^{th} job of task τ_i is denoted as $\tau_{i,j}$). For details regarding the traditional gang task model, refer to Section 2.1.2. To describe the dual-criticality gang task model, first, we provide details on *MC sporadic sequential task model.* Then, by leveraging these two models,i.e., traditional non-MC gang task model and the MC sporadic sequential task model, we generalize the gang task model to the MC context. In this work, we restrict our attention to dual-criticality model.

MC sporadic task model. In a dual-criticality systems, the criticality level of τ_i is represented by

$\chi_i = \{\text{LO}, \text{HI}\}$. The worst case execution time (WCET) estimations of each task is also represented by a tuple $(c_i^{\text{LO}}, c_i^{\text{HI}})$ where c_i^{LO} and c_i^{HI} represent the LO and HI-criticality WCETs respectively. c_i^{HI} is measured by a more pessimistic tool by considering all possible scenarios, while c_i^{LO} is calculated using a less pessimistic yet realistic tool. Collection of all LO- and HI-criticality tasks in τ are denoted by τ_{LO} and τ_{HI} respectively. u_i^{LO} and u_i^{HI} denotes the utilization of τ_i in LO- and HI-criticality mode respectively, where $u_i^{\text{LO}} = c_i^{\text{LO}}/T_i$ and $u_i^{\text{HI}} = c_i^{\text{HI}}/T_i$.

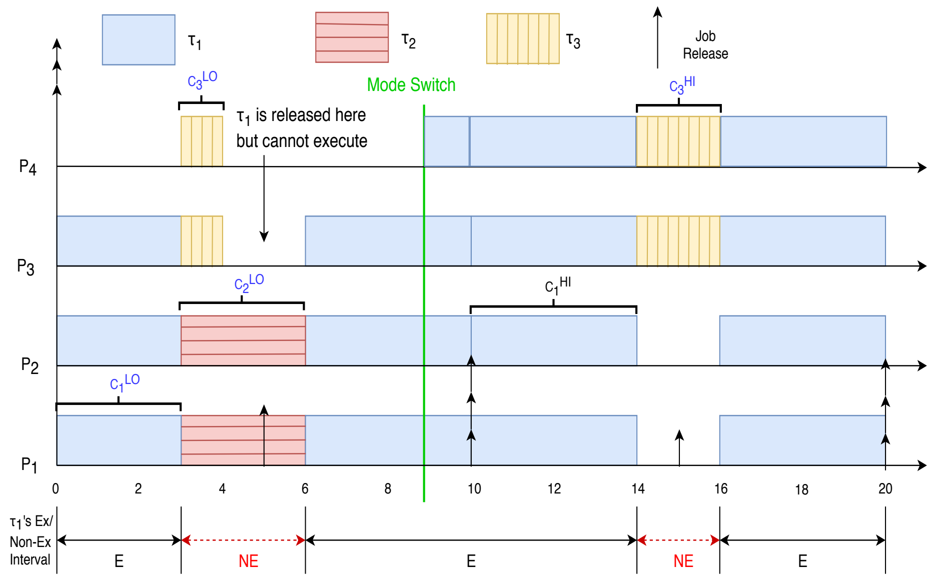


Figure 7.1: A GEDF scheduling of the MC gang task-set from Table 7.1 and the executing (E)/non-executing (NE) intervals of τ_1.

MC gang task model. By leveraging the above two models, we consider a workload model of MC gang tasks, where each task τ_i is represented by a 7-tuple $(m_i^{\text{LO}}, m_i^{\text{HI}}, \chi_i, c_i^{\text{LO}}, c_i^{\text{HI}}, T_i, D_i)$, where

m_i^{LO} = degree of parallelism of task τ_i in LO-criticality mode.

m_i^{HI} = degree of parallelism of task τ_i in HI-criticality mode, and $m_i^{\text{LO}} \leq m_i^{\text{HI}}$.

χ_i = criticality level of each task τ_i and $\chi_i \in \{\text{LO}, \text{HI}\}$.

c_i^{LO} = WCET of τ_i in LO-criticality mode.

c_i^{HI} = WCET of τ_i in HI-criticality mode, and $c_i^{\text{LO}} \leq c_i^{\text{HI}}$.

T_i = minimum inter-arrival time between jobs.

D_i = relative deadline.

In this model, a task τ_i occupies $m_i^{\text{LO}}(m_i^{\text{HI}})$ processors for $c_i^{\text{LO}}(c_i^{\text{HI}})$ time quantum at LO(HI)-criticality mode. Note that, if $\forall \tau_i, m_i^{\text{LO}} = m_i^{\text{HI}} = 1$, i.e., degree of parallelism for each gang task is 1, our analysis will reduce to the existing MC scheduling method designed for the sporadic task model. We believe this is common for a restricted special case of a more complex and expressive model. For example, the directed acyclic graph (DAG) task model [30, 82] is popular to represent intra-task parallelism. Many of the existing schedulability analysis considering the DAG model would also reduce to prior study for ordinary sporadic tasks if the number of nodes of each DAG task is equal to 1.

Now, we generalize the utilization concepts to suit the MC gang task model, which are analogous to the above-mentioned concepts. Refer to the Example 15 for details.

$$U_{\text{LO}}^{\text{LO}} \overset{\text{def}}{=} \sum_{\tau_i \in \tau_{\text{LO}}} m_i^{\text{LO}} \times c_i^{\text{LO}} / T_i,$$

$$U_{\text{HI}}^{\text{LO}} \overset{\text{def}}{=} \sum_{\tau_i \in \tau_{\text{HI}}} m_i^{\text{LO}} \times c_i^{\text{LO}} / T_i,$$

$$U_{\text{HI}}^{\text{HI}} \overset{\text{def}}{=} \sum_{\tau_i \in \tau_{\text{HI}}} m_i^{\text{HI}} \times c_i^{\text{HI}} / T_i$$

Example 15. *Consider the task-set $\tau = (\tau_1, \tau_2, \tau_3)$ in Table 7.1. For this task-set we derive the*

Table 7.1: An MC gang task set with GEDF schedule shown in Figure 7.1.

Task ID	c_i^{LO}	c_i^{HI}	T_i	χ_i	m_i^{LO}	m_i^{HI}
τ_1	3	4	5	HI	3	4
τ_2	3	3	10	LO	2	2
τ_3	1	2	10	HI	2	2

utilization as follows:

$U_{\text{LO}}^{\text{LO}} = c_2^{\text{LO}} \times m_2^{\text{LO}}/T_2 = 0.6, U_{\text{HI}}^{\text{LO}} = c_1^{\text{LO}} \times m_1^{\text{LO}}/T_1 + c_3^{\text{LO}} \times m_3^{\text{LO}}/T_3 = 2,$ *and* $U_{\text{HI}}^{\text{HI}} = c_1^{\text{HI}} \times m_1^{\text{HI}}/T_1 + c_3^{\text{HI}} \times m_3^{\text{HI}}/T_3 = 3.6.$

Example 16. *Consider the MC gang task set in Table 7.1 to be scheduled in four cores. A GEDF schedule for this task set is shown in Figure 7.1. The system starts at* LO*-criticality mode, and all the tasks (τ_1, τ_2, τ_3) will execute up-to C_i^{LO}. Recall that, m_i^{LO} is the degree of parallelism of τ_i in* LO*-criticality mode. Hence, τ_1 cannot execute at $t = 5$ as it needs three cores to execute while only two cores (P_3 and P_4) are idle. At a mode switch ($t = 9$), all* LO*-criticality tasks (τ_2) are dropped, and all* HI*-criticality tasks (τ_1, τ_3) will execute up-to C_i^{HI} (at m_i^{HI} cores). After a mode switch, all* HI*-criticality jobs (including the ones which are currently executing) will execute up-to their* HI*-criticality WCET.*

Motivations behind this model. Some commonly used parallel computing programming standards (e.g., OpenACC [95]) support the gang task model. OpenACC is one of the parallel computing programming standards used for GPU architecture. Recently, there has been extensive research on GPU architecture (few to mention [35, 49, 123, 127]). GPU architecture is popular because of the features like (1) highly threaded but low context switch latency architecture, (2) high parallelism and (3) minimal dependency between data elements, etc. Previous work on GPU scheduling considered limited or no preemption policy [49, 123]. However, this work is motivated by some recent attempts to incorporate the *preemptive* support in GPUs. For example, a prototype has been implemented and tested with preemptive support (at the pixel level and the thread level) in a virtualized environment in a recent work [35]. Its prototype is EDF based, and enhanced with a bandwidth isolation mechanism (e.g., constant/total bandwidth servers [114]) for the graphics and computing workloads. Also, the prototype is tested on a recent NVIDIA Tegra-based system on a chip (SoCs) [93]. Since some recent work study the preemptive support in the GPU architecture, there is a need for a comprehensive study of gang task scheduling using GEDF.

Now, we introduce some definitions and preliminaries which will be frequently used in later sections of this paper.

Definition* 1. *(MC-correct schedule): *Scheduling strategy must ensure an* MC-correct *schedule, as defined below [83].*

• *If the system stays in normal condition (i.e., each task in the system finishes execution within its* LO*-criticality WCET), all tasks must meet their deadlines.*

• *If the system transits into a critical condition (i.e., there exists a* HI*-criticality task executing beyond its* LO*-criticality WCET), all* HI*-criticality tasks must meet their deadlines, while* LO*-criticality tasks need not so.*

***Definition* 2. (Executing/Non-Executing interval)** *An interval $[t_1, t_2)$ (where $t_1 < t_2$) is an **executing** interval for a task τ_i if m_i out of M cores are executing the current active job released by τ_i throughout this interval. Otherwise, $[t_1, t_2)$ is a **non-executing** interval for τ_i. An illustrative example is shown in Figure 7.1 by pointing the executing and non-executing intervals for task τ_1.*

***Definition* 3. (Active/pending task)** *If there exists a task $\tau_i \in \tau$, such that it has a job $\tau_{i,j}$ where $r_{i,j} \leq t < d_{i,j}$. Here, $r_{i,j}$ and $d_{i,j}$ respectively denotes the release time and deadline of $\tau_{i,j}$, then τ_i is considered as an active task at time t. A job is* pending *if it is released but not finished [46].*

***Definition* 4.** *Maximum possible number of idle cores Δ_i^{LO} (Δ_i^{HI}) for a task τ_i refers to the maximum* *number of available cores (that are not executing any job) at any time in* LO *(*HI*-criticality mode) during τ_i's non-executing intervals in which it has a pending job [46].*

Example 17. *Let us consider a 4-core platform and task set $\tau = \{\tau_1, \tau_2, \tau_3\}$ from Table 7.1. At* LO*-criticality mode, the degree of parallelism for these tasks are given as: $m_1^{\text{LO}} = 3$, $m_2^{\text{LO}} = m_3^{\text{LO}} = 2$. For this task set, the maximum possible number of idle cores at* LO*-criticality mode is: $\Delta_1^{\text{LO}} = 2,\ \Delta_2^{\text{LO}} = \Delta_3^{\text{LO}} = 1.$ This is because τ_1 cannot execute at time t (even when it has a pending job) if τ_2 or τ_3 is executing at t. The degree of parallelism for τ_2 (or τ_3) is 2. So, the maximum number of idle cores for τ_1 is Δ_1, where $\Delta_1 = 4 - 2 = 2.$ We can calculate the value of Δ_2 and Δ_3 in the same approach (refer to Algorithm 1 in [46]). At* HI*-criticality mode, degree of parallelism for these tasks becomes: $m_1^{\text{HI}} = 4$, $m_2^{\text{HI}} = m_3^{\text{HI}} = 2$, and the maximum possible number of idle cores will be: $\Delta_1^{\text{HI}} = 2,\ \Delta_2^{\text{HI}} = \Delta_3^{\text{HI}} = 0.$*

System behavior and scope of this work. It is expected that an MC system starts execution in normal mode. The system-wide mode transition is triggered if a HI-criticality task τ_i has received cumulative execution length beyond its LO-criticality WCET and did not signal its finishing. Like the Vestal model [122], after a mode switch, no LO-criticality tasks get any service guarantee. After mode transition (from LO-criticality to HI-criticality), at the first idle instant, the system switches back to the LO-criticality mode again. All other scenarios (e.g., a HI-criticality task runs for more than its HI-criticality WCET) are considered as *erroneous*, where no guarantees will be made and hence is not considered.

7.4 GEDF-VD for Dual-Criticality System

Now we describe our algorithm for the MC task systems considering the GEDF-VD algorithm. In this work, we consider implicit deadline (So, we use the terms *deadline* and *period* interchangeably) sporadic task systems on identical multi-core platforms. We integrate an uniprocessor

MC scheduling technique (*EDF-VD* [14]) with a multiprocessor gang task scheduling technique (*GEDF* [46]) and derive a new algorithm named GEDF-VD (Subsections 7.4.1 and 7.4.2). In our approach, we determine a *scaling factor*, which scales the deadline of all HI-criticality tasks at LO-criticality mode. This factor will be calculated in such a way that the correctness of the system can be guaranteed at both LO- and HI-criticality modes (Subsections 7.4.3 and 7.4.4).

7.4.1 EDF-VD and GEDF-VD: An Overview

EDF-VD. In case of a mode switch (LO to HI), to generate an *MC-correct schedule* (Definition 1), a scheduler must ensure that all HI-criticality tasks meet their deadlines (while LO-criticality tasks can be sacrificed). To guarantee this criterion, a specific amount of CPU time must be reserved for those HI-criticality tasks even if the system is running at LO-criticality mode. This reservation of time can be achieved by shortening the deadlines of HI-criticality tasks under normal mode—those are virtual deadlines.

In EDF-VD, deadlines of all HI-criticality tasks are shortened by multiplying them with a *scaling factor,* and this updated deadline is called the *virtual deadline*. During run-time (at LO-criticality mode), all HI-criticality (LO-criticality) tasks are executed according to their virtual (actual) deadline, according to the EDF order. Upon a mode switch, only the HI-criticality tasks are executed in the EDF order w.r.t their actual/original deadlines.

In the case of a LO- to HI-criticality mode-switch, a HI-criticality task demands additional computational requirements. Setting a virtual deadline for the HI-criticality tasks leaves enough time to finish the extra workload within their actual deadlines. If the virtual deadline is too short, it increases the system density at normal (i.e., LO-criticality) mode. In contrast, a large virtual deadline threatens the schedulability of the system after a LO- to HI-criticality mode switch. The trick is to determine a balanced scaling factor x, such that the correctness under both execution modes can be guaranteed. [14] showed the steps to calculate the minimum x that guarantees the schedulability of all tasks in the system. They also proved the improvement in system schedulability by reducing the deadline for HI-criticality tasks at LO-criticality mode.

Remark 9. *In this work, we consider a completely different gang task workload model in a multi-core platform. As a result, the approach proposed by Baruah et al. [14] to calculate the scaling factor x, as well as the schedulability test, are no longer applicable for our case. We propose a novel approach to calculate a feasible scaling factor x in this section.*

GEDF-VD. Now, we provide an overview of our algorithm (GEDF-VD) considering an implicit-deadline sporadic MC gang task system τ to be scheduled on M identical cores. The GEDF-VD algorithm starts by checking whether GEDF can successfully schedule the *regular task system*. A regular task system denotes that, all LO-criticality tasks will execute up-to their LO-criticality WCET and all HI-criticality tasks will execute up-to their HI-criticality WCET. It returns *SUCCESS* immediately if the regular task system is schedulable. Otherwise, all HI-criticality tasks can execute up-to their LO-criticality WCETs and their deadline is shortened (i.e., virtual deadline) and set to $\hat{T}_i = xT_i$, while all LO-criticality tasks execute up-to their LO-criticality WCETs with their original deadline. If any of the currently executing job (of a HI-criticality task) executed beyond its LO-criticality WCET and did not signal its completion by $\hat{T}_i$, the scheduler immediately discards all currently active LO-criticality jobs. Also, the deadline for all HI-criticality jobs is changed to their release time plus their actual deadline. Subsection 7.4.2 provides a detailed description of GEDF-VD algorithm.

7.4.2 GEDF-VD: A Detailed Description

In this subsection, we describe the GEDF-VD scheduling approach in a two-phase process. First, we describe what happens prior to run-time (denoted as a *pre-processing phase*). In this phase,

Algorithm 3: GEDF-VD (online part)

```
Input: A dual-criticality task-set $\tau = \{\tau_1, \tau_2, \ldots, \tau_n\}$ and a feasible $x$.
/* Handling tasks at run-time */
Whenever a job is released by tasks $\tau_i$ at time instant $t$
if $\tau_i \in \tau_{\text{HI}}$ then
    $d_{i,j} = t + xT_i$;
end
if $\tau_i \in \tau_{\text{LO}}$ then
    $d_{i,j} = t + T_i$;
end
Schedule all active jobs by GEDF according to $d_{ij}$'s.
if $\exists \tau_{i,j} \in \tau_{\text{HI}}$ that is not finished by $d_{i,j}$ at time t' then
    /* Mode Switch */
    for $\forall \tau_i \in \tau_{\text{HI}}$ do
        $d_{i,j} = d_{i,j} + (1 - x)T_i$
    end
    Discard all $\tau_i \in \tau_{\text{LO}}$
    Schedule $\tau' = \{\tau_{\text{HI}}\}$ by GEDF.
end
```

GEDF-VD determines whether (or not) it is required to set a virtual deadline for the HI-criticality tasks. A lower and an upper bound of the virtual deadline is also calculated in this phase. Then, we discuss how the jobs are scheduled at run-time (denoted as *handling the dispatched jobs at run-time*). We present the pseudo-code for (the run-time part of) GEDF-VD in Algorithm 3.

Pre-processing phase. In this phase, we perform a schedulability test for *ordinary (non-MC)* GEDF to determine whether (or not) it can successfully schedule: (i) all $\tau_i \in \tau_{\text{LO}}$ up-to their LO-criticality WCET (c_i^{LO}), and (ii) all $\tau_i \in \tau_{\text{HI}}$ up-to their HI-criticality WCET (c_i^{HI}). If the GEDF test fails, then, for each HI-criticality task $\tau_i \in \tau_{\text{HI}}$, a virtual deadline $\hat{T}_i$ is computed (Step-2), and they execute up-to their LO-criticality WCET (c_i^{LO}).

Step 1. We start by checking whether the task-set can be successfully scheduled by GEDF. If so, then GEDF directly schedules the system. Else, we modify the task deadlines (Step 2).

Step 2: An additional *virtual deadline* parameter $\hat{T}_i$ is calculated for each HI-criticality task τ_i, where $\hat{T}_i = xT_i$. A schedulability test for GEDF-VD is provided next. Furthermore, when the schedulability test is passed, x can be arbitrarily chosen from the range $[A, B]$ while GEDF-VD is guaranteed to generate an MC-correct schedule, where A and B are defined and can be easily calculated for any given system by the following equations:

$$A = \max\{A_1, A_2\}; \tag{7.1}$$

$$A_1 = \max_{i:\tau_i \in \tau_{\text{LO}}} \left\{ \frac{U_{\text{HI}}^{\text{LO}}}{M - \Delta_i^{\text{LO}} - U_{\text{LO}}^{\text{LO}}} \right\}; \tag{7.2}$$

$$A_2 = \max_{i:\tau_i \in \tau} \left\{ \frac{m_i^{\text{LO}} U_{\text{HI}}^{\text{LO}} + u_i^{\text{LO}}(M - \Delta_i^{\text{LO}} - m_i^{\text{LO}})}{m_i^{\text{LO}}(M - \Delta_i^{\text{LO}} - U_{\text{LO}}^{\text{LO}})} \right\}; \tag{7.3}$$

$$B = \min_{i:\tau_i \in \tau_{\text{HI}}} \left\{ 1 - \frac{m_i^{\text{HI}} U_{\text{HI}}^{\text{HI}} + u_i^{\text{HI}}(M - \Delta_i^{\text{HI}} - m_i^{\text{HI}})}{m_i^{\text{HI}} \times (M - \Delta_i^{\text{HI}})} \right\}. \tag{7.4}$$

A schedulability test for GEDF-VD. The following theorem provides a sufficient schedulability test for GEDF-VD.

Theorem 11. *An MC gang task system is schedulable under GEDF-VD upon M identical unit-speed processors if both conditions hold:*

$$U_{\text{LO}}^{\text{LO}} < M - \max_i\{\Delta_i^{\text{LO}}\}, \tag{7.5}$$

$$A \leq B. \tag{7.6}$$

We will prove this theorem later by proving Lemmas 12 and 13 in Subsections 7.4.3 and 7.4.4.

Recall that, $\Delta_i^{\text{LO}} < M$ and $\Delta_i^{\text{HI}} < M$ for all i. Therefore, $m_i U_{\text{HI}}^{\text{LO}} + u_i^{\text{LO}}(M - \Delta_i^{\text{LO}} - m_i^{\text{LO}}) =$

$m_i^{\text{LO}}(U_{\text{HI}}^{\text{LO}} - u_i^{\text{LO}}) + u_i^{\text{LO}}(M - \Delta_i^{\text{LO}}) > 0$, which with (7.5) together implies $A > 0$; and also $m_i^{\text{HI}}U_{\text{HI}}^{\text{HI}} + u_i^{\text{HI}}(M - \Delta_i^{\text{HI}} - m_i^{\text{HI}}) = m_i^{\text{HI}}(U_{\text{HI}}^{\text{HI}} - u_i^{\text{HI}}) + u_i^{\text{HI}}(M - \Delta_i^{\text{HI}}) > 0$, which implies $B < 1$. Thus, both (7.5) and (7.6) being true implies that $0 < A \leq B < 1$, which guarantees that any x chosen from $[A, B]$ must be a valid scaling factor such that $0 < x < 1$.

Run-time dispatch. Similar to GEDF, at any specific time instant, a task with the earliest deadline gets the highest priority. In case of ties, task with a smaller index is favored. Let a binary variable ξ indicate the system-criticality level, then consider the following two possible cases:

Case 1. System is in the LO-criticality mode ($\xi = 0$), j^{th} job of task τ_i arrives at time t:

(i) If τ_i is a LO-criticality task, set the deadline as $d_{i,j} = t + T_i$. Else, set $d_{i,j} = t + \hat{T}_i$, where $\hat{T}_i = xT_i$.

(ii) If any of the currently executing jobs executes for more than c_i^{LO} and does not signal completion, then the system switches to the HI-criticality mode (Case 2).

Case 2. While the system is in the HI-criticality mode ($\xi = 1$):

(i) Discard all LO-criticality tasks (or use background scheduling).

(ii) Update the deadline for the currently active HI-criticality jobs into release time (of these jobs) plus their actual relative deadline.

(iii) For any future HI-criticality task τ_i that releases a job at time t, the deadline is set to $t + T_i$.

(iv) When there is an idle instant, switch to the LO-criticality mode (Case 1)[2].

[2]Note that HI-criticality mode exists for certification purposes. Such both directions of mode switch should be unlikely events during run time. Please also refer to the discussions about apriori verification and run-time robustness in Section 7.1.

7.4.3 Proof of Correctness in the LO*-Criticality Mode*

In this subsection, we show that GEDF-VD and its schedulability test given by Theorem 11 are able to guarantee MC correctness at LO-criticality mode.

Lemma 12. *If both* (7.5) *and* (7.6) *are true, GEDF-VD guarantees that all* LO*-criticality tasks meet their deadlines and all* HI*-criticality tasks meet their* virtual *deadlines during* LO*-criticality mode.*

Proof. Dong et al. [46] have proved that, given any real-time implicit deadline sporadic gang task system τ, GEDF can schedule it successfully if

$$\begin{aligned} U_{sum} &\leq (M - \Delta_i) \times (1 - \frac{u_i}{m_i}) + u_i \\ \Longleftrightarrow U_{sum} &\leq M - \Delta_i + u_i(1 - \frac{M - \Delta_i}{m_i}) \end{aligned} \tag{7.7}$$

holds for all $\tau_i \in \tau$ (refer to Theorem 2). The virtual deadline increases the utilization of these HI-criticality tasks (and hence the whole system). Note that, in the LO-criticality mode, each HI-criticality task is scheduled by its virtual relative deadline xT_i while each LO-criticality task is scheduled by its actual deadline T_i. Therefore, it is sufficient to view each LO-criticality task as a sporadic task with utilization u_i^{LO} and view each HI-criticality task as a sporadic task with utilization u_i^{LO}/x, in order to meet every LO-criticality deadline and every HI-criticality virtual deadline in LO-criticality mode. Then, for every i such that $\tau_i \in \tau$, we discuss the two cases for $M - \Delta_i^{\text{LO}} - m_i^{\text{LO}}$. Therefore, it suffice to evaluate (7.7) under such utilizations for every task τ_i. We show this by two cases: 1) $\tau_i \in \tau_{\text{HI}}$, and 2) $\tau_i \in \tau_{\text{LO}}$.

Case 1: $\tau_i \in \tau_{\text{HI}}$**.** In this case, using (7.7) as a result from [46], we just need the following inequality to hold for any $\tau_i \in \tau_{\text{HI}}$.

$$\begin{aligned} & U_{\text{LO}}^{\text{LO}} + \frac{U_{\text{HI}}^{\text{LO}}}{x} \leq M - \Delta_i^{\text{LO}} + \frac{u_i^{\text{LO}}}{x}(1 - \frac{M - \Delta_i^{\text{LO}}}{m_i^{\text{LO}}}) \\ \iff & \frac{U_{\text{HI}}^{\text{LO}}}{x} + \frac{u_i^{\text{LO}}}{x}(\frac{M - \Delta_i^{\text{LO}}}{m_i^{\text{LO}}} - 1) \leq M - \Delta_i^{\text{LO}} - U_{\text{LO}}^{\text{LO}} \\ \iff & \frac{m_i^{\text{LO}} U_{\text{HI}}^{\text{LO}} + u_i^{\text{LO}}(M - \Delta_i^{\text{LO}} - m_i^{\text{LO}})}{m_i^{\text{LO}} \cdot x} \leq M - \Delta_i^{\text{LO}} - U_{\text{LO}}^{\text{LO}} \end{aligned} \tag{7.8}$$

Notice that (7.5) implies

$$M - \Delta_i^{\text{LO}} - U_{\text{LO}}^{\text{LO}} > 0 \text{ for all } i \text{ such that } \tau_i \in \tau, \tag{7.9}$$

and (7.6) allows $x \in [A, B]$ can be chosen so that $x \geq A$, which, by (7.1) and (7.3), implies

$$x \geq \frac{m_i^{\text{LO}} U_{\text{HI}}^{\text{LO}} + u_i^{\text{LO}}(M - \Delta_i^{\text{LO}} - m_i^{\text{LO}})}{m_i^{\text{LO}}(M - \Delta_i^{\text{LO}} - U_{\text{LO}}^{\text{LO}})} \text{ for all } i \text{ such that } \tau_i \in \tau. \tag{7.10}$$

It is clear that (7.9) and (7.10) imply (7.8).

Case 2: $\tau_i \in \tau_{\text{LO}}$**.** In this case, using (7.7) as a result from [46], we just need the following condition to hold for any $\tau_i \in \tau_{\text{LO}}$.

$$U_{\text{LO}}^{\text{LO}} + \frac{U_{\text{HI}}^{\text{LO}}}{x} \leq M - \Delta_i^{\text{LO}} + u_i^{\text{LO}}(1 - \frac{M - \Delta_i^{\text{LO}}}{m_i^{\text{LO}}}) \tag{7.11}$$

Subcase 2.1: $M - \Delta_i^{\text{LO}} - m_i^{\text{LO}} \leq 0$**.** In this case, $M - \Delta_i^{\text{LO}} \leq m_i^{\text{LO}} \implies 1 - \frac{M - \Delta_i^{\text{LO}}}{m_i^{\text{LO}}} \geq 0$. Therefore, the following inequality implies (7.11):

$$U_{\text{LO}}^{\text{LO}} + \frac{U_{\text{HI}}^{\text{LO}}}{x} \leq M - \Delta_i^{\text{LO}}. \tag{7.12}$$

Notice that (7.5) implies

$$M - \Delta_i^{\text{LO}} - U_{\text{LO}}^{\text{LO}} > 0 \text{ for all } i \text{ such that } \tau_i \in \tau, \tag{7.13}$$

and (7.6) allows $x \in [A, B]$ can be chosen so that $x \geq A$, which, by (7.1) and (7.2), implies

$$x \geq \frac{U_{\text{HI}}^{\text{LO}}}{M - \Delta_i^{\text{LO}} - U_{\text{LO}}^{\text{LO}}} \text{ for all } i \text{ such that } \tau_i \in \tau_{\text{LO}}. \tag{7.14}$$

It is clear that (7.13) and (7.14) imply (7.12)

Subcase 2.2: $M - \Delta_i^{\text{LO}} - m_i^{\text{LO}} > 0$**.** In this case, $M - \Delta_i^{\text{LO}} > m_i^{\text{LO}} \implies 1 - \frac{M - \Delta_i^{\text{LO}}}{m_i^{\text{LO}}} < 0$. So, $\frac{u_i^{\text{LO}}}{x}(1 - \frac{M - \Delta_i^{\text{LO}}}{m_i^{\text{LO}}}) < u_i^{\text{LO}}(1 - \frac{M - \Delta_i^{\text{LO}}}{m_i^{\text{LO}}})$, as $0 < x < 1$. Therefore, the following inequality implies (7.11).

$$U_{\text{LO}}^{\text{LO}} + \frac{U_{\text{HI}}^{\text{LO}}}{x} \leq M - \Delta_i^{\text{LO}} + \frac{u_i^{\text{LO}}}{x}(1 - \frac{M - \Delta_i^{\text{LO}}}{m_i^{\text{LO}}}) \tag{7.15}$$

By the same reasoning as that for Case 1, (7.15) always holds because (7.9) and (7.10) are "for any $\tau_i \in \tau$" and both HI- and LO-criticality tasks are included in the set τ. That is, (7.11) is also true in Case 2.2 here.

Combining Cases 1 and 2 (the latter includes Sub-cases 2.1 and 2.2), the lemma follows. □

7.4.4 Proof of Correctness in the HI*-Criticality Mode*

In this subsection, we show that GEDF-VD and its schedulability test given by Theorem 11 are able to guarantee MC correctness at HI-criticality mode.

Lemma 13. *If both* (7.5) *and* (7.6) *are true, GEDF-VD guarantees that all* HI*-criticality tasks meet their deadlines during* HI*-criticality mode.*

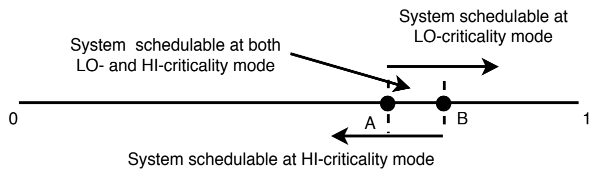

System schedulable at LO-criticality mode
System schedulable at both LO- and HI-criticality mode
0
A
B
1
System schedulable at HI-criticality mode

Figure 7.2: Any value of the scaling factor x, where $A \leq x \leq B$, guarantees an MC-correct schedule.

Proof. At the mode switch point from the lo- to HI-criticality mode, a job from any task $\tau_i \in \tau_{\text{HI}}$ must be either completed or has a deadline at least $(1-x)T_i$ after this mode-switch point; otherwise, an earlier time instant would have been the mode switch point.

Afterwards, any job from any task $\tau_i \in \tau_{\text{HI}}$ has at least T_i time units (which is more than $(1-x)T_i$ as $0 < x < 1$) from their releases in the HI-criticality mode to their corresponding deadlines. Therefore, viewing each task $\tau_i \in \tau_{\text{HI}}$ in the HI-criticality mode as a sporadic task with utilization $\frac{u_i^{\text{HI}}}{(1-x)}$ and using (7.7) as a result from [46], the following inequality is sufficient to ensure that all HI-criticality tasks meet their actual deadlines during HI-criticality mode. For all i such that $\tau_i \in \tau_{\text{HI}}$,

$$m_i^{\text{HI}} \times \frac{U_{\text{HI}}^{\text{HI}}}{(1-x)} \leq m_i^{\text{HI}} \times (M - \Delta_i^{\text{HI}}) - \frac{u_i^{\text{HI}}}{1-x} \times (M - \Delta_i^{\text{HI}} - m_i^{\text{HI}}) \tag{7.16}$$

Notice that (7.6) allows $x \in [A, B]$ can be chosen so that $x \leq B$, which, by (7.4), implies the following equation holds for all i such that $\tau_i \in \tau_{\text{HI}}$:

$$x \leq 1 - \frac{m_i^{\text{HI}} U_{\text{HI}}^{\text{HI}} + u_i^{\text{HI}}(M - \Delta_i^{\text{HI}} - m_i^{\text{HI}})}{m_i^{\text{HI}} \times (M - \Delta_i^{\text{HI}})} \tag{7.17}$$

Furthermore, Equation (7.17) is equivalent to Equation (7.16), as $0 < x < 1$ and $\Delta_i^{\text{HI}} < M$. Thus, the lemma follows. □

Finishing up. We establish Theorem 11 by combining Lemma 12 and 13, and it serves as a sufficient schedulability test for GEDF-VD to schedule MC gang task sets on M identical processors. In addition, Figure 7.2 gives a high-level intuition for validating Theorem 11, given that Lemma 12 and 13 have been proven. Note that we did leverage some insights (in our analysis) from prior works on MC scheduling and gang scheduling. However, considering both of these directions brought increased complexity in our system model. The existing analysis of MC scheduling or gang scheduling is not directly applicable to our work. For example, in the speedup bound analysis for MC scheduling of ordinary sporadic tasks, *an individual task's utilization is at most the speed of a processor* is a straightforward and necessary feasibility condition, while it no longer holds for the gang tasks.

Remark 10. *If we consider that the degree of parallelism of a task τ_i remains unchanged even after a mode-switch, i.e., $m_i^{\text{LO}} = m_i^{\text{HI}} = m_i$, then the correctness proofs become identical to the correctness proofs established in [27].*

7.5 Speed-up Bound Analysis

In this section, we evaluate the effectiveness of our algorithm GEDF-VD based on *speedup bound* metric, which is a widely accepted tool for evaluating the effectiveness of multiprocessor scheduling algorithms [13]. We will first provide the related definition and some existing results, and then (in Subsection 7.5.1) will derive the speedup bound for gang tasks under GEDF algorithm considering the *non-MC* systems. The speedup bound for (non-MC) gang tasks under GEDF scheduling policy, lays the foundation for deriving a speedup bound for MC gang tasks. Finally, in Subsection 7.5.2, considering the *MC* sporadic gang tasks, we prove a speedup bound for our proposed algorithm GEDF-VD. We derive the speedup bound for non-MC gang tasks (Subsection 7.5.1) and the MC gang tasks (Subsection 7.5.2) assuming that the degree of parallelism of a task τ_i does not change after a mode-switch, i.e., $m_i^{\text{LO}} = m_i^{\text{HI}} = m_i$. In this section, Specifically, we present the following findings:

- We prove that the speedup bound [13] for GEDF to gang tasks in a non-MC platform is at most $\big(2 - 1/(M + 1 - \min_i\{m_i\})\big)$, where M denotes the total number of processor cores and m_i denotes the degree of parallelism of task τ_i.
- With the result from the previous step, we then derive a speedup bound of $\sqrt{5}+1$ for GEDF-VD considering MC gang tasks.

Now we provide some definitions and required background.

Definition* 5. *(Speedup factor and speedup bound) *For a scheduler $\mathcal{S}$, a speedup factor $\mathcal{V}$ ($\mathcal{V} \geq 1$) (also known as resource augmentation factor) means that any task set that is schedulable by an optimal scheduler on a platform of speed-1 cores will be schedulable by $\mathcal{S}$ on a platform of speed$-\mathcal{V}$ cores.*

For a scheduler $\mathcal{S}$, a speedup bound refers to the lower bound of the speedup factor $\mathcal{V}$ achievable by it. A speedup bound for a scheduler $\mathcal{S}$ provides an estimation of how far the performance of $\mathcal{S}$ is from an optimal scheduler, and the lower the better.

Limitations. Our speedup factors result in this section rely on the following assumption that

$$m_i \leq \frac{M+1}{2} \text{ for all } \tau_i \in \tau, \tag{7.18}$$

The speedup factors results in this section apply only to systems that satisfy the condition (7.18). Nonetheless, condition (7.18) was not required for the schedulability test and analysis in the last section. Therefore those schedulability results apply to a broader range of MC gang task systems. In practice, condition (7.18) is often satisfied because of the number of cores in modern platforms increases.

Note that gang tasks cannot be scheduled on uniprocessor platforms due to their natures of the *mandatory* parallel processor access request. Therefore, in order to compare with a potential optimal scheduler on a uniprocessor, we propose a *De-ganging* transformation between a multiprocessor gang task set and a corresponding Liu-and-Layland (LL) task set:

- **De-ganging**: Given a gang task set $\tau = \{\tau_1, ..., \tau_n\}$, for each task $\tau_i = \{m_i, c_i, T_i\}$, construct m_i LL tasks $\{\tau_i^{'(1)}, ..., \tau_i^{'(m_i)}\}$, each with the same execution length and period, i.e., $\tau_i^{'(j)} = \{c_i, T_i\}$ for any $j = 1, ..., m_i$. For mapping of the other way around, any deganged LL task set can be clustered into n groups, where there are m_i tasks from the i^{th} group sharing the same execution time c_i and the same period T_i, resulting in a gang task $\tau_i = \{m_i, c_i, T_i\}$ of the same "total" utilization. The extension to MC task set is trivial—treat c_i as a vector and maintain the values during the transfer.

A moment thought should convince the reader that it suffices to restrict our attention to the de-ganged LL task set when deriving the speedup bound, as the de-ganged LL task set being schedulable is *necessary* for the corresponding gang task set to be schedulable. This transformation does not change the overall set utilization and thus does not change the utilization-based necessary schedulability conditions (i.e., basis of the speedup proofs). Throughout the proofs in this section, the following Greek letters will be used frequently:

$$\begin{aligned} &\psi = M/(2 - \frac{1}{M}); \\ &\phi = \frac{\sqrt{5}+1}{2} \text{ (i.e., golden ratio);} \\ &\Phi = \frac{\sqrt{5}-1}{2} \end{aligned} \tag{7.19}$$

7.5.1 Speedup Bound for Gang Tasks under GEDF

In this subsection, we derive the speedup bound (shown in Theorem 14) for the algorithm GEDF, considering the *non-MC gang task set* τ, executing on $\mathcal{V}$-speed cores. This is the first speedup bound result for gang task under GEDF scheduling. This analysis lays the basis for deriving the speedup bound for the proposed MC gang task scheduler.

Theorem 14. *Given any de-ganged task set that is schedulable on a speed-M uni-processor, the corresponding gang task set will pass the schedulabililty test of GEDF upon a M-core system, each of speed $\mathcal{V} = 2 - 1/(M + 1 - \min_i\{m_i\})$.*

Proof. Because $1 \le m_i \le M$ for any i, we know that for all $\tau_i \in \tau$,

$$\begin{aligned} \mathcal{V} &= 2 - \frac{1}{M + 1 - \min_i\{m_i\}} \\ &\ge 2 - \frac{1}{M + 1 - m_i} \\ &= \frac{2M + 1 - 2m_i}{M + 1 - m_i}; \end{aligned} \tag{7.20}$$

From feasibility of the LL task set on a speed-M uniprocessor, we have $U_{sum} \le M$. So,

$$\begin{aligned} (7.20) \implies \mathcal{V} &\ge \frac{U_{sum} + M + 1 - 2m_i}{M + 1 - m_i} \\ \iff \frac{U_{sum}}{\mathcal{V}} &\le M - (m_i - 1) + \frac{(2m_i - M - 1)}{\mathcal{V}} \\ \iff m_i\frac{U_{sum}}{\mathcal{V}} &\le m_iM - m_i(m_i - 1) + \frac{m_i}{\mathcal{V}}(2m_i - M - 1) \\ \implies m_i\frac{U_{sum}}{\mathcal{V}} &\le m_iM - m_i(m_i - 1) + \frac{u_i}{\mathcal{V}}(2m_i - M - 1) \\ & [u_i \le m_i, 2m_i - M - 1 \le 0] \end{aligned} \tag{7.21}$$

The condition $2m_i - M - 1 \le 0$ is equivalent to (7.18); while $u_i \le m_i$ is true for any gang task because the utilization of each gang task τ_i is $u_i = m_i(c_i/T_i)$, where $c_i \leq T_i$. Note that u_i can also be viewed as the total utilization of the m_i de-ganged LL tasks that correspond to the gang task τ_i. Again, de-ganging preserves the utilization of the set. From Equation (7.21):

$$\begin{aligned} m_i \frac{U_{sum}}{\mathcal{V}} &\leq m_i M - (m_i - 1)m_i + (m_i - 1)\frac{u_i}{\mathcal{V}} + \frac{u_i}{\mathcal{V}}(m_i - M) \\ \implies m_i \frac{U_{sum}}{\mathcal{V}} &\leq m_i M - (m_i - 1)(m_i - \frac{u_i}{\mathcal{V}}) - \frac{u_i}{\mathcal{V}}(M - m_i) \\ \implies m_i \frac{U_{sum}}{\mathcal{V}} &\leq m_i M - \Delta_i(m_i - \frac{u_i}{\mathcal{V}}) - \frac{u_i}{\mathcal{V}}(M - m_i) \end{aligned}$$

[From Definition 4: $0 \leq \Delta_i \leq m_i - 1$]

$$\begin{aligned} &= m_i(M - \Delta_i) - (M - \Delta_i - m_i)\frac{u_i}{\mathcal{V}} \text{ [re-arrange]} \\ &= (M - \Delta_i)(m_i - \frac{u_i}{\mathcal{V}}) + m_i\frac{u_i}{\mathcal{V}} \text{ [re-arrange]} \\ \implies \frac{U_{sum}}{\mathcal{V}} &\leq (M - \Delta_i)(1 - \frac{u_i}{m_i\mathcal{V}}) + \frac{u_i}{\mathcal{V}} \\ &\text{[divide } m_i \text{ on both sides, re-arrange], for all i} \end{aligned} \tag{7.22}$$

The equation above implies that the corresponding gang set is GEDF schedulable on M speed-$\mathcal{V}$ processors (Theorem 2 in [46]). Note that the last step is true because under speed of $\mathcal{V}$, all utilizations in the test should be treated as the ones under speed 1 divided by $\mathcal{V}$, in order to apply the original schedulability test under a speed-1 platform. □

Theorem 14 indicates that the speedup factor of the GEDF schedulability test in Theorem 2 of [46] (for gang task set) is no greater than $\mathcal{V} = 2 - 1/(M + 1 - \min_i\{m_i\})$. Because $1 \leq m_i \leq M$ for any i, $\mathcal{V} \leq 2 - \frac{1}{M}$. Therefore, the following corollary follows directly from Theorem 14.

Corollary 1. *Given any de-ganged task set that is schedulable on a speed-M uni-processor, the corresponding gang task set will pass the schedulability test of GEDF upon a M-core system, each of speed* $(2 - \frac{1}{M})$.

Furthermore, scaling all speeds by a factor of $1/(2 - \frac{1}{M})$ lead to the following corollary.

Corollary 2. *Given any de-ganged task set that is schedulable on a speed-ψ uni-processor, the corresponding gang task set will pass the schedulability test of GEDF upon M unit-speed processors, where $\psi = M/(2 - \frac{1}{M})$.*

7.5.2 Speedup Bound for Gang Tasks under GEDF-VD

The previous subsection proved the speedup bound for non-MC task under GEDF. We now brings MC and virtual deadlines into the picture, and derive the speedup bound for *MC gang task set τ* under GEDF-VD. From the definitions of ϕ and Φ in Equation (7.19), the following properties hold:

$$1 + \Phi = \phi = \frac{1}{\Phi} \tag{7.23}$$

$$\Phi + \Phi^2 = 1 \tag{7.24}$$

Theorem 15. *Given any de-ganged MC task set that is schedulable on a speed-$(\psi \cdot \Phi)$ uniprocessor, the corresponding MC gang task set will be schedulable under GEDF-VD upon M unit-speed processors, where $\psi = M/(2 - \frac{1}{M})$ and $\Phi = \frac{\sqrt{5}-1}{2}$.*

Proof. The de-ganged MC task set being schedulable on a speed-$(\psi \cdot \Phi)$ uni-processor implies

$$\max\{U_{\text{LO}}^{\text{LO}} + U_{\text{HI}}^{\text{LO}}, U_{\text{HI}}^{\text{HI}}\} \leq \psi \cdot \Phi. \tag{7.25}$$

We proceed the rest of this proof in two cases.

Case 1: $U_{\mathrm{HI}}^{\mathrm{LO}} \geq \Phi \cdot U_{\mathrm{LO}}^{\mathrm{LO}}$. By (7.25) and the condition of Case 1,

$$\psi \cdot \Phi \geq U_{\mathrm{LO}}^{\mathrm{LO}} + U_{\mathrm{HI}}^{\mathrm{LO}} \geq (1+\Phi)U_{\mathrm{LO}}^{\mathrm{LO}}$$
$$\Longrightarrow U_{\mathrm{LO}}^{\mathrm{LO}} \leq \frac{\Phi}{1+\Phi} \cdot \psi = \Phi^2 \cdot \psi. \text{ [by (7.23)]}$$

Then, by the above and (7.25),

$$U_{\mathrm{LO}}^{\mathrm{LO}} + U_{\mathrm{HI}}^{\mathrm{HI}} \leq \Phi^2 \cdot \psi + \Phi \cdot \psi = \psi. \text{ [by (7.24)]}$$

Thus, no virtual deadline needs to be set at all. Both HI- and LO-criticality tasks are scheduled by GEDF according to their actual deadlines on M unit-speed processors. By Corollary 2, no deadline will be missed.

Case 2: $U_{\mathrm{HI}}^{\mathrm{LO}} < \Phi \cdot U_{\mathrm{LO}}^{\mathrm{LO}}$. By (7.25) and the condition of Case 2,

$$\psi \cdot \Phi \geq U_{\mathrm{LO}}^{\mathrm{LO}} + U_{\mathrm{HI}}^{\mathrm{LO}} > (\frac{1}{\Phi}+1)U_{\mathrm{HI}}^{\mathrm{LO}}$$
$$= \frac{1+\Phi}{\Phi} \cdot U_{\mathrm{HI}}^{\mathrm{LO}} = \frac{1}{\Phi^2} \cdot U_{\mathrm{HI}}^{\mathrm{LO}}. \text{ [by (7.23)]}$$

That is,

$$U_{\mathrm{HI}}^{\mathrm{LO}} < \Phi^3 \cdot \psi. \tag{7.26}$$

Then, we have

$$\begin{aligned}
U_{\text{LO}}^{\text{LO}} + \frac{U_{\text{HI}}^{\text{LO}}}{1 - U_{\text{HI}}^{\text{HI}}/\psi} &= U_{\text{LO}}^{\text{LO}} + U_{\text{HI}}^{\text{LO}} + U_{\text{HI}}^{\text{LO}} \cdot \frac{U_{\text{HI}}^{\text{HI}}/\psi}{1 - U_{\text{HI}}^{\text{HI}}/\psi} \\
&\leq U_{\text{LO}}^{\text{LO}} + U_{\text{HI}}^{\text{LO}} + U_{\text{HI}}^{\text{LO}} \cdot \frac{\Phi}{1 - \Phi} \\
&[U_{\text{HI}}^{\text{HI}} \leq \psi \cdot \Phi \text{ by (7.25)}] \\
&= U_{\text{LO}}^{\text{LO}} + U_{\text{HI}}^{\text{LO}} + U_{\text{HI}}^{\text{LO}} \cdot \frac{\Phi}{\Phi^2} \text{ [by (7.24)]} \\
&< \psi \cdot \Phi + \Phi^3 \cdot \psi \cdot \frac{\Phi}{\Phi^2} \text{ [by (7.25) and (7.26)]} \\
&= (\Phi + \Phi^2)\psi \text{ [rearrange]} = \psi, \text{ [by (7.24)]}
\end{aligned}$$

which is concluded as

$$U_{\text{LO}}^{\text{LO}} + \frac{U_{\text{HI}}^{\text{LO}}}{1 - U_{\text{HI}}^{\text{HI}}/\psi} < \psi. \tag{7.27}$$

In this case, one could take $x = \frac{U_{\text{HI}}^{\text{LO}}}{\psi - U_{\text{LO}}^{\text{LO}}}$ as the scaling factor to set the virtual deadlines for HI-criticality tasks. Because the de-ganged task set is schedulable on a speed-($\psi \cdot \Phi$) uniprocessor, $U_{\text{LO}}^{\text{LO}} \leq \psi \cdot \Phi$, which implies $x > 0$, as $\Phi < 1$, $\psi > 0$, and $U_{\text{HI}}^{\text{LO}} > 0$. On the other hand, $U_{\text{HI}}^{\text{LO}} < \Phi \cdot U_{\text{LO}}^{\text{LO}}$ in Case 2, so

$$\begin{aligned}
x = \frac{U_{\text{HI}}^{\text{LO}}}{\psi - U_{\text{LO}}^{\text{LO}}} &< \frac{\Phi \cdot U_{\text{LO}}^{\text{LO}}}{\psi - U_{\text{LO}}^{\text{LO}}} \\
\leq \frac{\Phi \cdot \psi \cdot \Phi}{\psi - \psi \cdot \Phi} = \frac{\Phi^2}{1 - \Phi} &= 1. \text{ [by (7.24)]}
\end{aligned}$$

Thus, in this case, $0 < x < 1$ is guaranteed under this particular setting and therefore this x can always be used as the scaling factor to set the virtual deadlines for GEDF-VD. Then, we first show that all LO-criticality tasks meet their actual deadlines and all HI-criticality tasks meet their *virtual* deadlines during the LO-criticality mode.

$$U_{\text{LO}}^{\text{LO}} + \frac{U_{\text{HI}}^{\text{LO}}}{x} = U_{\text{LO}}^{\text{LO}} + \psi - U_{\text{LO}}^{\text{LO}} = \psi.$$

By Corollary 2, the above equation implies that using GEDF-VD to schedule the gang task set on M unit-speed processors, all LO-tasks meet their actual deadlines and all HI-tasks meet their *virtual* deadlines during the LO-mode. Next, we show that all HI-criticality tasks, including any carryover (HI-) jobs across the mode-switch point, meet their *actual* deadlines during the HI-criticality mode.

Because the virtual deadlines are set as $x \cdot T_i$ for each HI-criticality task τ_i, every HI-criticality job including the one triggering the mode switch will have *at least* $(1-x)T_i$ time units to finish its *at most* C_i^{HI} execution and to release its next job. It suffices to consider the schedulability when replacing each HI-criticality task in the HI-criticality mode by a implicit-deadline sporadic task with period $(1-x)T_i$ and execution C_i^{HI}. It can be done by checking their total utilization

$$\sum_{\tau_i \in \tau_{\text{HI}}} \frac{C_i^{\text{HI}}}{(1-x)T_i} = \frac{U_{\text{HI}}^{\text{HI}}}{1-x}.$$

On the other hand, by (7.27), we have

$$\frac{U_{\text{HI}}^{\text{LO}}}{\psi - U_{\text{LO}}^{\text{LO}}} < 1 - U_{\text{HI}}^{\text{HI}}/\psi,$$

and $1 - x > U_{\text{HI}}^{\text{HI}}/\psi$ holds since we set $x = \frac{U_{\text{HI}}^{\text{LO}}}{\psi - U_{\text{LO}}^{\text{LO}}}$. Thus,

$$\frac{U_{\text{HI}}^{\text{HI}}}{1-x} < \frac{U_{\text{HI}}^{\text{HI}}}{U_{\text{HI}}^{\text{HI}}/\psi} = \psi.$$

By Corollary 2, the above equation implies that using GEDF-VD to schedule the gang task-set on M unit-speed processors, all HI-criticality tasks, including any carryover (HI-) jobs across the mode-switch point, meet their *actual* deadlines during the HI-criticality mode. $\square$

Finally, we can easily use Theorem 15 to derive a speedup bound for GEDF-VD to schedule MC gang task sets on identical processors, as stated in the following theorem.

Theorem 16. *If any potentially optimal algorithm can schedule an MC gang task set on M unit-speed processors, GEDF-VD can schedule the same task set on M speed-$(\sqrt{5}+1)$ processors.*

Proof. Theorem 15 directly implies that: *If any potentially optimal algorithm can schedule an MC gang task set on M speed-$(\psi \cdot \Phi/M)$ processors, GEDF-VD is able to schedule the same MC gang task set on M unit-speed processors.* This is because for a MC gang task set to be schedulable on M speed-$(\psi \cdot \Phi/M)$ processors, it is necessary for its corresponding de-ganged MC task set to schedulable on a speed-$(\psi \cdot \Phi)$ uniprocessor. Note that, by definitions: $\psi = \frac{M}{2-\frac{1}{M}}$ and $\Phi = \frac{\sqrt{5}-1}{2}$, the following statement is true:

> If any potentially optimal algorithm can schedule a MC gang task set on M speed-$(\frac{1}{2-\frac{1}{M}} \cdot \frac{\sqrt{5}-1}{2})$ processors, GEDF-VD is able to schedule the same MC gang task set on M unit-speed processors.

Scaling the speed unit up by $(2-\frac{1}{M})\frac{\sqrt{5}+1}{2}$ (please note that $\frac{\sqrt{5}-1}{2} \cdot \frac{\sqrt{5}+1}{2} = 1$), the above statement can be re-written as:

> If any potentially optimal algorithm can schedule a MC gang task set on M unit-speed processors, GEDF-VD is able to schedule the same MC gang task set on M speed-$(2-\frac{1}{M})\frac{\sqrt{5}+1}{2}$ processors.

Since $(2-\frac{1}{M})\frac{\sqrt{5}+1}{2} < \sqrt{5}+1$, the theorem follows. $\square$

7.6 Evaluation

In this section, we evaluate the performance of GEDF-VD through simulation results. While the simulation results provide some representation of the proposed scheduling's performance, they may not represent the exact behavior of our proposed approach in real systems for several reasons. For example, memory plays a vital role from an implementation point of view and needs to be available and allocated to parallel threads. Although a recent work [35] has implemented a preemptive EDF scheduler for GPU tasks providing bandwidth isolation, MC scheduling on the GPU platform (with preemptive EDF) and the memory partitioning technique to the gang tasks are yet to be explored. In the future, we plant to explore implementation and experimentation on a real hardware platform. As our work is the first to propose MC gang task scheduling, there is no perfect baseline for comparison. We have performed many experiments by varying different factors to observe the efficiency of our algorithm.

7.6.1 Experimental Setup

Workload generation. We generate MC gang tasks based on the following parameters.

• M : The number of processor cores.

• $m_{min}, m_{max}, m_{avg}$: The minimum, maximum, and average value for m (i.e., degree of parallelism), respectively. We generate the task set by varying these three parameters, where $m_{min}, m_{max} \in [1, M]$ and $m_{min} \leq m_{avg} \leq m_{max}$.

• U_{avg} : The average utilization for the task set. We have varied U_{avg} value from $0.05 \times M$ to $0.95 \times M$ with $0.05 \times M$ difference at each step.

• $P_{\text{HI}} = 0.5$: The probability of a task $\tau_i \in \tau_{\text{HI}}$.

• R: Denotes the maximum ratio of u_i^{HI} to u_i^{LO}, where $R \in [4, 8]$. We generate u_i^{HI} uniformly from $[u_i^{\text{LO}}, R \times u_i^{\text{LO}}]$.

At first, for a specific value of n (number of tasks per task set), we generate the m values for each task. m is uniformly generated from $[m_{min}, m_{max}]$ range in a way so that the average m for all tasks remains equal to m_{avg}. Next, for a specific value of average utilization U_{avg}, we calculate the average utilization u_i^a for each task by following the log-normal distribution. Note that, for n number of gang tasks, there are total $\sum_{i=1}^{n} m_i = m_{avg} \times n$ amount of single task instances in each task set. For the sake of a proper distribution, we extend the *UUniFast algorithm* [29] for Gang task. We use log-normal distribution over $\sum_{i=1}^{n} m_i$ task instances similarly as UUnifast, but for a single task, we take the average of all of its instances as the task's average utilization. The values of u_i^{LO} is uniformly generated from $[\frac{2\times u_i^a}{R+1}, u_i^a]$ so that the value of u_i^{HI} is always in the range $[u_i^{\text{LO}}, R \times u_i^{\text{LO}}]$.

Simulation setup. We performed the simulation for average utilization ranging from $0.05M$ to $0.95M$ with a step size of $0.05M$. For each average, 100 task sets (each with 10 tasks) are generated.

7.6.2 Evaluation Results

We execute a set of gang tasks under our proposed algorithm by varying different parameters, and present the simulation results in Figure 7.3, 7.4, 7.5, 7.6, 7.7, 7.8, 7.9, and in Table 7.2. Note that, in Table 7.2, and in Figure 7.4-7.9, we consider that the degree of parallelism (of the HI-criticality tasks) does not change after a mode-switch.

Effect of changing the degree of parallelism after a mode switch. In this experiment, we set the value for M to 8, and vary (i.e., increase or unchanged) the degree of parallelism of a HI-criticality task after a mode-switch. To incorporate the change in the task model, i.e., the degree of parallelism of a HI-criticality task can change after a mode-switch, we slightly modify the simulation setup

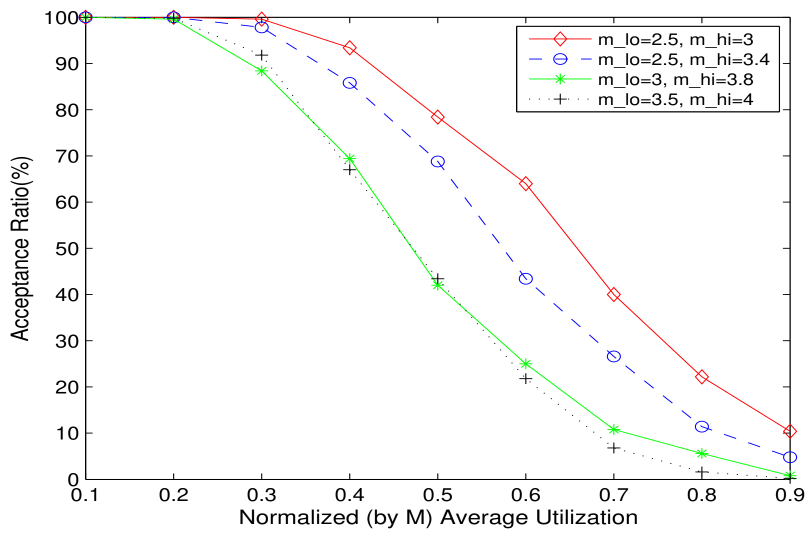


Figure 7.3: Acceptance ratio for GEDF-VD with a different (after a mode-switch) average degrees of parallelism.

Table 7.2: Acceptance ratio for different amount of tasks generated under various average utilization and R value.

$U_{avg} \rightarrow$ # of tasks$\downarrow$	R	2	2.5	3	3.5	4	4.5	5
8	4	100	100	97	59	5	5	4
12	4	100	99	94	65	2	2	2
16	4	100	100	98	50	0	0	0
8	8	97	75	67	31	23	19	11
12	8	100	100	74	58	40	17	6
16	8	100	98	91	77	48	34	11

described in Subsection 7.6.1. We report the acceptance ratio (in Figure 7.3) with a different average of m_i^{LO} (i.e., from 2.5 to 3.5) and m_i^{HI} (i.e., from 3 to 4) values. This figure reports that the *acceptance ratio* (i.e., the ratio of the number of schedulable task sets over the total number of task sets) decreases when the degree of parallelism increases, which can be explained by Equations (7.8) and (7.16). That is, a higher value of m_i^{LO} or m_i^{HI} inversely affects the acceptance ratio.

Effect of changing the degree of parallelism in a range with lower difference. In this set of experiments, for $M = 8$, we vary a task's degree of parallelism (m) in a different range, while the difference between the upper and lower bound in each range is fixed. The acceptance ratio under varying degree of parallelism (and different R values) is reported in Figure 7.4 and Figure 7.5. These figures indicate that in boundary cases (where the degree of parallelism is very low or very high) acceptance ratio changes proportionally with respect to the degree of parallelism. This behavior can be explained with the help of Equations (7.8) and (7.16). When m increases or decreases by a large amount, acceptance ratio will increase or decrease respectively. However, for a small change of m, acceptance ratio may not change proportionally. This is because the schedulability conditions provided by Equations (7.8) and (7.16) are also effected by the maximum number of idle cores (Δ_i), which is dependent on m.

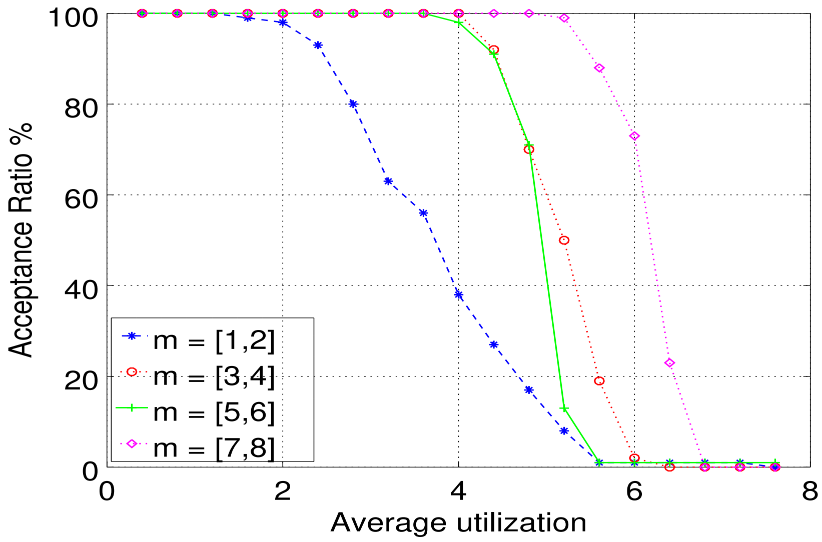


Figure 7.4: Acceptance ratio for GEDF-VD in an 8-core platform with $R = 4$, and under same ranges of degrees of parallelism.

Effect of changing the total number of cores. In Figure 7.6 and Figure 7.7, we report the acceptance ratio of the task set by varying the number of cores in the system, M. In this set of

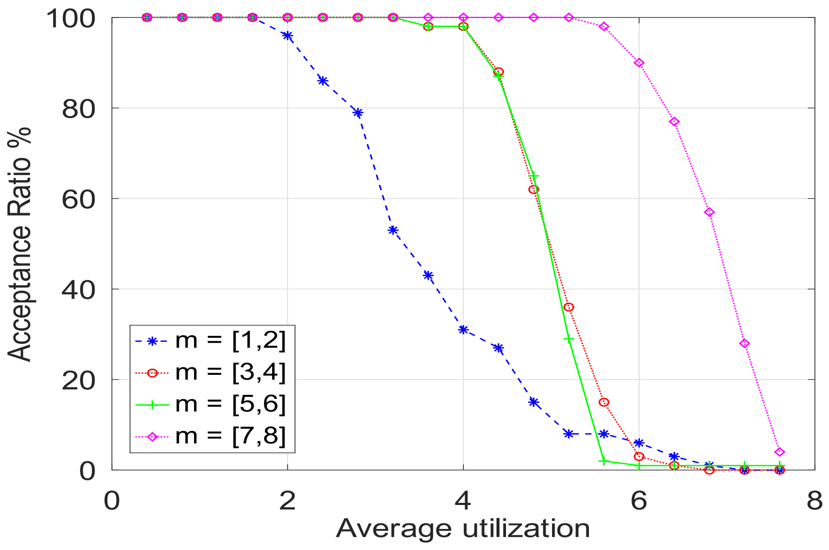


Figure 7.5: Acceptance ratio for GEDF-VD in an 8-core platform with $R = 8$, and under same ranges of degrees of parallelism.

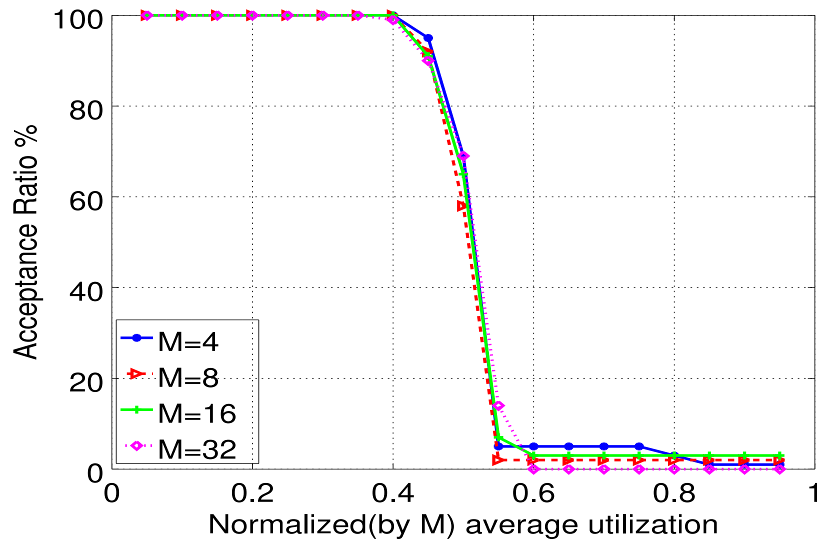


Figure 7.6: Acceptance ratio for GEDF-VD in an M-core platform (with $R = 4$).

experiments, we set a value for m_{avg}, which is uniformly generated from a range of $[\frac{M}{2}, \frac{3M}{4}]$. Simulations are conducted for $M = 4, 8, 16$, and 32 and the average utilization is weighted with

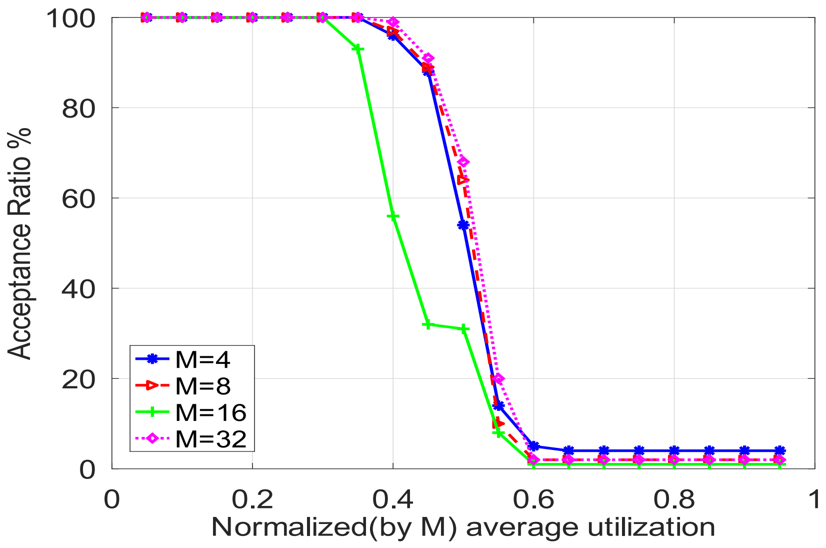


Figure 7.7: Acceptance ratio for GEDF-VD in an M-core platform (with $R = 8$).

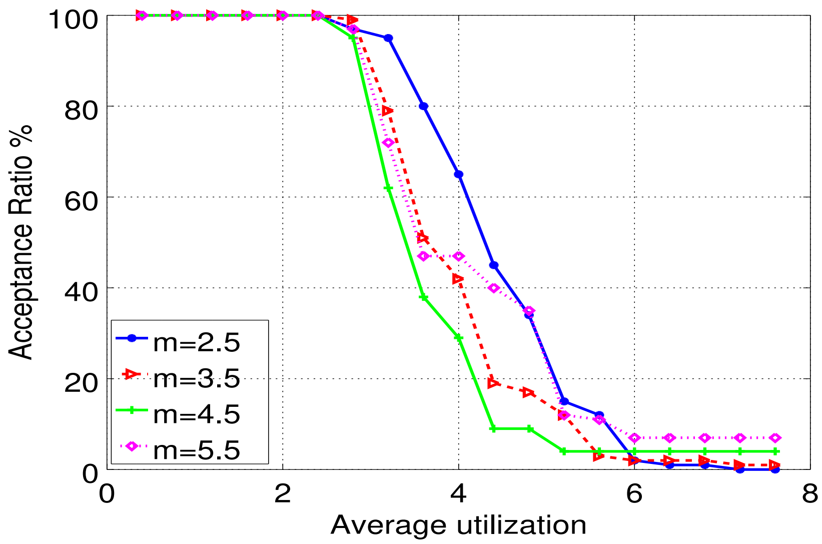


Figure 7.8: Acceptance ratio for GEDF-VD in an 8-core platform with $R = 4$ and a varying range of m_{avg}.

respect to the value of M. Figure 7.6 and Figure 7.7 shows that the acceptance ratio is not affected by different values of M and remains almost unchanged.

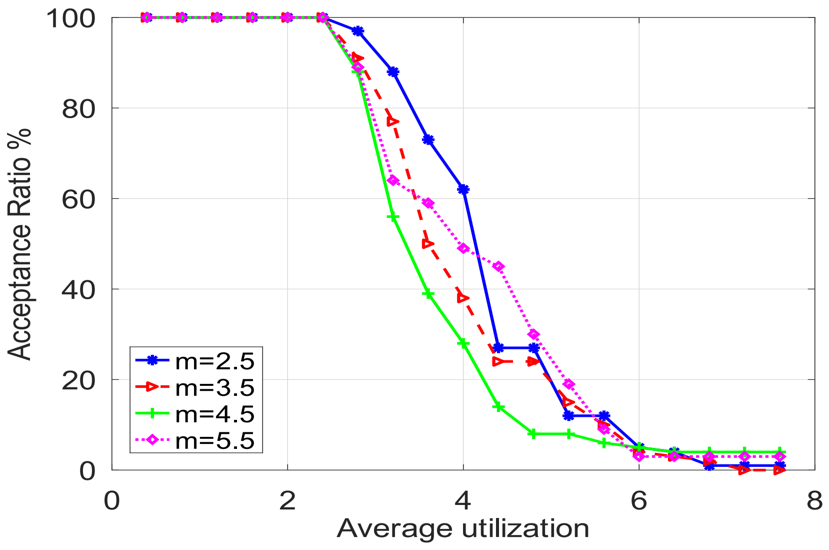


Figure 7.9: Acceptance ratio for GEDF-VD in an 8-core platform with $R = 8$ and a varying range of m_{avg}.

Effect of changing number of tasks per task set. In this set of experiments, we have randomly generated 100 task sets with different R values and 8, 12, and 16 tasks per task set (with U_{avg} changing from 2 to 5 with a step size of 0.5) and report the acceptance ratio in Table 7.2. From the reported data, it is clear that the acceptance ratio of the task set is not affected by the number of tasks per set. This result indicates the effectiveness of our proposed algorithm under a varying number of tasks in a task set.

Effect of changing m_{avg} value. In Figure 7.8 and Figure 7.9, we show the acceptance ratio by varying m_{avg} in an 8-core platform. The result does not demonstrate a direct relationship between m_{avg} and the acceptance ratio.

7.7 Conclusion

WCET measurements are pessimistic due to increased uncertainty. So, there is an emerging need to introduce MC into parallel computation models and system designs. This chapter discusses an initial step of more substantial efforts in bringing richer system modeling and analysis into the emerging need in many applications for parallel computing and MC. In this chapter, we leverage two existing algorithms (EDF-VD and GEDF) to schedule MC gang tasks efficiently and discuss the correctness criteria of our approach. We derive the first speedup bound for GEDF schedulability of (non-MC) gang tasks and further derived the bound for GEDF-VD of MC gang tasks.

CHAPTER 8: ENERGY EFFICIENT PRECISE SCHEDULING OF MIXED-CRITICALITY TASKS

Many embedded systems have hard real-time constraints that must be satisfied to guarantee the correctness of the system behavior. Missing a time deadline is, in fact, typically considered as a system failure. To ensure the timing correctness of a system, a schedulability test is performed to verify that no task misses any deadline under any condition. The task models are usually characterized by the arrival pattern, (periodic, sporadic, or aperiodic, etc.,) the deadline, and the execution time. In particular, most of the state of the art schedulability analysis considered that a task can execute up to its Worst-Case Execution Time (WCET). During the real execution, however, a task rarely needs to execute up to its WCET [111]. For a large class of real-life applications, WCET based schedulability analysis is often proved to be very pessimistic [84], as the task execution pattern may show great variability (w.r.t. time). Consequently, designing a system with the assumption that a task will execute up to its WCET, may lead to system over-provisioning, low utilization, high costs, and excessive power/energy consumption [88]. This is in contrast with the requirement of improving performance while maintaining the non-functional property at an acceptable level.

To efficiently utilize the non-negligible gap between the WCET and the actual execution time, and to minimize energy consumption, resource over-provisioning, and cost, the Mixed-Criticality (MC) framework [122] received attention from a wide community. In an MC setup, different software

The contents of this chapter have been previously published and available at
1. Bhuiyan, A., Sruti, S., Guo, Z., & Yang, K. (2019, November). Precise scheduling of mixed-criticality tasks by varying processor speed. In Proceedings of the 27^{th} International Conference on Real-Time Networks and Systems (pp. 123-132)
2. Yang, K., Bhuiyan, A., & Guo, Z. (2020, November). F2VD: Fluid rates to virtual deadlines for precise mixed-criticality scheduling on a varying-speed processor. In 2020 IEEE/ACM International Conference On Computer Aided Design (ICCAD) (pp. 1-9). IEEE.
3. Bhuiyan, A., Reghenzani, F., Fornaciari, W., & Guo, Z. (2020). Optimizing energy in non-preemptive mixed-criticality scheduling by exploiting probabilistic information. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 39(11), 3906-3917.

components with different criticality levels are integrated into a common platform. To each task, a criticality level is assigned together with multiple execution time thresholds (at different certification/pessimism levels), as described by Vestal's seminal paper [122]. This value is inspired by the industry standards for safety-critical systems: for instance, the DO-178C avionic software standard [1] sets 5 levels of criticality: $\{A, B, C, D, E\}$, where A is the criticality level referring to functions that may cause catastrophic failures, while at level E, to functions that do not affect safety. In real-time computing, this value is interpreted as the level of assurance of the WCET. To illustrate this concept, let us consider a dual-criticality system, where the tasks are classified in LO-criticality and HI-criticality. The tasks belonging to the latter category have two values for the WCET: where one value is pessimistic, but safe, while the other value is computed with an analysis that provides a lower level of assurance, and hence less pessimistic. In this case, the less pessimistic value may not correctly (over-)estimate the real WCET. Consequently, the execution time of a task may overrun this value. When this condition happens, i.e., the overrun, we say that a *mode switch* occurs.

Existing MC task-set scheduling strategies aimed at: (i) correctly scheduling all the tasks when the system exhibits less pessimistic behaviors (in this case, the system is said to be in LO-criticality mode), and, (ii) correctly scheduling the more important (HI-criticality) tasks under more pessimistic behaviors, while no scheduling guarantee is given to the less important (LO-criticality) tasks (in this case, the system is said to be in HI-criticality mode). The system starts running in LO-criticality mode by scheduling the tasks under optimistic assumptions. When a HI-criticality task violates its assigned execution time threshold, the system switches to the HI-criticality mode, i.e., mode switch, and it drops all the LO-criticality tasks to guarantee the deadlines of HI-criticality tasks. However, discarding (or providing degraded services) to the LO-criticality tasks may result in severe performance loss and it violates the task independence requirements for safety-critical systems [52].

Some recent work in [126, 26] handled the *precise scheduling* of MC systems, where full service is provided to all tasks under both the pessimistic and optimistic assumptions. They incorporated the dynamic voltage frequency scaling (DVFS) scheme to the precise scheduling strategy and derived the energy-aware CPU speed to execute in normal mode. However, the optimized energy consumption is in accordance to the *worst-case* behaviors under the normal mode. The energy consumption should be optimized for the average/expected scenarios instead, and the existing MC task model (with multiple WCETs) does not provide sufficient information to perform such optimization.

Probabilistic Approaches. Probabilistic real-time approaches have been proposed in the last two decades to overcome the problem of the WCET estimation for modern platforms [43, 110]. Most of these approaches are based on the estimation of the distribution tail by exploiting the Extreme Value Theory, a statistical theory to model the probability of extreme events, that in the real-time case it is used for the WCET of the tasks. Unfortunately, the theory is still immature and several challenges yet to be addressed before considering its results reliable [106]. Hence, in this work, we propose to exploit the probabilistic information not to directly estimate the WCET for scheduling analysis purposes, but to use them for the optimization of the system energy consumption. Differently from previous probabilistic energy approaches [105], the focus of this work is the satisfaction of real-time requirements with a *deterministic* approach and on top of that, exploiting the probabilistic information to minimize the expected energy consumption of the system.

Existing works aimed at minimizing energy consumption at LO-criticality mode, but considered a pessimistic assumption that all the tasks execute up to their WCET, at their respective criticality levels [26, 126]. Since a task rarely needs to execute up to its WCET, we integrate the probabilistic based prediction strategy and the DVFS scheme to the precise scheduling of MC tasks. In this chapter we present the following key contributions:

- We present the experimental studies based on randomly generated synthetic tasks (scheduled using the EDF-VD and MCF algorithm, proposed in [26]). The experimental result supports the theoretical findings as well as the effectiveness of the proposed algorithms.

- Then, we present the experimental studies based on randomly generated task sets scheduled using the F2VD algorithm proposed in [126].
- We present an energy-aware scheduling strategy that selects the optimistic WCETs for tasks and the processor speeds under both (LO- and HI-criticality) modes to minimize the overall average energy consumption. This optimization is performed via a novel probabilistic analysis of the execution time, coupled with a dedicated response time analysis which guarantees the timing correctness of all tasks under both the pessimistic and optimistic assumptions.
- Based on a randomly generated task sets, we conduct extensive simulation studies which supports the effectiveness of our algorithm (w.r.t. energy consumption).

The rest of the paper is organized as follows. Section 8.1 describes the task model, existing scheduling policies for the precise MC task model, and a comparison among these schedulers. Section 8.2 describes the probabilistic MC task model, provides a detailed response time analysis for such a task model, and reported the experimental results.

8.1 Traditional MC Task Model

In this section, we describe the traditional MC task model and provided some required definitions (Subsection 8.1.1). Then we present some existing scheduling policies, e.g., EDF-VD, F2VD, and MC-Fluid, proposed to handle the precise MC task model (Subsection 8.1.2). Finally, we report a detailed comparison among these schedulers (Subsection 8.1.3).

8.1.1 System Model

We consider a set of n implicit-deadline sporadic MC tasks $\tau = \{\tau_1, \tau_2, \cdots, \tau_n\}$, where each task is specified by a 3-tuple as $\tau_i = (C_i^{\text{LO}}, C_i^{\text{HI}}, T_i)$. Each task τ_i releases a (potentially infinite) sequence of *jobs* with a minimum release separation of T_i time units and every job has an absolute deadline T_i time units after its release. The worst-case execution requirement, which is defined by the worst-case execution time on a unit-speed processor, of task τ_i is estimated at two criticality levels: a LO-criticality estimate C_i^{LO} and HI-criticality estimate C_i^{HI}, where it is assumed that $\forall i, C_i^{\text{LO}} \leq C_i^{\text{HI}}$. Besides, C_i^{LO} (C_i^{HI}, respectively) is also the execution requirement budgets of task τ_i in the LO(HI, respectively)-*mode*, which is to be described later. In particular, $C_i^{\text{LO}} < C_i^{\text{HI}}$ indicates that task τ_i is a HI-criticality task (HI-task) that may trigger a system mode switch, whereas $C_i^{\text{LO}} = C_i^{\text{HI}}$ indicates that task τ_i is a LO-criticality task (LO-task) that cannot trigger any system mode switch. Furthermore, the j^{th} job of task τ_i is denoted by $J_{i,j}$, whose release time and absolute deadline are denoted by $r_{i,j}$ and $d_{i,j}$, respectively.

The per-mode utilization of each task τ_i are determined as follows:

$$\forall \tau_i, \quad u_i^{\text{LO}} = \frac{C_i^{\text{LO}}}{T_i};$$

$$\forall \tau_i, \quad u_i^{\text{HI}} = \frac{C_i^{\text{HI}}}{T_i}.$$

The total utilization for each mode of operation is represented as follows:

- Since we do not degrade services for LO-criticality tasks in HI-criticality mode, their utilization in both modes of operation remains the same, i.e.,

$$U_{\text{LO}}^{\text{LO}} = U_{\text{LO}}^{\text{HI}} = \sum_{\tau_i \in \tau_{\text{LO}}} u_i^{\text{LO}} = \sum_{\tau_i \in \tau_{\text{LO}}} u_i^{\text{HI}}.$$

- The total utilization for all HI-criticality tasks in LO- and HI-criticality modes can be represented as:

$$U_{\text{HI}}^{\text{LO}} = \sum_{\tau_i \in \tau_{\text{HI}}} u_i^{\text{LO}} \quad \text{and} \quad U_{\text{HI}}^{\text{HI}} = \sum_{\tau_i \in \tau_{\text{HI}}} u_i^{\text{HI}}.$$

We consider the problem of scheduling the set of tasks τ on a single processor whose executing speed may vary. The processor begins with a degraded speed $\rho < 1.0$, which indicates that any workload being executed under this speed for t time units is equivalent to that under a unit-speed processor for $\rho \times t$ time units. During runtime, the amount of workload completed for each job is being monitored. If any job $J_{i,j}$ has done C_i^{LO} workload under the degraded processing speed ρ (thus receiving a cumulative actual execution time of C_i^{LO}/ρ units) but still requires further execution, the system is immediately notified, and the processor starts to perform its full speed 1.0 right from that moment. We also call this moment as the time instant of *mode switch*, from the LO-mode (where the processor speed is ρ) to the HI-mode (where the processor speed becomes 1.0). The system can recover to the LO-mode once the processor becomes idle.

Note that, in contrast to a majority of existing work on MC scheduling, no task is *entirely or partially dropped* upon a mode switch, and every job meets its absolute deadline at any system mode. The difference between the two execution requirement budgets upon the mode switch, i.e., $C_i^{\text{HI}} - C_i^{\text{LO}}$, is compensated by the speed upgrade. Furthermore, any job $J_{i,j}$ that has done C_i^{HI} workload but still not completed yet, is considered as *erroneous* and would be terminated then. That is, only HI-tasks, for which $C_i^{\text{LO}} < C_i^{\text{HI}}$, could trigger a mode switch.

8.1.2 Scheduling Policies for the Precise MC Task Model

8.1.2.1 EDF-VD

We have already discussed the details of EDF-VD In Subsection 7.4.1. In this section, we describe in detail the enhanced EDF-VD algorithm to compromise our precise energy-conserving model. Figure 8.1 represents a *modified* EDF-VD algorithm which determines if the task-set τ is schedulable. Then, the modified EDF-VD assigns virtual deadline $\widehat{T_i}$ for all HI-criticality tasks of the schedulable task-set.

For a dual-criticality task-set $\tau = \{\tau_1, \tau_2, ..., \tau_n\}$ to be scheduled by energy conserving preemptive processor with normal speed ρ and max speed 1:

- Scaling factor x is computed to determine virtual deadline of HI-criticality tasks:

$$x \leftarrow \frac{U^L_{\text{HI}}}{\rho - U^L_{\text{LO}}}$$

- If $U^L_{\text{LO}} + \frac{U^H_{\text{HI}}}{(1-x)} \leq 1$

    then virtual-deadline $\widehat{T_i} \leftarrow xT_i$ for every HI task τ_i;

  Else return failure.

Figure 8.1: Modified EDF-VD schedulability condition and scaling factor x

According to the algorithm in Figure 8.1, the scaling factor x is computed and $\widehat{T_i}$ values are assigned to all HI-criticality tasks as $\widehat{T_i} \leftarrow xT_i$. Note that $U^{\text{LO}}_{\text{LO}} + U^{\text{LO}}_{\text{HI}} \leq \rho$ is a necessary condition for schedulability under LO-criticality mode, $x \leq 1$ always holds for the proposed assignment. Contradictory to the traditional MC model, instead of discarding all LO-criticality tasks in HI-criticality behaviors, our model schedules both LO- and HI-criticality tasks with their given WCETs at processor speed $s \leftarrow 1$. Note that, we did not consider any additional overhead (that may be introduced) for changing the speed from ρ to the maximum speed (i.e., $s \leftarrow 1$). This assumption does not have a profound effect for two reasons. First, mode-switching is a rare event, and hence the changing of the speed. Second, we change the speed only once per mode switch.

8.1.2.2 Fluid Scheduling

We now present another approach to tackle the same problem (energy-aware precise scheduling of MC tasks), which is based on fluid scheduling framework. In fluid scheduling, all the tasks receive a fraction of the processor and have a constant execution rate from their release to the deadline. To be a feasible schedule, the summation of the assigned executing speeds of all tasks should not exceed the capacity (or the speed) of the processor. For each LO-criticality task, a minimum necessary execution speed of $\theta_i = u_i$ would be sufficient under both modes. While for a HI-criticality task, it would need a relatively larger speed under the normal mode (to create enough gap after the mode switch to handle the additional execution requirement), and even a larger speed after the mode switch. Such a relationship is demonstrated in Figure 8.2, where the blue character indicates LO-criticality task setting and the red character represents HI-criticality task settings.

In this section, we use some simplified notations for the per-mode utilization of the whole task set τ:

$$U^{\text{LO}} = \sum_i u_i^{\text{LO}}; \;\; U^{\text{HI}} = \sum_i u_i^{\text{HI}}.$$

Each task τ_i is assigned a execution speed θ_i in the HI-criticality mode and is assigned $\lambda \cdot \theta_i$ in the LO-criticality mode where $0 < \lambda \leq 1$. Defining λ and θ_i in the following manner can result in a schedule where all deadlines are guaranteed to meet in both the HI-criticality mode and the LO-criticality mode, provided a minimum speed λ is granted in the LO-criticality mode (and the full

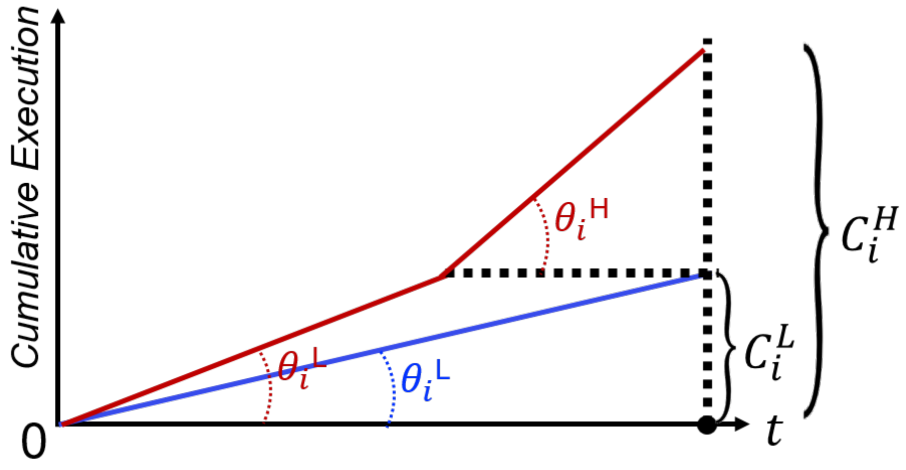


Figure 8.2: Relation between fluid execution speed and cumulative execution over time of a task under MCF framework.

speed, i.e., the unit-speed 1.0, of the processor that would be enabled in the HI-criticality mode).

Note that, we have $0 < \lambda \leq 1$ (see Equation (8.1)) because $U^{\text{LO}} > 0$ and $U^{\text{HI}} \leq 1$.

8.1.2.3 *A Generalized Fluid Scheduling Approach*

In this section, we focus on the *dual-rate fluid* scheduling, [1] where each task τ_i is assigned two constant executing rates in LO- and HI-modes, denoted by θ_i^{LO} and θ_i^{HI}, respectively. A set of rates assignment pairs $\{(\theta_i^{\text{LO}}, \theta_i^{\text{HI}})\}_{i=1}^n$ is *feasible* if and only if

$$\sum_i \theta_i^{\text{LO}} \leq \rho, \tag{8.3}$$

[1]The conventional fluid scheduling assumes a single constant rate for each task, whereas two rates, i.e., one rate change for each task, have been proposed and considered in the context of MC scheduling [78, 19]. Note that fluid scheduling with no restriction on the number of rate changes can be too general and pointless. For example, *any* actual schedule can be viewed as a fluid schedule where the rate for each task is switching between 0 and 1.0.

For a dual-criticality task-set $\tau = \{\tau_1, \tau_2, ..., \tau_n\}$ to be scheduled by energy conserving preemptive processor:

- A system-wide parameter λ and per-task parameters θ_i are computed as:

$$\lambda = \frac{U^L}{1 + U^L - U^H}. \tag{8.1}$$

$$\forall i, \quad \theta_i = \frac{u_i^L}{\lambda} + u_i^H - u_i^L \tag{8.2}$$

- If the energy-conserving speed $\rho \geq \lambda$

  then each task τ_i is to be executed at speed $\lambda \cdot \theta_i$ in the LO-criticality mode and at speed θ_i in the HI-criticality mode;

  Else return failure.

Figure 8.3: Modified MCF speed assignments and schedulability condition

$$\sum_i \theta_i^{\text{HI}} \leq 1.0, \tag{8.4}$$

where all deadline are guaranteed to meet in both LO- and HI-modes.

Note that, the algorithm proposed in Figure 8.3 is also based on fluid scheduling, and focused on rates assignments such that $\forall i, \theta_i^{\text{LO}}/\theta_i^{\text{HI}}$ are identical. However, the set of task systems that are schedulable under dual-rate fluid scheduling with *some* rates assignment—is more general and is a super set of the set of task systems that are schedulable by the algorithm in [26].

Now, we discuss how to form up the constraints on rates assignment set $\{(\theta_i^{\text{LO}}, \theta_i^{\text{HI}})\}_{i=1}^n$ in order to guarantee all deadlines to be met in both HI- and LO-modes. First, for each task τ_i, all its jobs that both are released and have a deadline in LO-mode must meet their deadlines if and only if

$$\frac{C_i^{\mathrm{LO}}}{\theta_i^{\mathrm{LO}}} \le T_i, \quad \forall i : 1 \le i \le n. \tag{8.5}$$

Second, for each task τ_i, all its jobs that both are released and have a deadline in HI-mode must meet their deadlines if and only if

$$\frac{C_i^{\mathrm{HI}}}{\theta_i^{\mathrm{HI}}} \le T_i, \quad \forall i : 1 \le i \le n. \tag{8.6}$$

Furthermore, the general idea of dual-rate fluid scheduling does not necessarily dictate the relationship between θ_i^{LO} and θ_i^{HI}. Nonetheless, Theorem 17 shows that we can restrict out attention to *non-decreasing* dual-rate fluid scheduling only, where it is required that

$$\theta_i^{\mathrm{LO}} \le \theta_i^{\mathrm{HI}}, \quad \forall i : 1 \le i \le n. \tag{8.7}$$

Theorem 17. *Any task system that is schedulable under dual-rate fluid scheduling must also be schedulable under* ***non-decreasing*** *dual-rate fluid scheduling.*

Proof. To see this theorem is true, we need to notice that if a task system is schedulable under some dual-rate fluid scheduling where $\theta_i^{\mathrm{LO}} > \theta_i^{\mathrm{HI}}$ for some i, then this system must still be schedulable when we assign $\theta_i^{\mathrm{LO}} \leftarrow \theta_i^{\mathrm{HI}}$. This is because the total rate constraints (8.3) and (8.4) cannot be violated by this assignment that reduces θ_i^{LO}, and Constraint (8.6) implies that a single constant execution rate of θ_i^{HI} in both LO- and HI-modes for task τ_i (which is the scenario for τ_i after the reducing) guarantees all its deadlines met regardless whether and where the mode switches, because $C_i^{\mathrm{LO}} \le C_i^{\mathrm{HI}}$. □

Therefore, under the **non-decreasing** dual-rate assumption, for each task τ_i, its job (if any) that is released in LO-mode but also executes in HI-mode must meet its deadline (which is in HI-mode) if and only if

$$\frac{C_i^{\text{LO}}}{\theta_i^{\text{LO}}} + \frac{C_i^{\text{HI}} - C_i^{\text{LO}}}{\theta_i^{\text{HI}}} \leq T_i, \quad \forall i : 1 \leq i \leq n. \tag{8.8}$$

This is sufficient because $C_i^{\text{LO}}/\theta_i^{\text{LO}}$ time units after its release is the latest time for the mode switch to be triggered and the mode switch is triggered earlier by any other job, the deadline must also be met by (8.7). This is necessary because any HI-task can be the one that triggers the mode switch and executes up to exact C_i^{HI} budget. We can generally claim $\forall i$ in (8.8) because if τ_i is a LO-task (i.e., $C_i^{\text{LO}} = C_i^{\text{HI}}$), then (8.8) reduces to (8.5).

Note that, each of the above equations is "if an only if" and all three possible situations of a job (entirely in LO or HI-mode, and across the mode switch time instant) have been exhausted. Therefore, Constraints (8.3)—(8.8) are a necessary and sufficient condition for the MC task system on the varying-speed processor to be schedulable by *any* two-rate fluid scheduling.

With Constraints (8.3)—(8.8) and the additional set of non-negative rate assignment constraints ($\theta_i^{\text{LO}} \geq 0, \forall i$), we have an optimization problem with linear and linear fractional inequality constraints, where θ_i^{LO} and θ_i^{HI} are variables and all others are problem input constants. Thus, in total, there are $\mathcal{O}(n)$ variables, $\mathcal{O}(n)$ linear constraints, and $\mathcal{O}(n)$ linear fractional constraints, where n is the number of tasks. Efficient numerical solvers (such as `fmincon` [56] or `CVX` [62] in Matlab) can be used to find a feasible solution.

Objective function. We have shown that the MC task system on the varying-speed processor to be schedulable under two-rate fluid scheduling if and only if a feasible solution exists for Constraints (8.3)—(8.8). Therefore, an objective function is not necessary for determining schedulability for a given degraded speed ρ. Thus, we can simply choose a trivial objective function "minimize 1" when applying the solver. On the other hand, if the degraded speed is not given as part of the system setting, then we can replace Constraint (8.3) by the following optimization objective:

$$\min \sum_i \theta_i^{\text{LO}}. \tag{8.9}$$

8.1.3 Evaluation

In this section, we report the performance of F2VD through experimental results, conducted on a randomly generated task-set. We also compare our algorithm with the approaches studied in [26]. To generate a random task-set, we use the workload generation model proposed by Guan et al. [63]. We describe the input specifications to generate the workload (used in this experiment) as follows:

- U_{bound}: the upper bound of the system utilization.
- $[T_{down}, T_{up}]$: the range of the minimum inter-arrival period of a task ,i.e., $0 \leq T_{down} \leq T_i \leq T_{up}$.
- $[U_{down}, U_{up}]$: the range of the utilization of a task. This value (let us denote it by u_i) is used to to obtain execution time of a task in the LO-mode, i.e., $\forall \tau_i \in \tau : c_i = u_i \times T_i$, where, $0 \leq U_{down} \leq u_i \leq U_{up} \leq 1$.
- $[Z_{down}, Z_{up}]$: the range of the ratio of HI and LO-criticality WCET, here $1 \leq Z_{down} \leq Z_{up}$.
- P: the probability that a task is a HI-task. Here, $0 \leq P \leq 1$

Finally, we compare our algorithm with the following baselines:

- ***EDF-VD*** [26] and ***MCF*** [26], respectively denoted by $\rho = X(EDF-VD)$ and $\rho = X(MCF)$, where $X = \rho$ values.

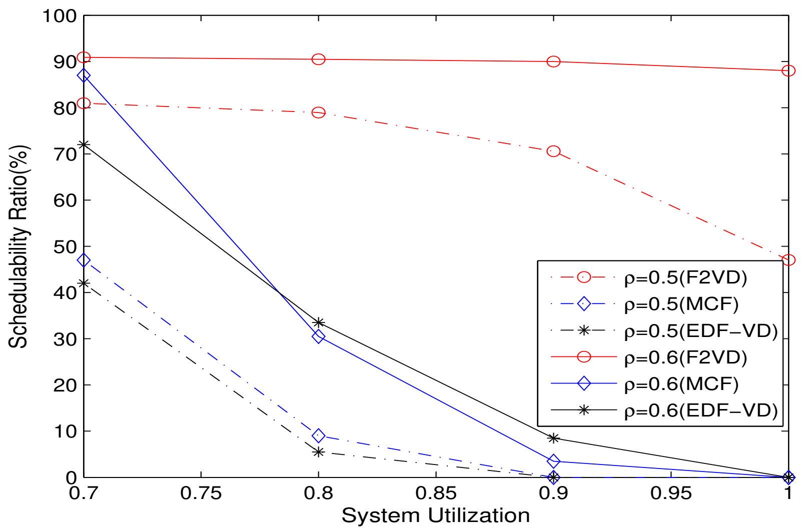


Figure 8.4: Comparison of schedulability ratio between F2VD, EDF-VD and the MCF.

In our experiment, we change the value of ρ (ranging from [0.5,0.6]) and the system utilization U_{bound} (ranging from [0.7,1.0]), and report the *schedulability ratio*, which is defined as the ratio of scheduled task-sets over the total number of task-sets. In Fig. 8.4, we show the comparison (w.r.t schedulability ratio) between our algorithm, EDF-VD, and the MCF for different utilization and ρ values. In this experimental settings, we consider the parameters as follow: $[U_{down}, U_{up}] = [0.02, 0.2]; [T_{down}, Tup] = [5, 50]; [Z_{down}, Z_{up}] = [1, 4]; P = 0.5$. Fig. 8.4 reports that our approach outperforms both the EDF-VD and MCF approaches by large margin. All these approaches follow the same trend, i.e., for any ρ values, schedulability ratio decreases when system utilization increases. While, for the same system utilization and ρ value, the schedulability ratio of our approach is almost twice as large compared to the other approaches. Similar to the Fig. 8.4, we report the performance comparison (w.r.t. schedulability ratio) for these three approaches in Fig. 8.5 with a different $[Z_{down}, Z_{up}]$ value. In this experiment$[Z_{down}, Z_{up}] = [1, 8]$, while other parameters are the same as Fig. 8.4. Compared to the previous evaluation setup (i.e., $[Z_{down}, Z_{up}] = 4$), we observe a similar improvement in the schedulability ratio for our approach,

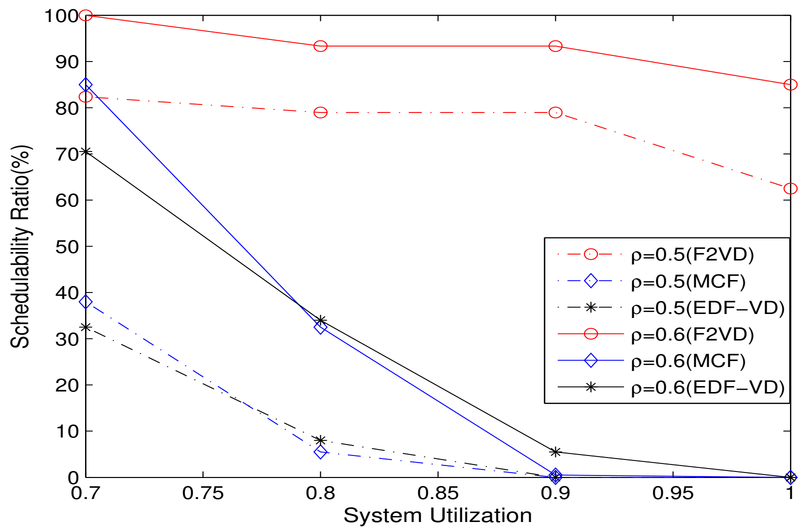


Figure 8.5: Comparison of schedulability ratio between F2VD, EDF-VD and the MCF.

i.e., our approach outperforms the other two approaches.

In Fig. 8.6, we report the time needed (with and without an objective function, refer to Sec. 8.1.2.3 for details) by F2VD to return the solution for a different size of task-set. We We set the value of ρ to 0.5 with other fixed parameters as in Fig. 8.4 and use the Matlab's CVX solver [62] to achieve a feasible solution. As expected, we observe a proportional relationship between the task-set size and the time needed (to return a feasible solution, if any) by the CVX solver. For a small task-set, we also see an improvement in the time when we use the solver without an objective function. Finally, these results do not demonstrate a significant variation in the time required to return a solution (with and without the objective function) with the changes in system utilization.

In Fig. 8.7, we report the *percentage of the feasible solution*, i.e., (the ratio of the number of the task sets for which F2VD returns a feasible solution over the total number of task sets), under different utilization and ρ values. From this figure, we observe that our algorithm finds a solution on average 70% cases. Also, we observe that the success percentage drops when the system utilization

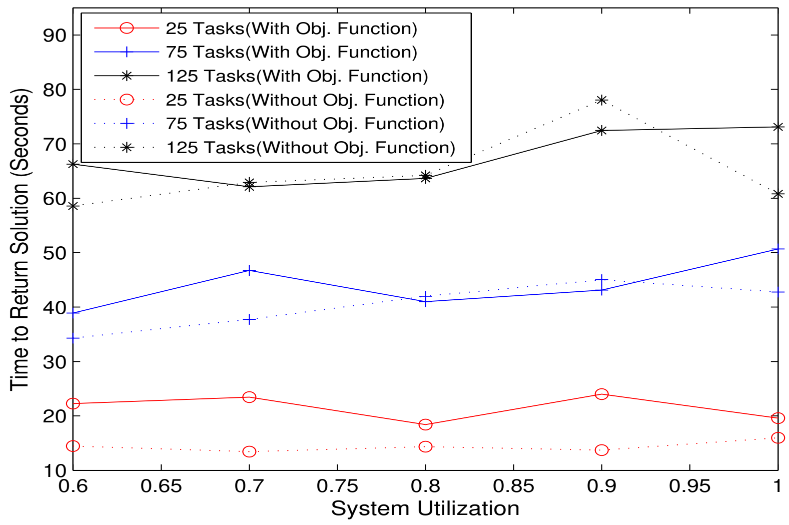


Figure 8.6: Time needed by F2VD to return the solution for a different size of task-set.

increases. This is because higher system utilization often leads to a system that is more difficult to schedule, which results in the optimization problem becoming harder or even infeasible to solve.

8.2 Probabilistic MC Task Model

We utilize the probabilistic execution profile of a task to find the optimistic WCETs for tasks. This optimization is performed via a novel probabilistic analysis (shown in [25]) of the execution time, coupled with a dedicated response time analysis (presented in this section). The response time analysis (RTA) guarantees the timing correctness of all tasks under both the pessimistic and optimistic assumptions. In Subsection 8.2.1, We present the modified MC task model to accommodate the probabilistic execution pattern of a task. Subsection 8.2.2 presents the RTA, and Subsection 8.2.3 presents the experimental results.

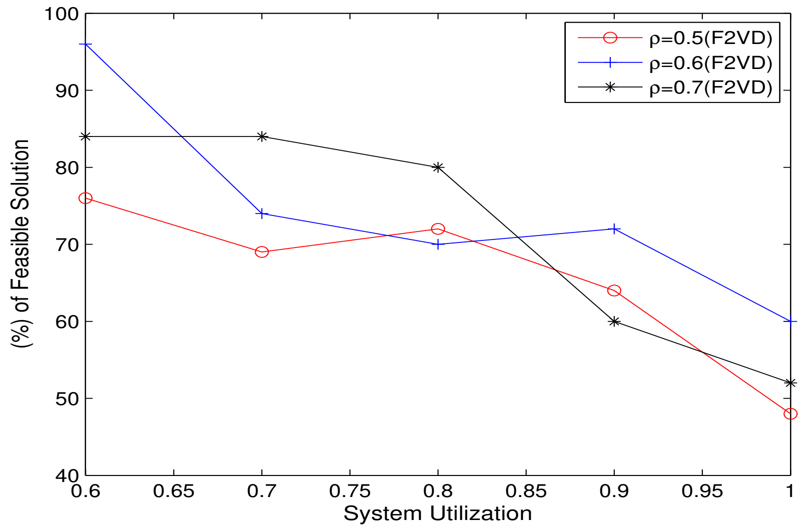


Figure 8.7: Percentage of solution returned by the F2VD algorithm under different utilization and ρ values.

8.2.1 System Model and Correctness Criteria

Now, we consider the probabilistic MC task model, where we represent τ_i by a 5-tuple parameter $\{T_i, C_i^{\text{LO}}, C_i^{\text{HI}}, pET_i, L_i\}$. Here, pET_i the probabilistic profile, for details refer to [25]. L_i is the criticality level, where $L_i \in \{\text{LO}, \text{HI}\}$, and all other notations carry the same meaning as described earlier in this section. Note that most traditional MC works assumed that the value of WCET in LO-criticality mode, i.e., C_i^{LO}, is considered to be provided as an input or empirically picked according to a defined percentage of the HI-criticality WCET. In this model, we compute the C_i^{LO} so that it minimizes the energy consumption [2]. There is a correlation between the probability system mode-switch and the minimum achievable speed in LO-criticality mode. Large C_i^{LO} would delay the activation of the HI-criticality mode, thus reducing the probability of this event to happen.

[2]Although the calculation under such a purpose would technically lead to execution thresholds that have nothing to do with 'worst-case', we nevertheless still follow the traditional MC work in the real-time and embedded systems community by calling them WCET under the LO-criticality mode [25]

However, it requires increasing the minimum speed in LO-criticality mode (which leads to more energy consumption during normal events) to guarantee the schedulability. Meanwhile, a small C_i^{LO} would decrease the minimum speed in LO-criticality mode, and hence more desired from the energy-saving perspective. However, a small C_i^{LO} increases the mode-switch probability and, if it happens too frequently, it may produce the opposite effect of increasing the energy consumption. In this work, we select C_i^{LO} considering a task's probabilistic profile, and in particular, by exploiting the inverse cumulative distribution function (ICDF): a value $p^{\text{LO}\rightarrow\text{HI}}$ that represents the probability per job to switch from LO-criticality to HI-criticality is selected according to the optimization problem and used to compute the WCET with the ICDF:

$$C_i^{\text{LO}} = F_i^{-1}(p^{\text{LO}\rightarrow\text{HI}})$$

Mode Switch Mechanism and Correctness Requirements. Most of the existing papers on MC systems adopt the system mode switch effect (refer to Section 8.1). In such a model, all the LO-criticality tasks are dropped when the mode-switch happens, i.e., a HI-criticality task overruns its C_i^{LO}. In this work, we adopted the precise scheduling policy where *each* task τ_i (including LO-criticality ones) is guaranteed execute under any condition. Similar to the previous sections, we the speed-change mechanism, where the systems start running at a reduced/degraded speed, which helps to achieve resource/Energy efficiency at LO-criticality mode. The system mode switch (i.e., LO to HI-criticality) causes an increase of the processor frequency to guarantee that all jobs are correctly scheduled. Let s_{LO} and s_{HI} denote the processor speed when the system stays in LO-criticality and HI-criticality mode, respectively [3]. Similar to the traditional MC setting, a system mode-switch takes place when a HI-criticality task overruns its C_i^{LO}. At the end of each hyper-

[3]The values of s_{LO}, s_{HI}, and C_i^{LO} depend on the schedulability of the task set and they impact the system energy consumption. Hence, we select these parameters in such a way so that the energy consumption is minimized. We utilize the probabilistic information, while guaranteeing the schedulability of the whole task-set according to the deterministic WCET.

period, or when the system is idle, whichever comes earlier, the system reverts to LO-criticality mode.

8.2.2 Response Time Analysis

In this section, we discuss the existing response time analysis (RTA) for a non-preemptive FP scheduler for both the non-MC and the MC tasks (Subsection 8.2.2.1). Then, in Subsection 8.2.2.2, we propose the RTA for our algorithm.

8.2.2.1 Existing RTA for Non-MC and MC Tasks

Two state-of-the-art works [8, 90] proposed an RTA of non-preemptive FP scheduling, but under settings that are different from this paper's focus. We will report the main results in the form of equations to make it easier to follow the subsequent RTA for our problem.

First, we describe some commonly used notations in most FP scheduling analysis work: Let $hep(i)$ (or $hp(i)$) denotes the set of tasks with higher or equal (or higher only) priority than τ_i. Similarly, $lep(i)$ (or $lp(i)$) denotes the set of tasks with lower or equal (or lower only) priority than τ_i.

Considering the non-preemptive FP scheduling for the non-MC tasks, Mohan et al. [90] calculated the Worst-Case Response Time (WCRT) R_i of task τ_i as follows:

$$R_i = B_i + C_i + \sum_{\tau_j \in hep(i)} N_j \times C_j$$

Here, B_i is the maximum duration that τ_i can be blocked by lower priority tasks and N_j is the total number of interfering jobs of $\tau_j \in hep(i)$, defined as:

$$B_i = \max_{\tau_k \in lp(i)} C_k - 1, \text{ and } N_j = \left\lfloor \frac{R_i - C_i}{T_j} + 1 \right\rfloor \tag{8.10}$$

Before moving further into the details, let us introduce the same notations but for the MC case: $lp\chi(i)$ (or $hp\chi(i)$, $lep\chi(i)$, $hep\chi(i)$, respectively) denote the set of χ-criticality tasks with lower (or higher only, lower or equal, lower or equal) priority than τ_i, where $\chi \in \{\text{LO}, \text{HI}\}$ and other notations carry their usual meaning. We also use B_i^χ, C_i^χ and N_j^χ to denote the maximum blocking time, execution time, and the total number of interfering jobs ($\tau_j \in hep(i)$) of task τ_i at χ-criticality mode, respectively.

Baek et al. [8] extended the analysis above to MC task model by considering the adaptive MC (AMC) scheme [18], and derived WCRTs for the following three cases separately:

Case 1. WCRT at LO-criticality mode for any task τ_i is calculated as:

$$R_i^{\text{LO}} = B_i^{\text{LO}} + C_i^{\text{LO}} + \sum_{\tau_j \in hep(i)} N_j^{\text{LO}} \times C_j^{\text{LO}}, \text{ where} \tag{8.11}$$

$$B_i^{\text{LO}} = \max_{\tau_k \in lp(i)} C_k^{\text{LO}} - 1, N_j^{\text{LO}} = \left\lfloor \frac{R_i^{\text{LO}} - C_i^{\text{LO}}}{T_j} + 1 \right\rfloor$$

Case 2. WCRT at HI-criticality mode for any HI-criticality task τ_i is calculated as:

$$R_i^{\text{HI}} = B_i^{\text{HI}} + C_i^{\text{HI}} + \sum_{\tau_j \in hep\text{HI}(i)} N_j^{\text{HI}} \times C_j^{\text{HI}}, \text{ where} \tag{8.12}$$

$$B_i^{\text{HI}} = \max_{\tau_k \in lp\text{HI}(i)} C_k^{\text{HI}} - 1, N_j^{\text{HI}} = \left\lfloor \frac{R_i^{\text{HI}} - C_i^{\text{HI}}}{T_j} + 1 \right\rfloor$$

Case 3. WCRT during mode switch of a job released by task τ_i (at LO-criticality mode but finished at HI-criticality mode) is calculated as:

$$R_i^{TR} = B_i^{TR} + C_i^{\text{HI}} + \sum_{\tau_j \in hep\text{LO}(i)} N_j^{\text{LO}} \times C_j^{\text{LO}} + \sum_{\tau_j \in hep\text{HI}(i)} N_j^{\text{HI}} \times C_j^{\text{HI}}, \text{ where} \tag{8.13}$$

$$B_i^{TR} = \max\left(\max_{\tau_k \in lp\text{LO}(i)} C_k^{\text{LO}}, \max_{\tau_k \in lp\text{HI}(i)} C_k^{\text{HI}}\right) - 1$$

$$N_j^{\text{LO}} = \left\lfloor \frac{R_i^{\text{LO}} - C_i^{\text{LO}}}{T_j} + 1 \right\rfloor$$

$$N_j^{\text{HI}} = \left\lfloor \frac{R_i^{TR} - C_i^{\text{HI}}}{T_j} + 1 \right\rfloor$$

For any HI-criticality task $\tau_i \in \tau_{\text{HI}}$, regardless of a mode-switch, WCRT is upper bounded by R_i^{TR}. Hence, under the AMC scheme, the following conditions determine the schedulability of a task-set $\overline{\tau}$: (i) $\forall_{\tau_i \in \overline{\tau}_{\text{LO}}}, R_i^{\text{LO}} \leq D_i$, and (ii) $\forall_{\tau_i \in \overline{\tau}_{\text{HI}}}, R_i^{TR} \leq D_i$.

8.2.2.2 RTA of Our Algorithm

In this subsection, we describe the RTA for our scheduling algorithm, which considers the following assumption:

Assumption: We assume that the WCET and the processor speed has a linear relationship [26], i.e., if τ_i starts executing on a processor (with C_i^{LO}) with a degraded speed s_{LO}, it will take $C_i^{\text{LO}}/s_{\text{LO}}$ time units to finish execution. Recall that under the LO-criticality mode, all the LO- and HI-criticality tasks execute at an energy-conserving speed, i.e., s_{LO}. After a LO- to HI-criticality mode switch, all the LO- and HI-criticality tasks execute at the maximum processor speed s_{HI}.

Deriving R_i^{LO}. Based on such assumption and the RTA analysis in Equation (8.11), we calculate

R_i^{LO} for all the LO- and HI-criticality tasks as follows:

$$R_i^{\text{LO}} = B_i^{\text{LO}} + \frac{C_i^{\text{LO}}}{s_{\text{LO}}} + \sum_{\tau_j \in hep(i)} N_j^{\text{LO}} \times \frac{C_j^{\text{LO}}}{s_{\text{LO}}} \tag{8.14}$$

with B_i^{LO} and N_j^{LO} computed as:

$$B_i^{\text{LO}} = \max_{\tau_k \in lp(i)} \left(\frac{C_k^{\text{LO}}}{s_{\text{LO}}} \right) - 1$$

$$N_j^{\text{LO}} = \left\lfloor \frac{R_i^{\text{LO}} - \frac{C_i^{\text{LO}}}{s_{\text{LO}}}}{T_j} + 1 \right\rfloor \tag{8.15}$$

Deriving R_i^{HI}. Now, we derive R_i^{HI} thanks to Equation (8.12). Recall that, we do not drop any LO-criticality tasks after a mode switch. Hence, $hep\text{HI}(i)$ in Equation (8.12) needs to be replaced by $hep(i)$:

$$R_i^{\text{HI}} = B_i^{\text{HI}} + \frac{C_i^{\text{HI}}}{s_{\text{HI}}} + \sum_{\tau_j \in hep(i)} N_j^{\text{HI}} \times \frac{C_j^{\text{HI}}}{s_{\text{HI}}}, \tag{8.16}$$

We derive the blocking time, B_i^{HI}, considering two cases.

Case 1. The blocking task, τ_k, initiates the mode-switch. Clearly, $\tau_k \in \overline{\tau}_{\text{HI}}$, and the blocking time becomes:

$$B_i^{(1)} = \max_{\tau_k \in lp\text{HI}(i)} \left(\frac{c_k^{\text{LO}}}{s_{\text{LO}}} + \frac{c_k^{\text{HI}} - c_k^{\text{LO}}}{s_{\text{HI}}} \right) - 1$$

Case 2. Task τ_k releases and completes execution at HI-criticality mode, and the blocking time becomes:

$$B_i^{(2)} = \max_{\tau_k \in lp(i)} \left(\frac{C_k^{\text{HI}}}{s_{\text{HI}}} \right) - 1$$

We calculate B_i^{HI} and N_j^{HI} as:

$$B_i^{\text{HI}} = \max(B_i^{(1)}, B_i^{(2)})$$

$$N_j^{\text{HI}} = \left\lfloor \frac{R_i^{\text{HI}} - \frac{C_i^{\text{HI}}}{s_{\text{HI}}}}{T_j} + 1 \right\rfloor \tag{8.17}$$

Deriving R_i^{TR}. Now, we calculate R_i^{TR}, i.e. the WCRT of a HI-criticality task τ_i that faces the mode-switch. Note that, R_i^{TR} cannot be deduced from the calculation of WCRT during the stable (LO- or HI-criticality) modes [119]. We instead calculate the WCRT of a HI-criticality task, τ_i, by summing the blocking time, WCET of τ_i and the worst-case interference from other tasks with a higher priority than τ_i. Unlike [8], we do not drop the LO-criticality tasks after a mode-switch. Hence, the LO-criticality tasks can contribute to the interference (even after a mode-switch). We derive the WCRT of a HI-criticality task τ_i that faces mode-switch considering two cases, (1) τ_i initiates the mode-switch, and (2) any other (HI-criticality) task initiates the mode-switch. For both these cases, we assume that the mode-switch happens at time t^*.

Case 1. In this case, τ_i initiates the mode-switch and once τ_i starts execution, it can execute till c_i^{HI}, thanks to the non-preemptivity. According to the assumption, τ_i starts execution at LO-criticality mode. So, a task $\tau_k \in lp(i)$ that blocks τ_i must start and finish execution at LO-criticality mode. We calculate the maximum blocking time $B_i^{(1)}$ as follows:

$$B_i^{(1)} = \max_{\tau_k \in lp(i)} \left(\frac{C_k^{\text{LO}}}{s_{\text{LO}}} \right) - 1$$

Before the mode-switch, τ_i will execute up to c_i^{LO} at speed s_{LO}, and after that it will execute at s_{HI}.

Considering this scenario, we calculate the task execution time $C_i^{(1)}$:

$$C_i^{(1)} = \frac{C_i^{\text{LO}}}{s_{\text{LO}}} + \frac{C_i^{\text{HI}} - C_i^{\text{LO}}}{s_{\text{HI}}} \tag{8.18}$$

Recall that, τ_i initiates the mode-switch. Hence, τ_i can be interfered by τ_j (where $\tau_j \in hep(i)$) at LO-criticality mode only. We calculate the maximum interference $I_i^{(1)}$ as follows:

$$I_i^{(1)} = \sum_{\tau_j \in hep(i)} \left\lfloor \frac{R_i^{\text{LO}} - \frac{C_i^{\text{LO}}}{s_{\text{LO}}}}{T_j} + 1 \right\rfloor \times \frac{C_j^{\text{LO}}}{s_{\text{LO}}}$$

Finally, we calculate the response time $R_i^{(1)}$:

$$R_i^{(1)} = B_i^{(1)} + C_i^{(1)} + I_i^{(1)} \tag{8.19}$$

Case 2. In this case, any HI-criticality task (other than τ_i) initiates the mode-switch. We calculate the maximum blocking time considering the following three sub-cases:

Sub-case 2.1. The blocking task τ_k is released at LO*-criticality mode and does not initiate a mode-switch.* Recall that:

(i) τ_i is also released at LO-criticality mode, and

(ii) a HI-criticality task τ_j initiates the mode-switch, where $\tau_j \in \overline{\tau} \setminus \{\tau_i, \tau_k\}$.

From the first assumption and the non-preemptive scheduling policy, it is not possible for τ_k to start execution at LO-criticality mode, while finish at at HI-criticality mode. So, τ_k must *start and finish execution at* LO*-criticality mode*, Finally, we calculate the maximum blocking time as follows:

$$B_i^{(2)} = \max_{\tau_k \in lp(i)} \left(\frac{C_k^{\text{LO}}}{s_{\text{LO}}} \right) - 1$$

Sub-case 2.2. The blocking task τ_k is released at LO*-criticality mode and initiates a mode-switch.*

Clearly, $\tau_k \in \overline{\tau}_{\text{HI}}$. By the same reasoning as that for Case 1 (Equation (8.18)), we calculate the maximum blocking time as follows:

$$B_i^{(3)} = \max_{\tau_k \in lp\text{HI}(i)} \left(\frac{c_k^{\text{LO}}}{s_{\text{LO}}} + \frac{c_k^{\text{HI}} - c_k^{\text{LO}}}{s_{\text{HI}}} \right) - 1$$

Sub-case 2.3. The blocking task τ_k is released at HI*-criticality mode*. This case can be discarded, as τ_k will never block τ_i. This is because, τ_i is released at LO-criticality mode, and will start execution before $\tau_k \in lp(i)$.

To calculate the task execution time, recall that, τ_i is released at LO-criticality mode and does not initiate the mode-switch. Hence, τ_i can start execution only at the HI-criticality mode. Otherwise, τ_i itself invokes a mode-switch or must finish execution before the mode-switch (due to the non-preemptivity), which contradicts our assumption. Hence, the task execution time is:

$$c_i^{(2)} = c_i^{\text{HI}}$$

As τ_i starts execution only at the HI-criticality mode, τ_i can be interfered by $\tau_j \in hep(i)$ at both LO and HI-criticality modes (Sub-case 2.1) or only at HI-criticality mode (Sub-case 2.2). Let, the mode-switch takes place at time t^*. The *upper bound* of the interference is then:

$$\begin{aligned} I_i^{(2)} = \sum_{\tau_j \in hep(i)} \Bigg[& \left(\left\lfloor \frac{t^*}{T_j} + 1 \right\rfloor \times \frac{C_j^{\text{LO}}}{s_{\text{LO}}} \right) \\ & + \left(\left\lfloor \frac{R_i^{TR} - t^* - \frac{C_i^{\text{HI}}}{s_{\text{HI}}}}{T_j} + 1 \right\rfloor \times \frac{C_j^{\text{HI}}}{s_{\text{HI}}} \right) \Bigg] \end{aligned}$$

and the response time $R_i^{(2)}$:

$$R_i^{(2)} = \max(B_i^{(2)}, B_i^{(3)}) + C_i^{(2)} + I_i^{(2)} \tag{8.20}$$

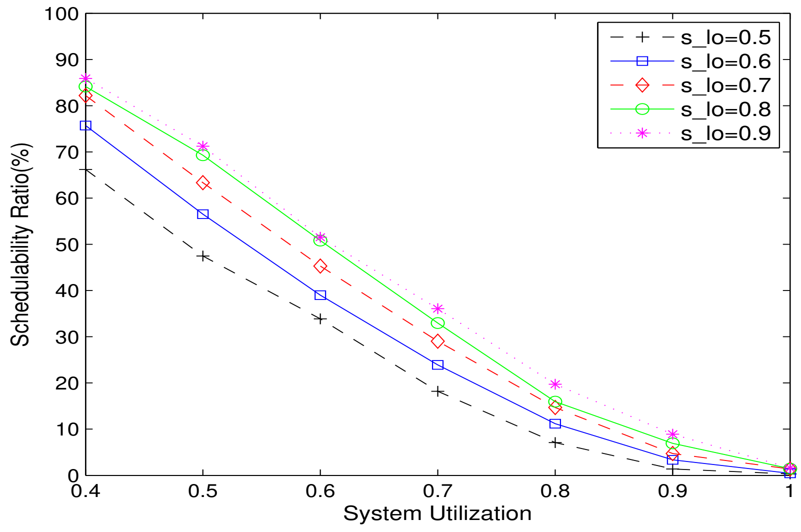


Figure 8.8: Schedulability ratio based on the RTA with a different s_{LO} values. In this experiment, $[Z_d, Z_u] = [1, 4]$.

Considering all the scenarios (i.e., Case 1, Sub-case 2.1, 2.2 and 2.3) described above, we calculate R_i^{TR} as:

$$R_i^{TR} = \max(R_i^{(1)}, R_i^{(2)}) \tag{8.21}$$

Finally, we conclude that, a task-set $\overline{\tau}$ is schedulable by our algorithm if it satisfies the following condition:

- For each LO-criticality task $\tau_i \in \overline{\tau},\ max(R_i^{\text{LO}}, R_i^{\text{HI}}) \leq D_i$, and
- For each HI-criticality task $\tau_i \in \tau_{\text{HI}}$, $max(R_i^{\text{LO}}, R_i^{\text{HI}}, R_i^{TR}) \leq D_i$; where, $R_i^{\text{LO}},\ R_i^{\text{HI}}$ and R_i^{TR} are calculated using Equation (8.14), Equation (8.16) and Equation (8.21)

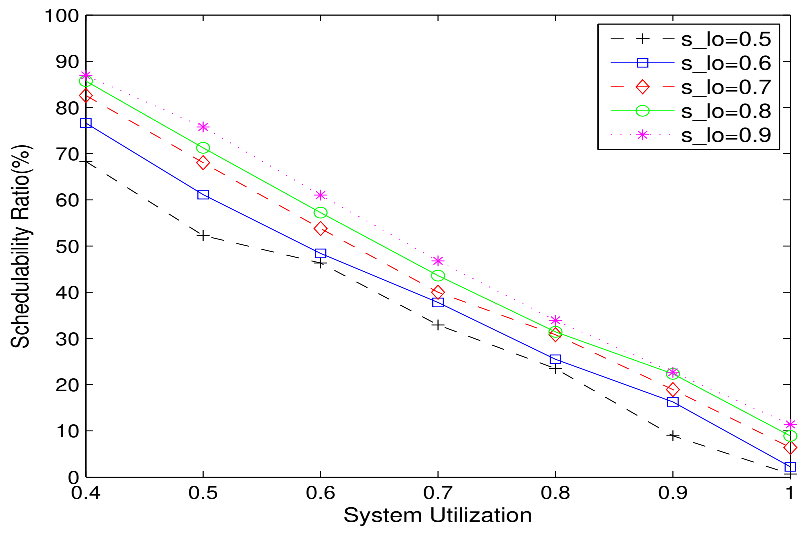


Figure 8.9: Schedulability ratio based on the RTA with a different s_{LO} values. In this experiment, $[Z_d, Z_u] = [1, 8]$.

8.2.3 Evaluation

In this section, we report the evaluation result of our algorithm. We report the *schedulability ratio*, i.e., the ratio of scheduled task-sets over the total number of task-sets, of our algorithm for fixed values of s_{LO}. To simplify the experimental evaluation, without loosing generality, we considered $s_{\text{HI}} = 1$, i.e. the maximum achievable speed in HI-criticality mode is set at the maximum processor speed. We conduct the experiments on a randomly generated task-set, and use the workload generation model proposed in [63]. We use the following input specifications to generate the workload:

- U_{bound}: the upper bound of the system utilization.
- $[T_d, T_u]$: the range of the minimum inter-arrival period of a task ,i.e., $0 < T_d \leq T_i \leq T_u$.
- $[U_d, U_u]$: the range of the utilization u_i (of τ_i). We use u_i to obtain execution time of τ_i in the LO-criticality mode, i.e., $\forall_{\tau_i \in \tau} : C_i^{\text{LO}} = u_i \times T_i$, where, $0 < U_d \leq u_i \leq U_u \leq 1$.

- $[Z_d, Z_u]$: the range of the ratio of HI and LO-criticality WCET, here $1 \leq Z_d \leq Z_u$.
- P: the probability of being a HI-criticality task.

In these experiments, we computed the schedulability ratio of our scheduling algorithm for different values of s_{LO} (ranging from [0.5,0.9]) and U_{bound} (ranging from [0.4,1.0]). Figure 8.8 reports that the schedulability ratio (for any s_{LO} value) with the following fixed parameters: $[U_d, U_u] = [0.02, 0.2]$;$[T_d, T_u] = [5, 50]$;$[Z_d, Z_u] = [1, 4]$; $P = 0.5$. It shows that the schedulability ratio decreases with the increase in system utilization, which matches with our RTA. Most of all task-sets are schedulable when U_{bound} is lower than 0.5, while most of the task-sets become not schedulable when utilization approaches 1. In this experiment, the selection of s_{LO} does not depend on the task-set itself but it has been fixed at different levels. This reduces the schedulability ratio, but it is implicitly solved by the inner optimization problem. We follow the experimental settings in Figure 8.9, except we use $[Z_d, Z_u] = [1, 8]$, and report the schedulability ratio. We observe the similar trend as shown in Figure 8.9.

8.3 Conclusion

The traditional MC task model provides no service guarantee to the LO-criticality tasks in HI-criticality mode. Some recent efforts proposed the precise MC task scheduling policy, where the scheduler offers a full or partial-service guarantee to the LO-criticality tasks after a system mode-switch. However, these studies rely on the pessimistic assumption that all the tasks execute up to their WCET at their respective criticality levels. A recent work studied the integration of probabilistic-based prediction strategy (of the task execution time) and the DVFS scheme to the precise scheduling of MC tasks. The probabilistic-based prediction strategy aims to minimize the energy consumption in an MC platform, while the requirement for the string timing guarantee is still in place. Hence, we propose the response time analysis of our algorithm under a fixed priority non-preemptive scheduler. The response time analysis is dedicated to guaranteeing the worst-case timing correctness for all tasks under any execution condition. We also evaluate our algorithm via simulation on randomly generated workloads and report energy-saving up to 46% with respect to the pessimistic choice of LO-criticality WCET commonly made by previous works.

CHAPTER 9: CONCLUSION

9.1 Summary of Results

Today's society observes the rapid growth of computation-intensive real-time embedded system applications with stringent timing requirements. The use of embedded system applications are sparked by billions of devices in all aspects of human life. Energy consumed by these billions of devices (i.e., mobile, laptop, tablet, cars, robot) is significant [108]. These embedded system applications often rely on unreliable energy sources/harvesters, such as batteries. Recharging the source/harvesters may not be possible during a mission, and hence it is crucial to consider the energy-aware design of the embedded system applications. Energy efficiency is also a prime requirement as it could reduce the power bills and increase battery life.

Embedded systems applications that demand strict timing guarantees, energy efficiency, and high performance, are moving towards multi-core platforms. The parallel task model, i.e., a task that can be executed on multiple cores simultaneously, can exploit the computational capability offered by the multi-core platform. Such a task model promotes a fair workload-processor distribution which leads to energy efficiency. Motivated by these above discussions, this thesis focuses on developing the theoretical foundation of an energy-aware scheduling algorithm for real-time parallel tasks.

First, considering the sporadic DAG task model (a widely accepted real-time task model to represent the intra-task parallelism), we propose an energy-efficient task scheduling approach in a multiprocessor platform. We propose an energy sub-optimal federated scheduling algorithm for the DAG task. We also present a greedy intra-task processor merging technique to reduce the leakage power consumption further. We propose a method to improve the energy-saving of the processor intra-merge technique by allowing multiple processors to merge at a single one. Finally, we also present the inter-task processor technique. We evaluate the effectiveness of the proposed approach both theoretically via approximation ratio bounds and experimentally through a simulation study.

Second, our analysis as mentioned above assumes the per-core DVFS technique (i.e., each core can tune its speed independently). We extend our solution to adapt the cluster-based homogeneous multi-core platform (processors in the same island execute at the same speed), which seems a promising platform to balance energy efficiency and hardware cost. We introduce a new concept of speed-profile that models per-task and per-cluster energy-consumption variations during run-time to minimize the expected long-term energy consumption. We have shown the effectiveness of our approach while handling the difficulties introduced by the cluster-based platform. We also have extended our approach to adapt the discrete processor frequency scheme and the platform heterogeneity.

Third, integrating the notion of MC with the parallel task models stems from many current trends. For example, nowadays, several safety-critical and non-safety-critical (i.e., mission-critical) tasks are embedded into a common computational platform. Hence, the demand for integrating functionality with different criticality levels (into the same platform) is increasing. Towards this goal, we combine the MC context into the gang task model, another well-known parallel task model. We propose a new technique GEDF-VD, which integrates the GEDF and EDF-VD scheduling policy. We also show the correctness of GEDF-VD at different systems criticality levels. Finally, we derive the first speedup bound for GEDF schedulability of non-MC gang tasks and use this analysis as a foundation to derive the bound for GEDF-VD of MC gang tasks.

Finally, a common non-functional requirement of embedded systems is to improve the average-case energy consumption while ensuring temporal correctness. Motivated by this requirement, we present the response time analysis of the MC task so that the probabilistic technique (proposed in [25]) minimizes the average energy consumption, will guarantee the worst-case timing correctness for all tasks under any execution condition.

9.2 Future Direction

Energy-aware scheduling (for both the MC and non-MC task model) will remain attractive in the embedded system research because of its vast applicability. This thesis has fulfilled several fundamental requirements in energy-aware real-time scheduling; still, several directions are yet to be explored. In this section, we point out some related and relevant future research areas.

In this work, we have restricted our attention mainly to the CPU power consumption. Although the CPU power consumption is one of the major contributors to the overall system power consumption, several other factors, e.g., cache misses, context switches, I/O usage, impact the overall power consumption [66, 24]. In the future, we plan to consider other components that may affect the total power consumption.

Despite the popularity of the DAG task model in the real-time community, it is not free of shortcomings [2]. one of them, the internal structure of the code, could be very complex. A recent effort [2] has been made to propose an alternative model that does not require comprehensive information regarding the internal structure (each node and their dependencies) of the DAG task and can be represented using only two parameters; *work* and *span*. It will be a promising direction to extend our analysis for the work-span DAG task model.

APPENDIX : PERMISSION TO REUSE PUBLISHED MATERIAL

ACM Author Gateway

Author Resources

Home > **Author Resources** > **Author Rights & Responsibilities**

ACM Author Rights

ACM exists to support the needs of the computing community. For over sixty years ACM has developed publications and publication policies to maximize the visibility, impact, and reach of the research it publishes to a global community of researchers, educators, students, and practitioners. ACM has achieved its high impact, high quality, widely-read portfolio of publications with:

- Affordably priced publications
- Liberal Author rights policies
- Wide-spread, perpetual access to ACM publications via a leading-edge technology platform
- Sustainability of the good work of ACM that benefits the profession

Choose

Authors have the option to choose the level of rights management they prefer. ACM offers three different options for authors to manage the publication rights to their work.

- Authors who want ACM to manage the rights and permissions associated with their work, which includes defending against improper use by third parties, can use ACM's traditional copyright transfer agreement.
- Authors who prefer to retain copyright of their work can sign an exclusive licensing agreement, which gives ACM the right but not the obligation to defend the work against improper use by third parties.
- Authors who wish to retain all rights to their work can choose ACM's author-pays option, which allows for perpetual open access through the ACM Digital Library. Authors choosing the author-pays option can give ACM non-exclusive permission to publish, sign ACM's exclusive licensing agreement or sign ACM's traditional copyright transfer agreement. Those choosing to grant ACM a non-exclusive permission to publish may also choose to display a Creative Commons License on their works.

- Contributing complete papers to any edited collection of reprints for which the author is notthe editor, requires permission and usually a republication fee.
- Authors can include partial or complete papers of their own (and no fee is expected) in a dissertation as long as citations and DOI pointers to the Versions of Record in the ACM Digital Library are included. Authors can use any portion of their own work in presentations and in the classroom (and no fee is expected).
- Commercially produced course-packs that are sold to students require permission and possibly a fee.

Create

ACM's copyright and publishing license include the right to make Derivative Works or new versions. For example, translations are "Derivative Works." By copyright or license, ACM may have its publications translated. However, ACM Authors continue to hold perpetual rights to revise their own works without seeking permission from ACM.

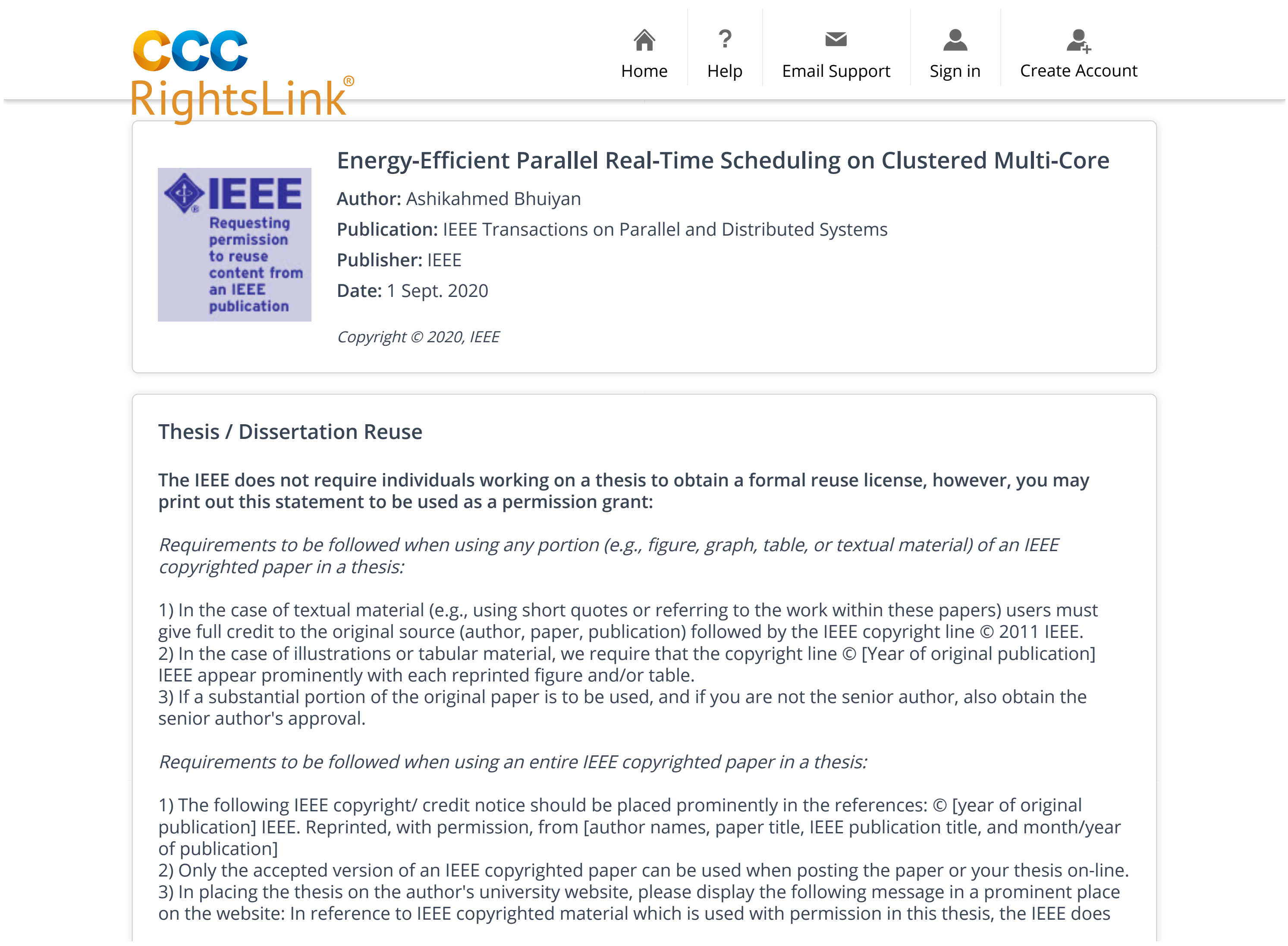
Minor Revisions and Updates to works already published in the ACM Digital Library are welcomed with the approval of the appropriate Editor-in-Chief or Program Chair.

- If the revision is minor, i.e., less than 25% of new substantive material, then the work should still have ACM's publishing notice, DOI pointer to the Definitive Version, and be labeled a "Minor Revision of"
- If the revision is major, i.e., 25% or more of new substantive material, then ACM considers this a new work in which the author retains full copyright ownership (despite ACM's copyright or license in the original published article) and the author need only cite the work from which this new one is derived.

Retain

Authors retain all perpetual rights laid out in the ACM Author Rights and Publishing Policy, including, but not limited to:

- Sole ownership and control of third-party permissions to use for artistic images intended for exploitation in other contexts
- All patent and moral rights
- Ownership and control of third-party permissions to use of software published by ACM


CCC RightsLink®
Home
Help
Email Support
Sign in
Create Account
IEEE
Requesting permission to reuse content from an IEEE publication

Energy-Efficient Parallel Real-Time Scheduling on Clustered Multi-Core

Author: Ashikahmed Bhuiyan

Publication: IEEE Transactions on Parallel and Distributed Systems

Publisher: IEEE

Date: 1 Sept. 2020

Copyright © 2020, IEEE

Thesis / Dissertation Reuse

The IEEE does not require individuals working on a thesis to obtain a formal reuse license, however, you may print out this statement to be used as a permission grant:

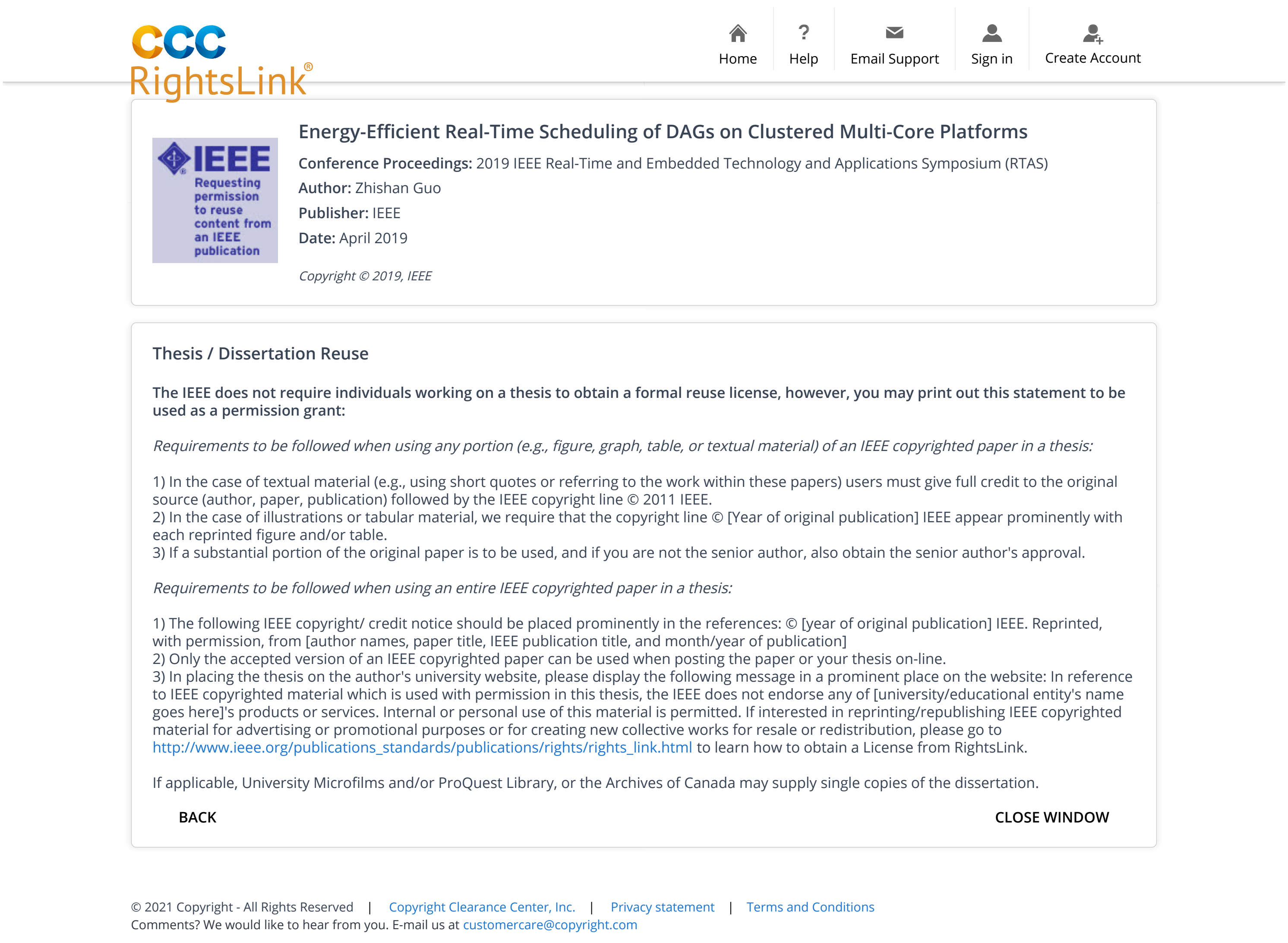
Requirements to be followed when using any portion (e.g., figure, graph, table, or textual material) of an IEEE copyrighted paper in a thesis:

1) In the case of textual material (e.g., using short quotes or referring to the work within these papers) users must give full credit to the original source (author, paper, publication) followed by the IEEE copyright line © 2011 IEEE.
2) In the case of illustrations or tabular material, we require that the copyright line © [Year of original publication] IEEE appear prominently with each reprinted figure and/or table.
3) If a substantial portion of the original paper is to be used, and if you are not the senior author, also obtain the senior author's approval.

Requirements to be followed when using an entire IEEE copyrighted paper in a thesis:

1) The following IEEE copyright/ credit notice should be placed prominently in the references: © [year of original publication] IEEE. Reprinted, with permission, from [author names, paper title, IEEE publication title, and month/year of publication]
2) Only the accepted version of an IEEE copyrighted paper can be used when posting the paper or your thesis on-line.
3) In placing the thesis on the author's university website, please display the following message in a prominent place on the website: In reference to IEEE copyrighted material which is used with permission in this thesis, the IEEE does

BACK **CLOSE WINDOW**



Home | Help | Email Support | Sign in | Create Account

Energy-Efficient Real-Time Scheduling of DAGs on Clustered Multi-Core Platforms

Conference Proceedings: 2019 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)
Author: Zhishan Guo
Publisher: IEEE
Date: April 2019

Copyright © 2019, IEEE

Thesis / Dissertation Reuse

The IEEE does not require individuals working on a thesis to obtain a formal reuse license, however, you may print out this statement to be used as a permission grant:

Requirements to be followed when using any portion (e.g., figure, graph, table, or textual material) of an IEEE copyrighted paper in a thesis:

1) In the case of textual material (e.g., using short quotes or referring to the work within these papers) users must give full credit to the original source (author, paper, publication) followed by the IEEE copyright line © 2011 IEEE.
2) In the case of illustrations or tabular material, we require that the copyright line © [Year of original publication] IEEE appear prominently with each reprinted figure and/or table.
3) If a substantial portion of the original paper is to be used, and if you are not the senior author, also obtain the senior author's approval.

Requirements to be followed when using an entire IEEE copyrighted paper in a thesis:

1) The following IEEE copyright/ credit notice should be placed prominently in the references: © [year of original publication] IEEE. Reprinted, with permission, from [author names, paper title, IEEE publication title, and month/year of publication]
2) Only the accepted version of an IEEE copyrighted paper can be used when posting the paper or your thesis on-line.
3) In placing the thesis on the author's university website, please display the following message in a prominent place on the website: In reference to IEEE copyrighted material which is used with permission in this thesis, the IEEE does not endorse any of [university/educational entity's name goes here]'s products or services. Internal or personal use of this material is permitted. If interested in reprinting/republishing IEEE copyrighted material for advertising or promotional purposes or for creating new collective works for resale or redistribution, please go to http://www.ieee.org/publications_standards/publications/rights/rights_link.html to learn how to obtain a License from RightsLink.

If applicable, University Microfilms and/or ProQuest Library, or the Archives of Canada may supply single copies of the dissertation.

BACK CLOSE WINDOW

Licence to Publish

SPRINGER NATURE

Journal Name:	Real-Time Systems	(the ‘Journal’)
Manuscript Number:	TIME-D-20-00016R1	
Proposed Title of Article:	Mixed-criticality real-time scheduling of gang task systems	
Author(s) [Please list all named Authors]:	Ashik Ahmed Bhuiyan, Kecheng Yang, Samsil Arefin, Abusayeed Saifullah, Nan Guan, Zhishan Guo	(the ‘Author’)
Corresponding Author Name:	Ashik Ahmed Bhuiyan	

1 Publication

Springer Science+Business Media LLC (the ‘Licensee’) will consider publishing this article, including any supplementary information and graphic elements therein (e.g. illustrations, charts, moving images) (the 'Article').
Headings are for convenience only.

2 Grant of Rights

In consideration of the Licensee evaluating the Article for publication, the Author grants the Licensee the exclusive (except as set out in clauses 3, 4 and 5a) iv) and sub-licensable right, unlimited in time and territory, to copy-edit, reproduce, publish, distribute, transmit, make available and store the Article, including abstracts thereof, in all forms of media of expression now known or developed in the future, including pre- and reprints, translations, photographic reproductions and extensions.

Furthermore, to enable additional publishing services, such as promotion of the Article, the Author grants the Licensee the right to use the Article (including the use of any graphic elements on a stand-alone basis) in whole or in part in electronic form, such as for display in databases or data networks (e.g. the Internet), or for print or download to stationary or portable devices. This includes interactive and multimedia use as well as posting the Article in full or in part or its abstract on social media, and the right to alter the Article to the extent necessary for such use. The Licensee may also let third parties share the Article in full or in part or its abstract on social media and may in this context sub-license the Article and its abstract to social media users. Author grants to Licensee the right to re-license Article metadata without restriction (including but not limited to author name, title, abstract, citation, references, keywords and any additional information as determined by Licensee).

If the Article is rejected by the Licensee and not published, all rights under this agreement shall revert to the Author.

3 Self Archiving

Author is permitted to self-archive a preprint and the accepted manuscript version of their Article.

a) A preprint is the version of the Article before peer-review has taken place ("Preprint”). Prior to acceptance for publication, Author retains the right to make a Preprint of their Article available on any of the following: their own personal, self-maintained website; a legally compliant Preprint server such as but not limited to arXiv and bioRxiv. Once the Article has been published, the Author should update the acknowledgement and provide a link to the definitive version on the publisher’s website: “This is a preprint of an article published in [insert journal title]. The final authenticated version is available online at: https://doi.org/[insert DOI]”

b) The accepted manuscript version, by industry standard called the “Author’s Accepted Manuscript” (“AAM”) is the version accepted for publication in a journal following peer review but prior to copyediting and typesetting that can be made available under the following conditions:
(i) Author retains the right to make an AAM of the Article available on their own personal, self-maintained website immediately on acceptance, (ii) Author retains the right to make an AAM of the Article available for public release on any of the following 12 months after first publication ("Embargo Period"): their employer’s internal website; their institutional and/or funder repositories; AAMs may also be deposited in such repositories immediately on acceptance, provided that they are not made publicly available until after the Embargo Period.

An acknowledgement in the following form should be included, together with a link to the published version on the publisher’s website: “This is a post-peer-review, pre-copyedit version of an article published in [insert journal title]. The final authenticated version is available online at: http://dx.doi.org/[insert DOI]”.

4 Reuse Rights

Author retains the following non-exclusive rights for the published version provided that, when reproducing the Article or extracts from it, the Author acknowledges and references first publication in the Journal according to current citation standards. In any event the acknowledgement should contain as a minimum, "First published in [Journal name, volume, page number, year] by Springer Nature".

a) to reuse graphic elements created by the Author and contained in the Article, in presentations and other works created by them;

b) the Author and any academic institution where they work at the time may reproduce the Article for the purpose of course teaching (but not for inclusion in course pack material for onward sale by libraries and institutions);

c) to reuse the published version of the Article or any part in a thesis written by the same Author , and to make a copy of that thesis available in a repository of the Author(s)' awarding academic institution, or other repository required by the awarding academic institution. An acknowledgement should be included in the citation: "Reproduced with permission from Springer Nature"; and

d) to reproduce, or to allow a third party to reproduce the Article, in whole or in part, in any other type of work (other than thesis) written by the Author for distribution by a publisher after an embargo period of 12 months.

5 Warranties & Representations

Author warrants and represents that:

a)
   i. the Author is the sole copyright owner or has been authorised by any additional copyright owner(s) to grant the rights defined in clause 2,
   ii. the Article does not infringe any intellectual property rights (including without limitation copyright, database rights or trade mark rights) or other third party rights and no licence from or payments to a third party are required to publish the Article,
   iii. the Article has not been previously published or licensed,
   iv. if the Article contains materials from other sources (e.g. illustrations, tables, text quotations), Author has obtained written permissions to the extent necessary from the copyright holder(s), to license to the Licensee the same rights as set out in clause 2 and has cited any such materials correctly;

b) all of the facts contained in the Article are according to the current body of research true and accurate;

c) nothing in the Article is obscene, defamatory, violates any right of privacy or publicity, infringes any other human, personal or other rights of any person or entity or is otherwise unlawful and that informed consent to publish has been obtained for all research participants;

d) nothing in the Article infringes any duty of confidentiality which Author might owe to anyone else or violates any contract, express or implied, of Author. All of the institutions in which work recorded in the Article was created or carried out have authorised and approved such research and publication; and

e) the signatory who has signed this agreement has full right, power and authority to enter into this agreement on behalf of all of the Authors.

6 Cooperation

a) Author shall cooperate fully with the Licensee in relation to any legal action that might arise from the publication of the Article, and the Author shall give the Licensee access at reasonable times to any relevant accounts, documents and records within the power or control of the Author. Author agrees that the distributing entity is intended to have the benefit of and shall have the right to enforce the terms of this agreement.

b) Author authorises the Licensee to take such steps as it considers necessary at its own expense in the Author's name(s) and on their behalf if the Licensee believes that a third party is infringing or is likely to infringe copyright in the Article including but not limited to initiating legal proceedings.

7 Author List

Changes of authorship, including, but not limited to, changes in the corresponding author or the sequence of authors, are not permitted after acceptance of a manuscript.

8 Corrections

Author agrees that the Licensee may retract the Article or publish a correction or other notice in relation to the Article if the Licensee considers in its reasonable opinion that such actions are appropriate from a legal, editorial or research integrity perspective.

9 Governing Law

This agreement shall be governed by, and shall be construed in accordance with, the laws of New York State. The courts of competent jurisdiction in New York, N.Y. shall have the exclusive jurisdiction.

CCC RightsLink®

Home | Help | Email Support | Sign in | Create Account

IEEE Requesting permission to reuse content from an IEEE publication

Mixed-Criticality Multicore Scheduling of Real-Time Gang Task Systems

Conference Proceedings: 2019 IEEE Real-Time Systems Symposium (RTSS)

Author: Ashik ahmed Bhuiyan

Publisher: IEEE

Date: Dec 2019

Thesis / Dissertation Reuse

The IEEE does not require individuals working on a thesis to obtain a formal reuse license, however, you may print out this statement to be used as a permission grant:

Requirements to be followed when using any portion (e.g., figure, graph, table, or textual material) of an IEEE copyrighted paper in a thesis:

1) In the case of textual material (e.g., using short quotes or referring to the work within these papers) users must give full credit to the original source (author, paper, publication) followed by the IEEE copyright line © 2011 IEEE.
2) In the case of illustrations or tabular material, we require that the copyright line © [Year of original publication] IEEE appear prominently with each reprinted figure and/or table.
3) If a substantial portion of the original paper is to be used, and if you are not the senior author, also obtain the senior author's approval.

Requirements to be followed when using an entire IEEE copyrighted paper in a thesis:

1) The following IEEE copyright/ credit notice should be placed prominently in the references: © [year of original publication] IEEE. Reprinted, with permission, from [author names, paper title, IEEE publication title, and month/year of publication]
2) Only the accepted version of an IEEE copyrighted paper can be used when posting the paper or your thesis on-line.
3) In placing the thesis on the author's university website, please display the following message in a prominent place on the website: In reference to IEEE copyrighted material which is used with permission in this thesis, the IEEE does

BACK **CLOSE WINDOW**

ACM Copyright and Audio/Video Release

Title of the Work: Precise Scheduling of Mixed-Criticality Tasks by Varying Processor Speed
Submission ID:33
Author/Presenter(s): Ashik Ahmed Bhuiyan:University of Central Florida;Zhishan Guo:University of Central Florida;Sai Sruti:Missouri University of Science and Technology;Kecheng Yang:Texas State University
Type of material: Full Paper

Publication and/or Conference Name: 27th International Conference on Real-Time Networks and Systems Proceedings

I. **Copyright Transfer, Reserved Rights and Permitted Uses**

published in the ACM Digital Library, the rights revert back to the author(s).

When preparing your paper for submission using the ACM TeX templates, the rights and permissions information and the bibliographic strip must appear on the lower left hand portion of the first page.

The new ACM Consolidated TeX template Version 1.3 and above automatically creates and positions these text blocks for you based on the code snippet which is system-generated based on your rights management choice and this particular conference.

NOTE: For authors using the ACM Microsoft Word Master Article Template and Publication Workflow, The ACM Publishing System (TAPS) will add the rights statement to your papers for you. Please check with your conference contact for information regarding submitting your source file(s) for processing.

Please copy and paste \setcopyright{acmcopyright} before \begin{document} and please copy and paste the following code snippet into your TeX file between \begin{document} and \maketitle, either after or before CCS codes.

```
\copyrightyear{2019}
\acmYear{2019}
\acmConference[RTNS 2019]{27th International Conference on Real-Time Networks and
Systems}{November 6--8, 2019}{Toulouse, France}
\acmBooktitle{27th International Conference on Real-Time Networks and Systems (RTNS 2019),
November 6--8, 2019, Toulouse, France}
\acmPrice{15.00}
\acmDOI{10.1145/3356401.3356410}
\acmISBN{978-1-4503-7223-7/19/11}
```

If you are using the ACM Microsoft Word template, or still using an older version of the ACM TeX template, or the current versions of the ACM SIGCHI, SIGGRAPH, or SIGPLAN TeX templates, you must copy and paste the following text block into your document as per the instructions provided with the templates you are using:

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from Permissions@acm.org.

RTNS 2019, November 6–8, 2019, Toulouse, France
© 2019 Association for Computing Machinery.
ACM ISBN 978-1-4503-7223-7/19/11…$15.00
https://doi.org/10.1145/3356401.3356410

NOTE: Make sure to include your article's DOI as part of the bibstrip data; DOIs will be registered and become active shortly after publication in the ACM Digital Library. Once you have your camera ready copy ready, please send your source files and PDF to your event contact for processing.

✓ A. Assent to Assignment. I hereby represent and warrant that I am the sole owner (or authorized agent of the copyright owner(s)), with the exception of third party materials detailed in section III below. I have obtained permission for any third-party material included in the Work.

☐ B. Declaration for Government Work. I am an employee of the National Government of my country and my Government claims rights to this work, or it is not copyrightable (Government work is classified as Public Domain in U.S. only)

Are any of the co-authors, employees or contractors of a National Government? Yes [✓ No]

II. Permission For Conference Recording and Distribution

* Your Audio/Video Release is conditional upon you agreeing to the terms set out below.

I hereby grant permission for ACM to include my name, likeness, presentation and comments in any and all forms, for the Conference and/or Publication.

I further grant permission for ACM to record and/or transcribe and reproduce my presentation as part of the ACM Digital Library, and to distribute the same for sale in complete or partial form as part of an ACM product on CD-ROM, DVD, webcast, USB device, streaming video or any other media format now or hereafter known.

I understand that my presentation will not be sold separately as a stand-alone product without my direct consent. Accordingly, I give ACM the right to use my image, voice, pronouncements, likeness, and my name, and any biographical material submitted by me, in connection with the Conference and/or Publication, whether used in excerpts or in full, for distribution described above and for any associated advertising or exhibition.

Do you agree to the above Audio/Video Release? [✓ Yes] No

III. Auxiliary Material

Do you have any Auxiliary Materials? Yes [✓ No]

IV. Third Party Materials

In the event that any materials used in my presentation or Auxiliary Materials contain the work of third-party individuals or organizations (including copyrighted music or movie excerpts or anything not owned by me), I understand that it is my responsibility to secure any necessary permissions and/or licenses for print and/or digital publication, and cite or attach them below.

[✓ We/I have not used third-party material.]
We/I have used third-party materials and have necessary permissions.

V. Artistic Images

If your paper includes images that were created for any purpose other than this paper and to which you or your employer claim copyright, you must complete Part V and be sure to include a notice of copyright with each such image in the paper.
[✓ We/I do not have any artistic images.]
We/I have any artistic images.

VI. Representations, Warranties and Covenants

The undersigned hereby represents, warrants and covenants as follows:

Funding Agents

1. National Science Foundation award number(s): CNS-1850851

DATE: **08/28/2019** sent to ashik@Knights.ucf.edu at **15:08:33**

ACM Copyright and Audio/Video Release

Title of the Work: F2VD: Fluid Rates to Virtual Deadlines for Precise Mixed-Criticality Scheduling on a Varying-Speed Processor

Author/Presenter(s): Kecheng Yang:Texas State Univ. ;Ashikahmed Bhuiyan:Univ. of Central Florida;Zhishan Guo:Univ. of Central Florida
Type of material: Full Paper

Publication and/or Conference Name: ICCAD '20: IEEE/ACM International Conference on Computer-Aided Design Proceedings

I. **Copyright Transfer, Reserved Rights and Permitted Uses**

* Your Copyright Transfer is conditional upon you agreeing to the terms set out below.

Copyright to the Work and to any supplemental files integral to the Work which are submitted with it for review and publication such as an extended proof, a PowerPoint outline, or appendices that may exceed a printed page limit, (including without limitation, the right to publish the Work in whole or in part in any and all forms of media, now or hereafter known) is hereby transferred to the ACM (for Government work, to the extent transferable) effective as of the date of this agreement, on the understanding that the Work has been accepted for publication by ACM.

Reserved Rights and Permitted Uses

(a) All rights and permissions the author has not granted to ACM are reserved to the Owner, including all other proprietary rights such as patent or trademark rights.

(b) Furthermore, notwithstanding the exclusive rights the Owner has granted to ACM, Owner shall have the right to do the following:

(i) Reuse any portion of the Work, without fee, in any future works written or edited by the Author, including books, lectures and presentations in any and all media.

(ii) Create a "Major Revision" which is wholly owned by the author

(iii) Post the Accepted Version of the Work on (1) the Author's home page, (2) the Owner's institutional repository, (3) any repository legally mandated by an agency funding the research on which the Work is based, and (4) any non-commercial repository or aggregation that does not duplicate ACM tables of contents, i.e., whose patterns of links do not substantially duplicate an ACM-copyrighted volume or issue. Non-commercial repositories are here understood as repositories owned by non-profit organizations that do not charge a fee for accessing deposited articles and that do not sell advertising or otherwise profit from serving articles.

(iv) Post an "Author-Izer" link enabling free downloads of the Version of Record in the ACM Digital Library on (1) the Author's home page or (2) the Owner's institutional repository;

(v) Prior to commencement of the ACM peer review process, post the version of the Work as submitted to ACM ("Submitted Version" or any earlier versions) to non-peer reviewed servers;

(vi) Make free distributions of the final published Version of Record internally to the Owner's employees, if applicable;

(vii) Make free distributions of the published Version of Record for Classroom and Personal Use;

(viii) Bundle the Work in any of Owner's software distributions; and

(ix) Use any Auxiliary Material independent from the Work. (x) If your paper is withdrawn before it is published in the ACM Digital Library, the rights revert back to the author(s).

When preparing your paper for submission using the ACM TeX templates, the rights and permissions information and the bibliographic strip must appear on the lower left hand portion of the first page.

The new ACM Consolidated TeX template Version 1.3 and above automatically creates and positions these text blocks for you based on the code snippet which is system-generated based on your rights management choice and this particular conference.

NOTE: For authors using the ACM Microsoft Word Master Article Template and Publication Workflow, The ACM Publishing System (TAPS) will add the rights statement to your papers for you. Please check with your conference contact for information regarding submitting your source file(s) for processing.

Please put the following LaTeX commands in the preamble of your document - i.e., before \begin{document}:

```
\copyrightyear{2020}
\acmYear{2020}
\setcopyright{acmcopyright}\acmConference[ICCAD '20]{IEEE/ACM International Conference on Computer-Aided Design}{November 2--5, 2020}{Virtual Event, USA}
\acmBooktitle{IEEE/ACM International Conference on Computer-Aided Design (ICCAD '20), November 2--5, 2020, Virtual Event, USA}
\acmPrice{15.00}
\acmDOI{10.1145/3400302.3415716}
\acmISBN{978-1-4503-8026-3/20/11}
```

NOTE: For authors using the ACM Microsoft Word Master Article Template and Publication Workflow, The ACM Publishing System (TAPS) will add the rights statement to your papers for you. Please check with your conference contact for information regarding submitting your source file(s) for processing.

If you are using the ACM Interim Microsoft Word template, or still using or older versions of the ACM SIGCHI template, you must copy and paste the following text block into your document as per the instructions provided with the templates you are using:

© 2020 Association for Computing Machinery.
ACM ISBN 978-1-4503-8026-3/20/11...$15.00
https://doi.org/10.1145/3400302.3415716

NOTE: Make sure to include your article's DOI as part of the bibstrip data; DOIs will be registered and become active shortly after publication in the ACM Digital Library. Once you have your camera ready copy ready, please send your source files and PDF to your event contact for processing.

[✓] A. Assent to Assignment. I hereby represent and warrant that I am the sole owner (or authorized agent of the copyright owner(s)), with the exception of third party materials detailed in section III below. I have obtained permission for any third-party material included in the Work.

☐ B. Declaration for Government Work. I am an employee of the National Government of my country and my Government claims rights to this work, or it is not copyrightable (Government work is classified as Public Domain in U.S. only)

Are any of the co-authors, employees or contractors of a National Government? Yes [✓ No]

II. Permission For Conference Recording and Distribution

* Your Audio/Video Release is conditional upon you agreeing to the terms set out below.

I hereby grant permission for ACM to include my name, likeness, presentation and comments in any and all forms, for the Conference and/or Publication.

I further grant permission for ACM to record and/or transcribe and reproduce my presentation as part of the ACM Digital Library, and to distribute the same for sale in complete or partial form as part of an ACM product on CD-ROM, DVD, webcast, USB device, streaming video or any other media format now or hereafter known.

I understand that my presentation will not be sold separately as a stand-alone product without my direct consent. Accordingly, I give ACM the right to use my image, voice, pronouncements, likeness, and my name, and any biographical material submitted by me, in connection with the Conference and/or Publication, whether used in excerpts or in full, for distribution described above and for any associated advertising or exhibition.

Do you agree to the above Audio/Video Release? [✓ Yes] No

III. Auxiliary Material

Do you have any Auxiliary Materials? Yes [✓ No]

IV. Third Party Materials

In the event that any materials used in my presentation or Auxiliary Materials contain the work of third-party individuals or organizations (including copyrighted music or movie excerpts or anything not owned by me), I understand that it is my responsibility to secure any necessary permissions and/or licenses for print and/or digital publication, and cite or attach them below.

[✓ We/I have not used third-party material.]
We/I have used third-party materials and have necessary permissions.

V. Artistic Images
If your paper includes images that were created for any purpose other than this paper and to which you or your employer claim copyright, you must complete Part V and be sure to include a notice of copyright with each such image in the paper.
[✓ We/I do not have any artistic images.]
We/I have any artistic images.

VI. Representations, Warranties and Covenants

The undersigned hereby represents, warrants and covenants as follows:

(a) Owner is the sole owner or authorized agent of Owner(s) of the Work;

(b) The undersigned is authorized to enter into this Agreement and grant the rights included in this license to ACM;

(c) The Work is original and does not infringe the rights of any third party; all permissions for use of third-party materials consistent in scope and duration with the rights granted to ACM have been obtained, copies of such permissions have been provided to ACM, and the Work as submitted to ACM clearly and accurately indicates the credit to the proprietors of any such third-party materials (including any applicable copyright notice), or will be revised to indicate such credit;

(d) The Work has not been published except for informal postings on non-peer reviewed servers, and Owner covenants to use best efforts to place ACM DOI pointers on any such prior postings;

(e) The Auxiliary Materials, if any, contain no malicious code, virus, trojan horse or other software routines or hardware components designed to permit unauthorized access or to disable, erase or otherwise harm any computer systems or software; and

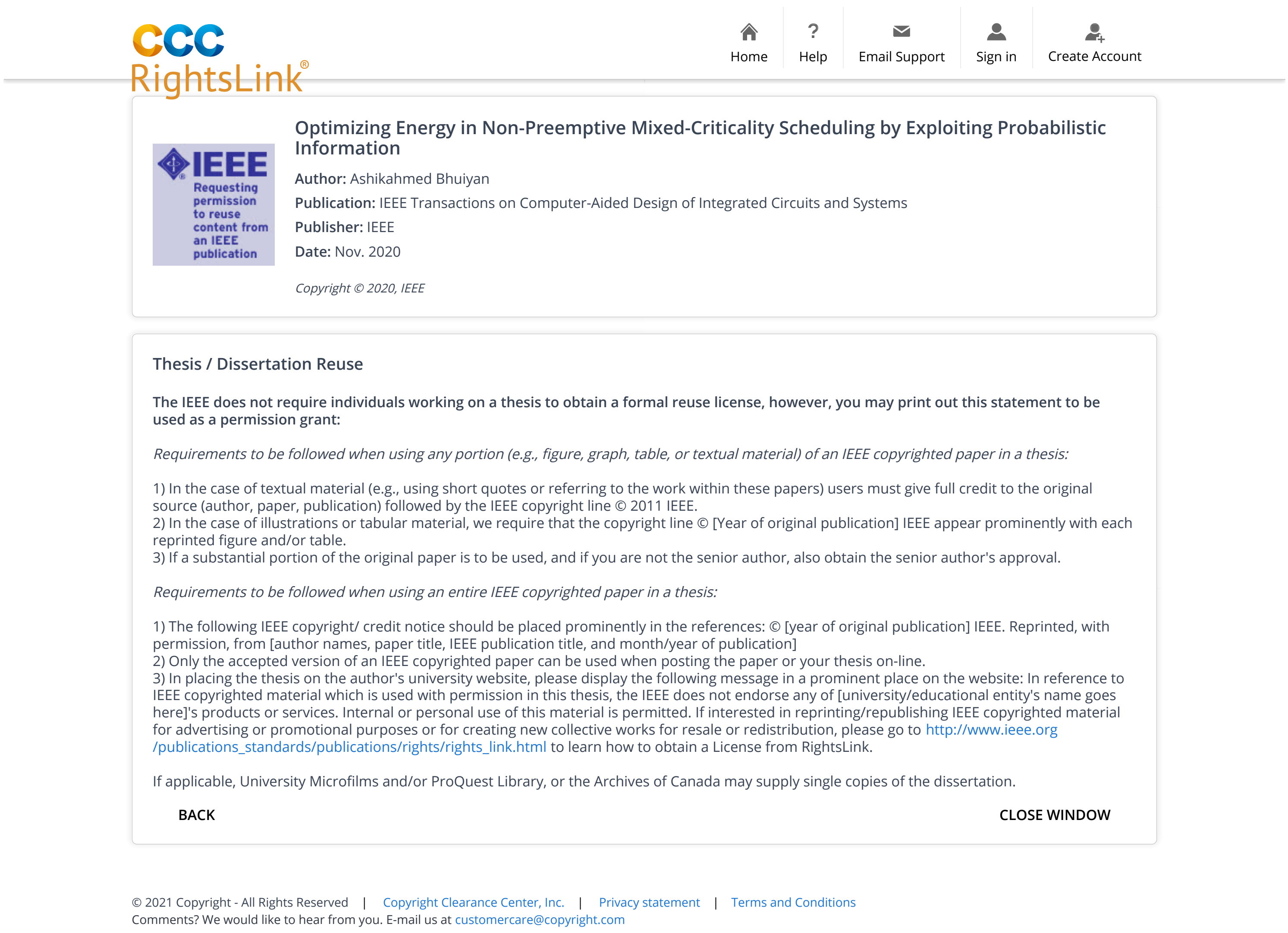
(f) The Artistic Images, if any, are clearly and accurately noted as such (including any applicable copyright notice) in the Submitted Version.

✓ I agree to the Representations, Warranties and Covenants

Funding Agents

1. Division of Computer and Network Systems award number(s): 1850851

DATE: **08/05/2020** sent to ashik@knights.ucf.edu at **11:08:40**



Home | Help | Email Support | Sign in | Create Account

Optimizing Energy in Non-Preemptive Mixed-Criticality Scheduling by Exploiting Probabilistic Information

Author: Ashikahmed Bhuiyan
Publication: IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems
Publisher: IEEE
Date: Nov. 2020

Copyright © 2020, IEEE

Thesis / Dissertation Reuse

The IEEE does not require individuals working on a thesis to obtain a formal reuse license, however, you may print out this statement to be used as a permission grant:

Requirements to be followed when using any portion (e.g., figure, graph, table, or textual material) of an IEEE copyrighted paper in a thesis:

1) In the case of textual material (e.g., using short quotes or referring to the work within these papers) users must give full credit to the original source (author, paper, publication) followed by the IEEE copyright line © 2011 IEEE.
2) In the case of illustrations or tabular material, we require that the copyright line © [Year of original publication] IEEE appear prominently with each reprinted figure and/or table.
3) If a substantial portion of the original paper is to be used, and if you are not the senior author, also obtain the senior author's approval.

Requirements to be followed when using an entire IEEE copyrighted paper in a thesis:

1) The following IEEE copyright/ credit notice should be placed prominently in the references: © [year of original publication] IEEE. Reprinted, with permission, from [author names, paper title, IEEE publication title, and month/year of publication]
2) Only the accepted version of an IEEE copyrighted paper can be used when posting the paper or your thesis on-line.
3) In placing the thesis on the author's university website, please display the following message in a prominent place on the website: In reference to IEEE copyrighted material which is used with permission in this thesis, the IEEE does not endorse any of [university/educational entity's name goes here]'s products or services. Internal or personal use of this material is permitted. If interested in reprinting/republishing IEEE copyrighted material for advertising or promotional purposes or for creating new collective works for resale or redistribution, please go to http://www.ieee.org/publications_standards/publications/rights/rights_link.html to learn how to obtain a License from RightsLink.

If applicable, University Microfilms and/or ProQuest Library, or the Archives of Canada may supply single copies of the dissertation.

BACK | CLOSE WINDOW

LIST OF REFERENCES

[1] DO-178C - software considerations in airborne systems and equipment certification, 2011.

[2] Kunal Agrawal and Sanjoy Baruah. A measurement-based model for parallel real-time tasks. In *30th Euromicro Conference on Real-Time Systems (ECRTS 2018)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.

[3] Bader Alahmad and Sathish Gopalakrishnan. Isochronous execution models for high-assurance real-time systems. In *HASE*. IEEE, 2019.

[4] Björn Andersson and Dionisio de Niz. Analyzing global-EDF for multiprocessor scheduling of parallel tasks. In *OPODIS*. Springer, 2012.

[5] Muhammad Awan, Konstantinos Bletsas, Pedro Souto, and Eduardo Tovar. Semi-partitioned mixed-criticality scheduling. In *ARCS*. Springer, 2017.

[6] Muhammad Ali Awan, Damien Masson, and Eduardo Tovar. Energy efficient mapping of mixed criticality applications on unrelated heterogeneous multicore platforms. In *SIES*. IEEE, 2016.

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

[8] Hyeongboo Baek and Jinkyu Lee. Incorporating security constraints into mixed-criticality real-time scheduling. *IEICE TRANSACTIONS on Information and Systems*, 100(9):2068–2080, 2017.

[9] Mario Bambagini, Mauro Marinoni, Hakan Aydin, and Giorgio Buttazzo. Energy-aware scheduling for real-time systems: A survey. *ACM TECS*, 15(1):7, 2016.

[10] Sanjoy Baruah. Improved multiprocessor global schedulability analysis of sporadic DAG task systems. In *26th Euromicro Conference on Real-Time Systems*, pages 97–105. IEEE, 2014.

[11] Sanjoy Baruah. The federated scheduling of systems of mixed-criticality sporadic DAG tasks. In *RTSS*. IEEE, 2016.

[12] Sanjoy Baruah. Mixed-criticality scheduling theory: Scope, promise, and limitations. *IEEE DESIGN AND TEST*, 35(2):31–37, 2018.

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

[14] Sanjoy Baruah, Vincenzo Bonifaci, Gianlorenzo DAngelo, Haohan Li, Alberto Marchetti-Spaccamela, Suzanne Van Der Ster, and Leen Stougie. The preemptive uniprocessor scheduling of mixed-criticality implicit-deadline sporadic task systems. In *ECRTS*. IEEE, 2012.

[15] Sanjoy Baruah, Vincenzo Bonifaci, Gianlorenzo D'angelo, Alberto Marchetti-Spaccamela, Suzanne Van Der Ster, and Leen Stougie. Mixed-criticality scheduling of sporadic task systems. In *ESA*. Springer, 2011.

[16] Sanjoy Baruah, Vincenzo Bonifaci, and Alberto Marchetti-Spaccamela. The global EDF scheduling of systems of conditional sporadic DAG tasks. In *27th Euromicro Conference on Real-Time Systems*, pages 222–231. IEEE, 2015.

[17] Sanjoy Baruah, Vincenzo Bonifaci, Alberto Marchetti-Spaccamela, Leen Stougie, and Andreas Wiese. A generalized parallel task model for recurrent real-time processes. In *Real-Time Systems Symposium (RTSS), IEEE 33rd*, pages 63–72. IEEE, 2012.

[18] Sanjoy Baruah, Alan Burns, and Robert Davis. Response-time analysis for mixed criticality systems. In *2011 IEEE 32nd Real-Time Systems Symposium*, pages 34–43. IEEE, 2011.

[19] Sanjoy Baruah, Arvind Easwaran, and Zhishan Guo. MC-Fluid: simplified and optimally quantified. In *2015 IEEE Real-Time Systems Symposium*, pages 327–337. IEEE, 2015.

[20] Sanjoy Baruah and Zhishan Guo. Mixed-criticality scheduling upon varying-speed processors. In *2013 IEEE 34th Real-Time Systems Symposium*, pages 68–77. IEEE, 2013.

[21] Sanjoy Baruah and Zhishan Guo. Scheduling mixed-criticality implicit-deadline sporadic task systems upon a varying-speed processor. In *2014 IEEE Real-Time Systems Symposium*, pages 31–40. IEEE, 2014.

[22] Cristina Bazgan, Bruno Escoffier, and Vangelis Th. Paschos. Completeness in standard and differential approximation classes: Poly-apx- and ptas-completeness. *Theoretical Computer Science*, 339(2-3):272–292, 2005.

[23] 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):84, 2018.

[24] 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.

[25] Ashikahmed Bhuiyan, Federico Reghenzani, William Fornaciari, and Zhishan Guo. Optimizing energy in non-preemptive mixed-criticality scheduling by exploiting probabilistic

information. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 39(11):3906–3917, 2020.

[26] 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. ACM, 2019.

[27] Ashikahmed Bhuiyan, Kecheng Yang, Samsil Arefin, Abusayeed Saifullah, Nan Guan, and Zhishan Guo. Mixed-criticality multicore scheduling of real-time gang task systems. In *2019 IEEE Real-Time Systems Symposium (RTSS)*, pages 469–480. IEEE, 2019.

[28] Enrico Bini, Giorgio Buttazzo, and Giuseppe Lipari. Minimizing CPU energy in real-time systems with discrete speed management. *ACM Transactions on Embedded Computing Systems (TECS)*, 8(4):31, 2009.

[29] M Bolado, Hector Posadas, Javier Castillo, Pablo Huerta, Pablo Sanchez, C Sánchez, H Fouren, and Francisco Blasco. Platform based on open-source cores for industrial applications. In *Design, Automation and Test in Europe Conference and Exhibition, 2004. Proceedings*, volume 2, pages 1014–1019. IEEE, 2004.

[30] Vincenzo Bonifaci, Alberto Marchetti-Spaccamela, Sebastian Stiller, and Andreas Wiese. Feasibility analysis in the sporadic DAG task model. In *25th Euromicro Conference on Real-Time Systems*, pages 225–233. IEEE, 2013.

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

[32] Alan Burns and Robert Davis. Mixed criticality systems-a review. *Department of Computer Science, University of York, Tech. Rep*, pages 1–69, 2013.

[33] Alan Burns and Robert Davis. A survey of research into mixed criticality systems. *ACM Computing Surveys (CSUR)*, 50(6):82, 2018.

[34] Giorgio C Buttazzo. *Hard real-time computing systems: predictable scheduling algorithms and applications*, volume 24. Springer Science & Business Media, 2011.

[35] Nicola Capodieci, Roberto Cavicchioli, Marko Bertogna, and Aingara Paramakuru. Deadline-based scheduling for GPU with preemption support. In *2018 IEEE Real-Time Systems Symposium (RTSS)*, pages 119–130. IEEE, 2018.

[36] David Chandler. Introduction to modern statistical mechanics. *pp. 288. Foreword by David Chandler. Oxford University Press, Sep 1987. ISBN-10: 0195042778. ISBN-13: 9780195042771*, page 288, 1987.

[37] Gang Chen, Kai Huang, and Alois Knoll. Energy optimization for real-time multiprocessor system-on-chip with optimal DVFS and DPM combination. *ACM Transactions on Embedded Computing Systems (TECS)*, 13(3s):111, 2014.

[38] Jian-Jia Chen and Chin-Fu Kuo. Energy-efficient scheduling for real-time systems on dynamic voltage scaling (DVS) platforms. In *RTCSA*. IEEE, 2007.

[39] Jian-Jia Chen, Andreas Schranzhofer, and Lothar Thiele. Energy minimization for periodic real-time tasks on heterogeneous processing units. In *Parallel & Distributed Processing. IPDPS 2009. IEEE International Symposium on*, pages 1–12. IEEE, 2009.

[40] Hoon Sung Chwa, Jaebaek Seo, Jinkyu Lee, and Insik Shin. Optimal real-time scheduling on two-type heterogeneous multicore platforms. In *RTSS*. IEEE, 2015.

[41] Alexei Colin, Arvind Kandhalu, and Ragunathan Rajkumar. Energy-efficient allocation of real-time applications onto heterogeneous processors. In *RTCSA*, 2014.

[42] Daniel Cordeiro, Gregory Mouni, Swann Perarnau, Denis Trystram, Jean-Marc Vincent, and Frederic Wagner. Random graph generation for scheduling simulations. In *Proceedings of the 3rd international ICST conference on simulation tools and techniques*, page 60. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2010.

[43] Robert Davis and Liliana Cucu-Grosjean. A survey of probabilistic timing analysis techniques for real-time systems. *Leibniz Transactions on Embedded Systems*, 6(1):03–1, 2019.

[44] Vinay Devadas and Hakan Aydin. Coordinated power management of periodic real-time tasks on chip multiprocessors. In *Green Computing Conference, 2010 International*, pages 61–72. IEEE, 2010.

[45] Björn Döbel, Hermann Härtig, and Michael Engel. Operating system support for redundant multithreading. In *Proceedings of the tenth ACM international conference on Embedded software*, pages 83–92, 2012.

[46] Zheng Dong and Cong Liu. Analysis techniques for supporting hard real-time sporadic gang task systems. In *RTSS*. IEEE, 2017.

[47] Adam Dunkels, Fredrik Osterlind, Nicolas Tsiftes, and Zhitao He. Software-based on-line energy estimation for sensor nodes. In *Proceedings of the 4th workshop on Embedded networked sensors*, pages 28–32. ACM, 2007.

[48] Arvind Easwaran. Demand-based scheduling of mixed-criticality sporadic tasks on one processor. In *RTSS*. IEEE, 2013.

[49] Glenn A Elliott, Bryan C Ward, and James H Anderson. GPUSync: A framework for real-time GPU management. In *RTSS*. IEEE, 2013.

[50] 2017. https://github.com/tuxamito/emoxu3.

[51] Michael Engel and Björn Döbel. The reliable computing base–a paradigm for software-based reliability. *INFORMATIK 2012*, 2012.

[52] Rolf Ernst and Marco Di Natale. Mixed criticality systems—a history of misconceptions? *IEEE Design & Test*, 33(5):65–74, 2016.

[53] Alexandre Esper, Geoffrey Nelissen, Vincent Nélis, and Eduardo Tovar. How realistic is the mixed-criticality real-time system model? In *RTNS*. ACM, 2015.

[54] Dror G Feitelson and Larry Rudolph. Gang scheduling performance benefits for fine-grain synchronization. *Journal of Parallel and distributed Computing*, 16(4):306–318, 1992.

[55] Nathan Fisher, Jian-Jia Chen, Shengquan Wang, and Lothar Thiele. Thermal-aware global real-time scheduling on multicore systems. In *Real-Time and Embedded Technology and Applications Symposium, 2009. RTAS 2009. 15th IEEE*, pages 131–140. IEEE, 2009.

[56] 2018. https://www.mathworks.com/help/optim/ug/fmincon.html.

[57] Shelby Hyatt Funk. *EDF scheduling on heterogeneous multiprocessors*. University of North Carolina at Chapel Hill, 2004.

[58] 2018. https://en.wikipedia.org/wiki/Gamma_distribution.

[59] Edward Gehringer, Daniel Siewiorek, and Zary Segall. *Parallel processing: the Cm* experience*. Digital Press, 1987.

[60] Joël Goossens and Vandy Berten. Gang FTP scheduling of periodic and parallel rigid real-time tasks. *arXiv preprint arXiv:1006.2617*, 2010.

[61] Joël Goossens and Pascal Richard. Optimal scheduling of periodic gang tasks. *Leibniz transactions on embedded systems*, 3(1):04–1, 2016.

[62] Michael Grant and Stephen Boyd. CVX: Matlab software for disciplined convex programming, version 2.1. `http://cvxr.com/cvx`, March 2014.

[63] Nan Guan, Pontus Ekberg, Martin Stigge, and Wang Yi. Improving the scheduling of certifiable mixed-criticality sporadic task systems. *Technical Report 2013–008*, 2013.

[64] Yifeng Guo, Dakai Zhu, and Hakan Aydin. Reliability-aware power management for parallel real-time applications with precedence constraints. In *Green Computing Conference and Workshops (IGCC), 2011 International*, pages 1–8. IEEE, 2011.

[65] Zhishan Guo and Sanjoy Baruah. The concurrent consideration of uncertainty in wcets and processor speeds in mixed-criticality systems. In *RTNS*. ACM, 2015.

[66] 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.

[67] 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.

[68] Tarek Hagras and Jan Janecek. A high performance, low complexity algorithm for compile-time job scheduling in homogeneous computing environments. In *Parallel Processing Workshops*. IEEE, 2003.

[69] Magnús Halldórssonz and Jaikumar Radhakrishnan. Greed is good: Approximating independent sets in sparse and bounded-degree graphs. *Algorithmica*, 18(1):145–163, 1997.

[70] Sebastian Herbert and Diana Marculescu. Analysis of dynamic voltage/frequency scaling in chip-multiprocessors. In *ISLPED*. IEEE, 2007.

[71] Jason Howard, Saurabh Dighe, Sriram Vangal, Gregory Ruhl, Nitin Borkar, Shailendra Jain, Vasantha Erraguntla, Michael Konow, Michael Riepen, Matthias Gries, et al. A 48-core ia-32 processor in 45 nm cmos using on-die message-passing and DVFS for performance and power scaling. *IEEE Journal of Solid-State Circuits*, 46(1):173–183, 2011.

[72] Pengcheng Huang, Pratyush Kumar, Georgia Giannopoulou, and Lothar Thiele. Energy efficient DVFS scheduling for mixed-criticality systems. In *EMSOFT*. IEEE, 2014.

[73] Intel. Intel cilkplus v1.2, 2013.

[74] Kevin Jeffay, Donald Stone, and Donelson Smith. Kernel support for live digital audio and video. *Computer communications*, 15(6):388–395, 1992.

[75] Ravindra Jejurikar. Energy aware non-preemptive scheduling for hard real-time systems. In *17th Euromicro Conference on Real-Time Systems (ECRTS'05)*, pages 21–30. IEEE, 2005.

[76] Shinpei Kato and Yutaka Ishikawa. Gang EDF scheduling of parallel task systems. In *30th IEEE Real-Time Systems Symposium (RTSS)*, pages 459–468. IEEE, 2009.

[77] Junsung Kim, Hyoseung Kim, Karthik Lakshmanan, and Ragunathan Raj Rajkumar. Parallel scheduling for cyber-physical systems: Analysis and case study on a self-driving car. In *ICCPS*. ACM, 2013.

[78] Jaewoo Lee, Kieu Phan, Xiaozhe Gu, Jiyeon Lee, Arvind Easwaran, Insik Shin, and Insup Lee. MC-Fluid: Fluid model-based mixed-criticality scheduling on multiprocessors. In *RTSS*. IEEE, 2014.

[79] Haohan Li. *Scheduling mixed-criticality real-time systems*. PhD thesis, The University of North Carolina at Chapel Hill, 2013.

[80] Haohan Li and Sanjoy Baruah. An algorithm for scheduling certifiable mixed-criticality sporadic task systems. In *RTSS*. IEEE, 2010.

[81] Jing Li, Kunal Agrawal, Chenyang Lu, and Christopher Gill. Analysis of global EDF for parallel tasks. In *25th Euromicro Conference on Real-Time Systems*, pages 3–13. IEEE, 2013.

[82] Jing Li, Jian-Jia Chen, Kunal Agrawal, Chenyang Lu, Chris Gill, and Abusayeed Saifullah. Analysis of federated and global scheduling for parallel real-time tasks. In *26th Euromicro Conference on Real-Time Systems*, pages 85–96. IEEE, 2014.

[83] Jing Li, David Ferry, Shaurya Ahuja, Kunal Agrawal, Christopher Gill, and Chenyang Lu. Mixed-criticality federated scheduling for parallel real-time tasks. *Real-Time Systems*, 53(5):760–811, 2017.

[84] Yau-Tsun Steven Li and Sharad Malik. Performance analysis of embedded software using implicit path enumeration. In *Workshop on Languages, compilers, & tools for real-time systems*, pages 88–98, 1995.

[85] Cong Liu, Jian Li, Wei Huang, Juan Rubio, Evan Speight, and Xiaozhu Lin. Power-efficient time-sensitive mapping in heterogeneous systems. In *Proceedings of the 21st international conference on Parallel architectures and compilation techniques*, pages 23–32. ACM, 2012.

[86] Di Liu, Jelena Spasic, Gang Chen, and Todor Stefanov. Energy-efficient mapping of real-time streaming applications on cluster heterogeneous mpsocs. In *ESTIMedia*. IEEE, 2015.

[87] Guangdong Liu, Ying Lu, Shige Wang, and Zonghua Gu. Partitioned multiprocessor scheduling of mixed-criticality parallel jobs. In *RTCSA*. IEEE, 2014.

[88] Alexander Maxiaguine, Simon Kunzli, and Lothar Thiele. Workload characterization model for tasks with variable execution demand. In *Proceedings Design, Automation and Test in Europe Conference and Exhibition*, volume 2, pages 1040–1045. IEEE, 2004.

[89] Dorin Maxim and Liliana Cucu-Grosjean. Response time analysis for fixed-priority tasks with multiple probabilistic parameters. In *RTSS*. IEEE, 2013.

[90] Sibin Mohan, Man Ki Yoon, Rodolfo Pellizzoni, and Rakesh Bobba. Real-time systems security through scheduler constraints. In *ECRTS*. IEEE, 2014.

[91] Sanjay Moulik, Rajesh Devaraj, and Arnab Sarkar. Healers: a heterogeneous energy-aware low-overhead real-time scheduler. *IET Computers & Digital Techniques*, 13(6):470–480, 2019.

[92] Sujay Narayana, Pengcheng Huang, Georgia Giannopoulou, Lothar Thiele, and R Venkatesha Prasad. Exploring energy saving for mixed-criticality systems on multi-cores. In *IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 1–12. IEEE, 2016.

[93] 2017. `http://www.nvidia.com/page/home.html`.

[94] 2013. http://www.hardkernel.com/main/main.php.

[95] 2017. `https://www.openacc.org/`.

[96] OpenMP. Openmp application program interface v4.0., 2013.

[97] John K Ousterhout et al. Scheduling techniques for concurrent systems. In *ICDCS*, volume 82, pages 22–30, 1982.

[98] Santiago Pagani and Jian-Jia Chen. Energy efficient task partitioning based on the single frequency approximation scheme. In *Real-Time Systems Symposium (RTSS), IEEE 34th*, pages 308–318. IEEE, 2013.

[99] Santiago Pagani and Jian-Jia Chen. Energy efficiency analysis for the single frequency approximation (SFA) scheme. *ACM Transactions on Embedded Computing Systems (TECS)*, 13(5s):158, 2014.

[100] 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*, pages 1–10. IEEE, 2014.

[101] Antonio Paolillo, Paul Rodriguez, Nikita Veshchikov, Joël Goossens, and Ben Rodriguez. Quantifying energy consumption for practical fork-join parallelism on an embedded real-time operating system. In *Proceedings of the 24th International Conference on Real-Time Networks and Systems*, pages 329–338. ACM, 2016.

[102] Manar Qamhieh, Frédéric Fauberteau, Laurent George, and Serge Midonnet. Global EDF scheduling of directed acyclic graphs on multiprocessor systems. In *RTNS*. ACM, 2013.

[103] Xuan Qi and Dakai Zhu. Energy efficient block-partitioned multicore processors for parallel applications. *Journal of Computer Science and Technology*, 26(3):418–433, 2011.

[104] Eberle A Rambo and Rolf Ernst. Replica-aware co-scheduling for mixed-criticality. In *ECRTS 2017*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.

[105] Federico Reghenzani, Giuseppe Massari, and William Fornaciari. A probabilistic approach to energy-constrained Mixed-Criticality systems. In *ISLPED*. IEEE, 2019.

[106] Federico Reghenzani, Giuseppe Massari, and William Fornaciari. Probabilistic-WCET reliability: Statistical testing of EVT hypotheses. *Microprocessors and Microsystems*, 77:103–135, 2020.

[107] 2017. https://github.com/scheduler-tools/rt-app/.

[108] 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.

[109] Abusayeed Saifullah, David Ferry, Jing Li, Kunal Agrawal, Chenyang Lu, and Christopher D Gill. Parallel real-time scheduling of DAGs. *IEEE Transactions on Parallel and Distributed Systems*, 25(12):3242–3252, 2014.

[110] Luca Santinelli, Zhishan Guo, and Laurent George. Fault-aware sensitivity analysis for probabilistic real-time systems. In *DFT*. IEEE, 2016.

[111] Youngsoo Shin and Kiyoung Choi. Power conscious fixed priority scheduling for hard real-time systems. In *DAC*. IEEE, 1999.

[112] Sebastian Siebert and Jochen Teizer. Mobile 3d mapping for surveying earthwork projects using an unmanned aerial vehicle (uav) system. *Automation in Construction*, 41:1–14, 2014.

[113] Jelena Spasic, Di Liu, and Todor Stefanov. Energy-efficient mapping of real-time applications on heterogeneous mpsocs using task replication. In *2016 International Conference on Hardware/Software Codesign and System Synthesis (CODES+ ISSS)*. IEEE, 2016.

[114] Marco Spuri and Giorgio Buttazzo. Efficient aperiodic service under earliest deadline scheduling. In *RTSS*, pages 2–11, 1994.

[115] Georgios Stavrinides and Helen Karatza. Scheduling real-time DAGs in heterogeneous clusters by combining imprecise computations and bin packing techniques for the exploitation of schedule holes. *Future Generation Computer Systems*, 28(7):977–988, 2012.

[116] Georgios Stavrinides and Helen Karatza. Scheduling real-time parallel applications in saas clouds in the presence of transient software failures. In *SPECTS*. IEEE, 2016.

[117] Georgios Stavrinides and Helen Karatza. Energy-aware scheduling of real-time workflow applications in clouds utilizing DVFS and approximate computations. In *FiCloud*. IEEE, 2018.

[118] Georgios Stavrinides and Helen Karatza. An energy-efficient, QoS-aware and cost-effective scheduling approach for real-time workflow applications in cloud computing systems utilizing DVFS and approximate computations. *Future Generation Computer Systems*, 96:216–226, 2019.

[119] Ken Tindell, Alan Burns, and Andy Wellings. Mode changes in priority pre-emptively scheduled systems. In *RTSS*. IEEE, 1992.

[120] Sebastian Tobuschat and Rolf Ernst. Efficient latency guarantees for mixed-criticality networks-on-chip. In *RTAS*. IEEE, 2017.

[121] Roman Trüb, Georgia Giannopoulou, Andreas Tretter, and Lothar Thiele. Implementation of partitioned mixed-criticality scheduling on a multi-core platform. *ACM Transactions on Embedded Computing Systems (TECS)*, 16(5s):122, 2017.

[122] Steve Vestal. Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance. In *RTSS*. IEEE, 2007.

[123] Shucai Xiao and Wu-chun Feng. Inter-block GPU communication via fast barrier synchronization. In *IPDPS*. IEEE, 2010.

[124] Huiting Xu, Fanxin Kong, and Qingxu Deng. Energy minimizing for parallel real-time tasks based on level-packing. In *IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, pages 98–103. IEEE, 2012.

[125] Ruibin Xu, Dakai Zhu, Cosmin Rusu, Rami Melhem, and Daniel Mossé. Energy-efficient policies for embedded clusters. In *ACM SIGPLAN Notices*, volume 40, pages 1–10. ACM, 2005.

[126] Kecheng Yang, Ashikahmed Bhuiyan, and Zhishan Guo. F2VD: fluid rates to virtual deadlines for precise mixed-criticality scheduling on a varying-speed processor. In *2020 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, pages 1–9. IEEE, 2020.

[127] Ming Yang, Tanya Amert, Kecheng Yang, Nathan Otterness, James H Anderson, F Donelson Smith, and Shige Wang. Making OpenVX really" real time". In *RTSS*. IEEE, 2018.

[128] Hai-Ying Zhou, Dan-Yan Luo, Yan Gao, and De-Cheng Zuo. Modeling of node energy consumption for wireless sensor networks. *Wireless Sensor Network*, 3(01):18, 2011.

[129] Dakai Zhu, Nevine AbouGhazaleh, Daniel Mossé, and Rami Melhem. Power aware scheduling for and/or graphs in multiprocessor real-time systems. In *Parallel Processing, 2002. Proceedings. International Conference on*, pages 593–601. IEEE, 2002.

[130] Dakai Zhu, Daniel Mosse, and Rami Melhem. Power-aware scheduling for and/or graphs in real-time systems. *IEEE Transactions on Parallel and Distributed Systems*, 15(9):849–864, 2004.