Power-Aware Run-Time Scheduler for Mixed-Criticality Systems on Multi-Core Platform

Behnaz Ranjbar, Tuan D. A. Nguyen, Alireza Ejlali, and Akash Kumar, *Senior Member, IEEE*

***Abstract*—In modern multi-core Mixed-Criticality (MC) systems, a rise in peak power consumption due to parallel execution of tasks with maximum frequency, specially in the overload situation, may lead to thermal issues, which may affect the reliability and timeliness of MC systems. Therefore, managing peak power consumption has become imperative in multi-core MC systems. In this regard, we propose an online peak power and thermal management heuristic for multi-core MC systems. This heuristic reduces the peak power consumption of the system as much as possible during runtime by exploiting dynamic slack and per-cluster Dynamic Voltage and Frequency Scaling (DVFS). Specifically, our approach examines multiple tasks ahead to determine the most appropriate one for slack assignment, that has the most impact on the system peak power and temperature. However, changing the frequency and selecting a proper task for slack assignment and a proper core for task re-mapping at runtime can be time-consuming and may cause deadline violation which is not admissible for high-criticality tasks. Therefore, we analyze and then optimize our run-time scheduler and evaluate it for various platforms. The proposed approach is experimentally validated on the ODROID-XU3 (DVFS-enabled heterogeneous multi-core platform) with various embedded real-time benchmarks. Results show that our heuristic achieves up to 5.25% reduction in system peak power and 20.33% reduction in maximum temperature compared to an existing method while meeting deadline constraints in different criticality modes.**

***Index Terms*—Multi-Core Platform, Mixed-Criticality Systems, Run-Time Management, Dynamic Slack, Timing Overhead.**

I. Introduction

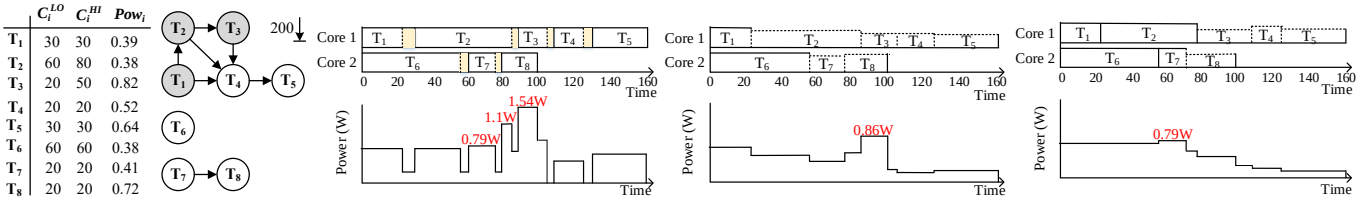
MIXED-CRITICALITY (MC) systems are getting more attention in the last decade due to its significance in safety-critical applications (medical, flight control, etc.). In these applications, tasks are classified into multiple criticality levels in order to maintain the predictability of the applications under different unexpected behaviors [1]–[3]. The criticality of the tasks is based on their importance and functionality to the application. For instance, the Unmanned Air Vehicle (UAV) controller is an example of an MC system [4] shown in Fig. 1a, in which tasks have different criticality levels. In this application, the tasks with higher criticality (HC) (shown by gray color in Fig. 1a) are responsible for the collision avoidance, navigation, and stability of the system. Failure in the execution of these tasks may lead to system failure, and cause irreparable damage to the system. The roles of low-criticality tasks (LC tasks) (shown by white color) are recording sensors data, GPS coordination, and video transmissions, which help the system to carry out its mission successfully. In these MC systems, to guarantee the safety of systems, HC tasks are analyzed with different assumptions, pessimistic and optimistic, in order to obtain different Worst-Case Execution Times (WCETs) for them [5], [6]. If the execution time of at least one HC task exceeds its optimistic WCET, the system switches from low-criticality (LO) to high-criticality (HI) mode. Then, all HC tasks continue their execution by considering the pessimistic WCET to guarantee the safety of the system [3], [4], [7].

In modern sophisticated MC systems, there are a large number of tasks. Therefore, multi-core platforms are utilized to cope with the high demands in performance [8]. However, these platforms require higher power to operate especially when the system switches to the HI mode. If the task scheduler is not aware of the power consumption, all cores might be activated at the same time with the highest performance. Therefore, the system will draw a significantly larger power than it is designed for. Systems with high peak power are more likely to generate unexpected heat that is beyond the cooling capacity. They will be more susceptible to failures and instability [9], which is not acceptable for the HC tasks and it may cause catastrophic consequences. In addition, as the degree of freedom (in terms of the availability of the cores) increases, it is not trivial to guarantee the real-time constraints while managing the system peak power. Therefore, managing the peak power consumption and maximum temperature of the multi-core system, while the deadlines of tasks are guaranteed at runtime, is crucial to be studied. In this work, we target a run-time scheduler to address this problem in the real world.

Motivational Example: To clarify the problem and provide some insight into how a run-time scheduler can manage the peak power consumption, a motivational example is given. Fig. 1a shows a precedence constraint MC task graph with eight tasks mapped on two cores, and tasks' information such as WCETs and peak power consumption. Although, each task in the graph can have a local deadline, the whole task graph has the deadline of $D = 200\ ms$. In addition, in order to simulate the variation in the actual run-time, these values are selected from a uniform distribution of $[\frac{2}{3}.WCET, WCET]$. We obtain the task mapping and scheduling table using the algorithm presented in [4]. In this example, we suppose that the system is only in the LO mode for simplicity of presentation. Besides, we assume that the tasks consume their maximum power continuously during their executions. Fig. 1b shows the task schedule and the system power trace at runtime. In the worst-case scenario, the system's peak power may be high and may lead to thermal hotspots and instability, which has not been

Manuscript received Mar. 11, 2020; revised Jun. 11, & Aug. 22, 2020; accepted Oct. 17, 2020. This manuscript was recommended for publication by Associate Editor Chia-Lin Yang. (*Corresponding author: Akash Kumar)*
This work is supported in part by the German Research Foundation (DFG) within the Cluster of Excellence Center for Advancing Electronics Dresden (CFAED) at the Technische Universität Dresden.

B. Ranjbar and A. Kumar are with Technische Universität Dresden, 01062 Dresden, Germany. e-mail: {behnaz.ranjbar, akash.kumar}@tu-dresden.de
T.D.A. Nguyen is now with Xilinx Research Labs. The work was done when he was with the Department of Processor Design, Technische Universität Dresden, 01062 Dresden, Germany. e-mail: duyatuan@acm.org
A. Ejlali is with the Department of Computer Engineering, Sharif University of Technology, Tehran, Iran. e-mail: ejlali@sharif.edu
B. Ranjbar is also with the Department of Computer Engineering, Sharif University of Technology, Tehran, Iran. e-mail: branjbar@ce.sharif.edu



(a) A Task Graph (b) system power trace at runtime without using DVFS (c) system power trace by using DVFS and considering one task look-ahead (d) system power trace by using DVFS and considering two tasks look-ahead

Fig. 1: A motivational example for a **real-life application** in different scenarios.

investigated in recent studies in MC systems that have different criticality modes. As shown in Fig. 1b, since the tasks may finish earlier than their WCET, for example, $T1$ at $22ms$ while its WCET= $30ms$, the incurred slack can be exploited and assigned to the following tasks to reduce the peak power consumption. In Fig. 1c, these dynamic slacks are used for the immediately ready tasks (one task look ahead) to decrease the speed of its corresponding core. Some power reduction can be observed. However, in some cases, the immediate task that follows may consume much less power than the other tasks after that. Therefore, it is better to reserve that slack to the task after that if it is possible. As shown in Fig. 1d, if we select the task by looking two tasks ahead, more peak power reduction can be achieved as compared to Fig. 1c. Therefore, by comparing the maximum power consumption of two scenarios with using the Dynamic Voltage and Frequency Scaling (DVFS) technique by looking two tasks ahead (Fig. 1d) and without using DVFS (Fig. 1b), we have 48.7% reduction in peak power consumption. In addition, we have 20.12% and 7.94% reduction in energy consumption and peak temperature, respectively.

Proposed Method: In this paper, we propose a heuristic to manage peak power consumption in MC systems during runtime. To achieve this, we exploit dynamic slacks, the slack between tasks' actual completion time and their WCET, along with DVFS. There are two phases in our approach: 1) at design-time, the tasks are scheduled on each core based on the Earliest Deadline First (EDF) algorithm, and the resulting schedule is stored to be used as a static scheduling table. This is performed for both LO and HI modes. In this case, the number of LC tasks that have to be dropped in the HI mode is minimized, in order to improve the overall quality of service (QoS) of the system. 2) at runtime, we examine multiple tasks in the future (look-ahead) to select the most appropriate one to assign the currently available dynamic slack. The selection is based on the impact of the tasks on the peak power and temperature of the system which is quantified by a weighted multi-objective cost function. Therefore, the speed of the core that runs the task can be decreased accordingly using per-cluster DVFS. Additionally, besides exploiting the dynamic slacks, we propose a task re-mapping technique at runtime to further improve the system temperature profile. However, the online scheduler's timing overhead to select an appropriate task and check the re-mapping technique to select an appropriate core are crucial for the MC systems and may cause deadline violations. Furthermore, the timing overhead of changing *V-f* levels in the use of the DVFS technique is critical in run-time task scheduling. Therefore, we analyze and evaluate the effect of these overheads on the schedule of MC tasks in real multi-core platforms. We study that these overheads cannot be neglected due to their impact on meeting MC tasks' deadlines. Besides, we optimize the run-time scheduler to minimize the timing overhead.

Contributions: In summary, the main contributions of this paper are:

- An online peak power and maximum temperature management of MC systems in heterogeneous multi-core platforms while respecting deadline requirements of tasks in both LO and HI modes.
- A multi-task look-ahead approach to make sure that dynamic slacks are assigned to the tasks that lead to more peak power and maximum temperature reduction.
- An online task re-mapping technique that exploits dynamic slacks to re-map the tasks to other cores within a cluster in order to lower the system temperature.
- Studying the online scheduler and DVFS governor in terms of timing overhead to provide the deadline guarantee of MC tasks during run-time phase.
- By measuring on a real platform, we observe that while the latency of the scheduler is minimal (less than 10 μs on average), the latency of the DVFS switching is 5.313ms on average, and thus, cannot be neglected.

Evaluation: We evaluated our run-time scheduler on ODROID XU3/XU4, in which there are four ARM Cortex A7 and four ARM Cortex A15. Experiments show that our method provides peak power reduction, peak temperature reduction and average energy saving up to 5.25%, 20.33% (16.8°C) and 22.44%, respectively, compared to recent studies in the worst-case scenario.

Organization: The rest of the paper is organized as follows. In Section II, we review related works in detail. In Section III, we introduce the models. The problem and our proposed method in detail are presented in Section IV and Section V, respectively. In Section VI, the analysis and optimization of the run-time scheduler are studied. Finally, we analyze and conclude experiments in Sections VII and VIII, respectively.

II. Related Works

Many previous works in the context of MC systems have just focused on proposing techniques in the field of task scheduling and mapping in both online and offline phases. Since our focus is on online MC task scheduling to manage power and temperature, we only consider the works presented for MC or non-MC systems with similar scope. Generally, the related works on power and thermal management for real-time systems can be classified based on the assumed platform, single (S) or multi (M)-core, MC or non-MC systems and the target optimization objectives of peak power, average power or maximum temperature. Table I summarizes the recent works with different target optimization objectives.

TABLE I: Summary of state-of-the-art approaches

#		S/M-Core	Peak Power	Avg. Power	Temp.	DVFS (online/offline)	MC Tasks	DAG Model	DVFS Timing Overhead	Scheduler Timing Overhead	Real Platform
1	Li'14, Huang'14, Li'16, Taherin'18 [10]–[13]	S-Core	×	✓	×	offline	✓	×	×	×	×
2	Awan'16 [8]	M-Core	×	✓	×	offline	✓	×	×	×	×
3	Li'19 [14]	S-Core	×	✓	✓	offline	✓	×	×	×	×
4	Munawar'14 [9], Lee'14 [15]	M-Core	✓	×	×	×	×	×	-	-	×
5	Lee'10 [16]	M-Core	✓	×	×	×	×	✓	-	×	×
6	Ansari'19 [17]	M-Core	✓	✓	×	offline	×	✓	×	-	×
7	Chaturvedi'14, Kabir'16, Bao'09 [18]–[20]	M-Core	×	✓	✓	offline	×	✓	×	×	×
8	Qiu'12 [21]	M-Core	×	✓	✓	offline	×	✓	✓	-	✓
9	Hong'13 [22], Chantem'10 [23]	M-Core	×	✓	✓	offline	×	✓	×	×	×
10	Chen'6, Kang'10, Zhu'03, Singh'13, Zhang'16, Martins'17 [24]–[29]	M-Core	×	✓	×	online	×	✓	×	×	×
11	Chisholm'17, Sigrist'15, Herman'12 [30]–[32]	M-Core	×	×	×	×	✓	×	-	✓	×
12	Trub'17 [33]	M-Core	×	×	×	×	✓	×	-	✓	✓
13	Guo'19 [34]	M-Core	×	✓	×	online	×	✓	✓	✓	✓
14	Our Work	M-Core	✓	✓	✓	online	✓	✓	✓	✓	✓

From the perspective of power and thermal management in MC systems, some works such as [8], [10]–[13] have presented methods to minimize the average power consumption in MC systems theoretically in which systems are single or multi-core (rows 1 and 2 in Table I). In general, they only optimize the average power in the LO mode in simulation. When the system switches to the HI mode, all HC tasks are executed with the highest frequency; and all LC tasks are dropped. As a result, in the HI mode, with higher frequency, the peak power consumption of the system may increase significantly, which is not admissible. Furthermore, there is a paper [14] that has considered thermal management in MC systems (third row of the Table I). The researchers minimize the temperature of single-core processors by finding the optimum speed for each task in the design-time phase. Hence, they discard LC tasks when the system switches to the HI mode, which is not acceptable in many MC applications. Besides, they do not consider the latency of changing the *V-f* level at runtime, which may cause deadline violation and, consequently, catastrophic consequences.

As we focus on peak power management, some studies concentrate on peak power management in multi-core systems at design-time (rows 4-6). These papers have only considered hard real-time tasks with one criticality level which is not practical for MC. It should be mentioned that authors in [16] work on the dependent task model in which the execution of some tasks is postponed to manage the simultaneous peak power consumption. It is not suitable for MC tasks, especially in the HI mode.

The previous works in the context of power or thermal management in non-MC systems that use DVFS by considering the dependent task model are shown in rows (7-10) of Table I. Researchers in [35] give a comprehensive study in the field of energy and thermal management of multi-core platforms. Since our approach is reducing the power and temperature at runtime, we review the works that have used the online DVFS, which are [20], [24]–[29]. In [20], a look-up table for each task is generated in the offline phase, which contains the optimum voltage and frequency settings for each core for every possible run-time condition, task execution time, and core temperature measurement. The memory overhead incurred in generating these tables may not be desirable, especially for multi-core systems with many tasks and cores. Researchers in [24]–[29] have used slack reclamation to apply online DVFS to the system while executing dependent tasks. Kang et al. [25] propose an algorithm that uses dynamic voltage scaling to minimize energy without considering the tasks' deadlines, which is not suitable for MC systems. Researchers in [26]–[28] suggest a run-time energy management technique that uses reclaimable slack for the immediately ready task to decrease average power. Their results show that the power can be reduced; however, the possibilities of looking further ahead into the future execution of the following tasks to have better results are not explored. Besides, in [29], the authors have considered two types of tasks, best effort and real-time, and they have just used the dynamic slack for the next real-time task to reduce its *V-f* level, which is inefficient. There is an aggressive slack reclamation algorithm, presented by [24], in which, the generated dynamic slack is checked to be able to use for the next task if the remaining tasks could complete their execution before the deadline. However, in general, the average energy consumption is reduced, but this algorithm has focused more on meeting the deadlines, while we target both energy minimization and meeting the deadlines.

Most of the related works that we discussed here, have evaluated their method in a simulation, and just focused on power or temperature minimization, and are not concerned about run-time behaviour and its associated timing overhead. On the other hand, there are some previous works in the context of MC systems that just focus on considering timing overhead of scheduler [30]–[33], memory sharing and bus communication latency [36]–[39] to evaluate their method and have a real implementation. Since our focus is on online MC task scheduling, we only consider works that have studied the scheduling latency. Most of the papers, which have considered the latency of the scheduler in the field of MC systems (row 11-12 in Table I), are limited to the timing overhead of task monitoring, task termination and arriving and mode switch-

ing, not the scheduler. Further, these run-time overheads are measured on the Intel platform, i.e., not embedded processors.

In addition, from the DVFS latency perspective, some few works, e.g., [34], have presented a method to minimize energy in a multi-core platform by using the DVFS technique. Researchers in [34] have considered a task graph model running on the cluster-based platform. They have also considered the latency of changing frequency in their paper. As shown in Table I row 13, they have not considered peak power or thermal management and also, their method is not suitable for MC systems where tasks have different criticality levels.

We study run-time scheduling of dependent MC tasks, which are executed on a multi-core platform to manage peak power and maximum temperature by considering the timing overheads such as frequency changing and run-time scheduler, which is not found in existing MC works.

III. System Models

A. Task Model

We consider real-time applications consisting of dependent periodic MC tasks, such that, each task τ_i is represented as $\{\zeta_i, C_i^{LO}, C_i^{HI}, d_i, Su_i, Pr_i\}$. Analogous to [4], [7], we consider dual-criticality system where each dependent MC task can be either high-critical (ζ_i = HC) or low-critical (ζ_i = LC). Further, each task τ_i has a deadline d_i. The successors and predecessors of each task are determined by Su_i and Pr_i, respectively. A task can be executed after all its predecessor tasks have finished their execution. Each MC task has different WCETs, C_i^{LO} (optimistic) and C_i^{HI} (pessimistic) that for each LC task $C_i^{LO} = C_i^{HI}$ and also, for each HC task $C_i^{LO} \leq C_i^{HI}$. If a task is a predecessor of an HC task, then it is considered as an HC task as well. In addition, all tasks have a common period P which is the period of the task graph.

In general, MC systems have two modes of operation: LO and HI. Initially, the system starts in the LO mode in which all LC and HC tasks must be executed correctly before their deadlines. When the execution time of at least one HC task exceeds its C_i^{LO} due to unexpected conditions, the system switches to the HI mode and then all HC tasks are executed with their C_i^{HI}. In the dependent MC task model, the system switches back safely to the LO mode at the end of each period [4], [7].

B. Hardware Architecture Model

We consider a multi-core processor comprising of m cores $\{C_1, C_2, ..., C_m\}$ based on the ODROID XU3 board, where the system is DVFS-enabled and the cores can operate at multiple voltage (V) and frequency (f) levels. The ODROID XU3 consists of two clusters with ARM cortex-A15 (big) and ARM Cortex-A7 (LITTLE) (four big cores and four LITTLE cores); hence, cores within the same cluster operate at the same *V-f* level and also, each cluster can operate at different frequency and voltage levels. In this board, the allowed frequency is in the range of [0.2, 1.4] *GHz* for LITTLE cores and [0.2, 2] *GHz* for big cores. Besides, the voltage is in the range of [0.9, 1.3] *V* for LITTLE cores and [0.9, 1.3625] *V* for big cores.

C. Power Model

The total power consumption of a core is composed of static (P_s), dynamic (P_d) and independent power consumption (P_{ind}) [12], [13]. P_{ind} refers to the power related to the memory and I/O activities. As mentioned in Section III-B, the *V-f* level of an entire cluster can be changed. This implies that all cores in a cluster must have the same *V-f* level. The total power consumption is given by Eq. (1). In this equation, I_{sub} and C_L are the sub-threshold leakage current and load capacitance, respectively. In this paper, we focus on decreasing P_d.

$$P = P_s + P_d + P_{ind} = I_{sub}V + C_L V^2 f + P_{ind} \quad (1)$$

in which: (ρ_1 and ρ_2 are the scaling factors of frequency and voltage, respectively)

$$f_{min} \leq f = \rho_1 \times f_{max} \leq f_{max}, V_{min} \leq V = \rho_2 \times V_{max} \leq V_{max}$$

Therefore, by using these scaling factors, Eq. 1 can be written based on the V_{max} and f_{max} as:

$$P = I_{sub}(\rho_2 V_{max}) + C_L(\rho_2 V_{max})^2(\rho_1 f_{max}) + P_{ind} \quad (2)$$

As our system is based on the ODROID XU3, some frequency levels work with the same voltage level on this board. It means, by reducing the frequency level, the voltage level does not change. Therefore, the scaling factors ρ_1 and ρ_2 do not have the same value. According to the range of frequency for big and LITTLE cores presented in Section III-B, ρ_1 can be set in the range of [0.143, 1] for the A7 cores and [0.1, 1] for A15 cores. In addition, ρ_2 is in the range of [0.692, 1] and [0.6606, 1] for A7 and A15 cores, respectively. Although the ODROID XU3 has power sensors, they only report values for the entire cluster, not for each core.

IV. Research Objectives

We target peak power consumption and maximum temperature issues in MC systems and evaluate the algorithm on a real multi-core platform. Although there are works that manage or minimize the power consumption of MC systems as previously discussed in Section II, they have not considered the instantaneous peak power consumption in both HI and LO modes and their algorithms have often been limited to simulation. One of the most common approaches to solve the problem is to exploit the dynamic slack generated at runtime to change *V-f* levels of cores, while the MC tasks' deadlines are guaranteed. The crucial research questions that are addressed in our article are as follows: (1) How to select the most appropriate tasks to assign the dynamic slack to, for managing the peak power consumption; (2) Whether it is possible to re-map the tasks to other cores for better thermal control, and if yes, where and when should the tasks be re-mapped to; (3) Which timing overheads during runtime have an impact on task scheduling and deadline misses; (4) How these run-time timing overheads can be managed to not affect tasks' deadlines.

V. Proposed Method: Online Peak Power and Max. Temperature Management Method in MC Systems

The goal of our proposed method is to minimize the peak power and the maximum temperature of individual cores during run-time.

$$Minimize(P_j, (T_{max})_j)|_{(j \in Cores)} \quad (3)$$

DVFS is one of the techniques that we use to manage the metrics (peak power and maximum temperature). Reducing

the *V-f* level of a core while executing a task, increase the execution time of the task and it may cause deadline violation. In addition, the latency of changing *V-f* level or run-time scheduling may cause deadline violation. Eq. 4 represents that the sum of the execution time of each task i on the core j at the *V-f* level l and timing overheads of the run-time scheduler ($TO_{Sch.}$) and changing *V-f* level (TO_{Vf}) must not exceed the task deadline in each criticality mode.

$$TO_{Sch.} + TO_{Vf} + \frac{C_i}{f_{jl}} \leq d_i \rightarrow \begin{cases} C_i = C_i^{LO} & \text{if } mode = LO \\ C_i = C_i^{HI} & \text{if } mode = HI \end{cases} \quad (4)$$

The proposed approach consists of design-time and run-time phases. It is worth noting that the proposed method takes advantages of the run-time phase to manage the peak power and temperature; hence it is not possible to use any optimization method such as ILP (Integer Linear Programming) due to its long execution time. Thus, we develop a heuristic-based method. Fig. 2 shows the flow of our proposed approach, along with the Hardware Platform. The Hardware Platform is used in Design-Time Phase for tasks' power profiling and in Run-Time Phase for execution of task on cores. Now, we explain our approach comprising of the Design-Time and Run-Time Phases, in detail.

A. Design-Time Phase

The input to the algorithm is a precedence constrained task set and the multi-core system description, as shown in Fig. 2. The power required by the tasks can be obtained by running the benchmarks on a real platform, which is discussed in detail in Section VII. It should be noted that handling an unknown application during runtime is beyond the scope of this paper. Since we target embedded applications, normally, the designer knows the system's tasks and their parameters at design-time. Therefore, by using the parameters of MC tasks such as WCETs, two tables of static task mapping and scheduling for LO and HI modes are created as shown in Design-Time Phase of Fig. 2. EDF algorithm is used to calculate the schedule of the tasks in each of the two modes statically based on the WCETs of LC and HC tasks, using the algorithm presented in [4]. In the LO mode, all tasks are scheduled with equal priority; in the HI mode, HC tasks are scheduled with a higher priority. These static schedules in the respective modes are then used to execute all tasks at run-time. This enforces a strict ordering in the execution of the tasks and guarantees that all deadlines are met according to the design-time analysis in both modes. It should be noted that since the WCETs of HC tasks are higher in the HI mode, not all LC tasks may be schedulable in the HI mode. In order to maximize the overall QoS, the algorithm tries to drop as few LC tasks as possible when computing the HI mode table. These tables and the info associated with the tasks are used during run-time phase by our algorithm to manage the system.

B. Run-Time Phase

The run-time phase of our proposed method consists of several function control units, as shown in Fig. 2. The Scheduler Unit is the main unit that is communicating with the other units. Two main functions are supported in this unit: 1) Execute the tasks according to the tables; 2) Change the

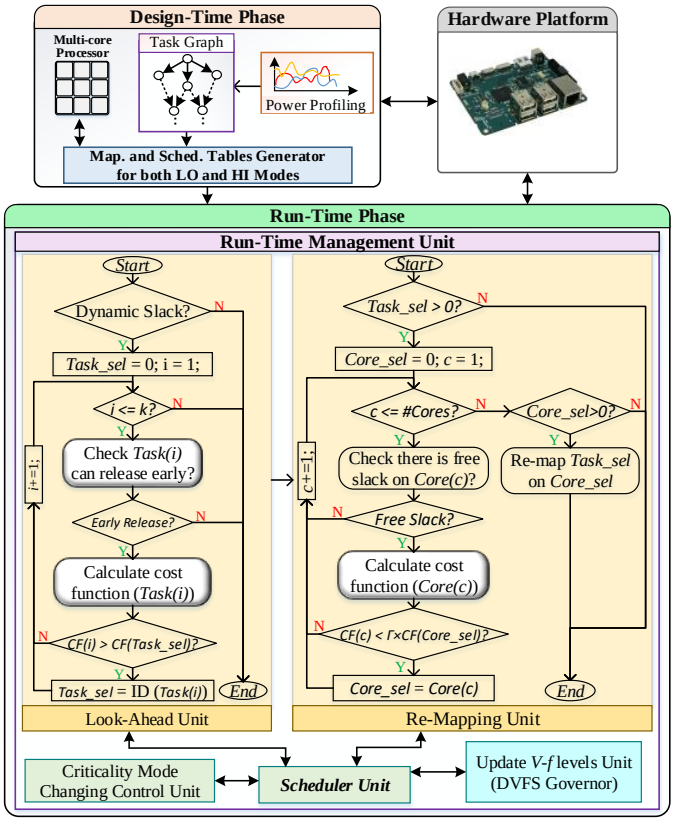


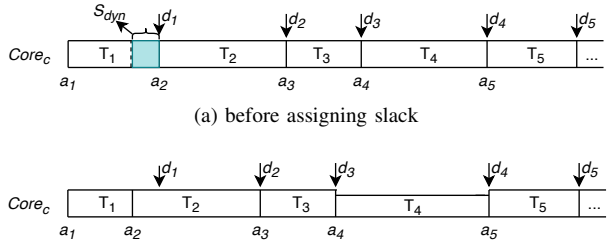
Fig. 2: Overview of our proposed approach.

scheduling and mapping of the tasks according to our proposed policy which we discuss in Sections V-B1 and V-B2. When there is any free slack on a core, or a task finishes its execution early, the Look-Ahead Unit is executed. This unit is used to choose a subset of tasks and select the most appropriate one among them. If an appropriate task is selected in a core, according to the core temperature and temperature of other cores, the Re-Mapping Unit is used to reduce the maximum temperature and decide whether to re-map the task to other cores or not. After that, the obtained *V-f* level for the core is stored. This stored frequency is used by the DVFS governor Unit when the task is ready to be executed. The details of the DVFS Governor Unit to select the optimum *V-f* level for a cluster is discussed in Section V-D. Due to MC systems' behavior, the system switches to the HI mode if the execution of at least one HC task exceeds its defined C^{LO}. It should be checked by the Criticality Mode Changing Control Unit presented in Fig. 2. In this case, the system changes its task scheduling according to the HI scheduling table which is generated at design-time. The details for Look-Ahead Unit and Re-mapping Unit are described as follows.

*1) **Selecting the Appropriate Task to Assign Slack**:* In Look-Ahead Unit, we consider an approach named look-ahead in which our algorithm chooses k tasks after generated dynamic slack and also mapped on the same core in which the dynamic slack (S_{dyn}) is generated[1]. For each of the k tasks, a cost function is computed as defined by Eq. 5 below.

$$CF_i = \alpha \times E_i + \beta \times Pow_i \quad (5)$$

[1]Finding the optimum value for k is discussed in Section VII.



(a) before assigning slack

(b) after assigning slack

Fig. 3: An example of look-ahead policy

In this function, Pow_i and E_i are the maximum instantaneous power and maximum energy, respectively, that a task consumes to execute. In addition, α and β are in the range of [0,1]. Besides, energy reduction leads to a decrease in chip temperature [11]. Note that, if we consider $\langle \alpha,\ \beta \rangle = \langle 0, 1 \rangle$, the cost function only considers power of a task, and not it's energy. Hence, the task with the largest peak power consumption is chosen to be executed at reduced core speed, in order to reduce the peak power consumption. If we have $\langle \alpha,\ \beta \rangle = \langle 1, 0 \rangle$, only energy is considered in the cost function. Hence, the task with the largest energy consumption is chosen to be executed at reduced core speed, thereby reducing the maximum energy consumption (Appendix A shows the proof of optimal solution of peak power minimization in individual cores). After selecting the task, the maximum power consumption, and its WCET (C_i^{LO} or C_i^{HI}) are changed based on the size of generated slack time and the *V-f* level. As a result, the start time and the deadline of tasks that are executed between the generated dynamic slack and selected task are *shifted* left based on the amount of slack to let the chosen task run with less speed, for example tasks T_2 and T_3 illustrated in Fig. 3.

Furthermore, Eq. 5 is applied to a set of tasks that can start their executions earlier. A task (τ_i) can start early if it is released before $a_i - S_{dyn}$, where a_i is the start time of τ_i. As mentioned, a task can be released when all its predecessors finish their execution. Therefore, we just check $T_{ri} \leq\ a_i - S_{dyn}$, where T_{ri} is the release time of τ_i. Consider the selected task τ_i with the start time a_i and deadline d_i that $a_i + WCET_i \leq\ d_i$. Assuming that we have the amount of slack, S_{dyn} generated by τ_j, during runtime. To utilize this slack time for the appropriate task τ_i, in general, the scheduler finds the minimum acceptable frequency based on $f_i = max(f_{min}, \frac{WCET_i}{WCET_i + S_{dyn}}.f_{max})$. This ensures that only the start time of the task is earlier by S_{dyn} and the deadline is kept unchanged, for example T_4 shown in Fig. 3. Hence, $a_i - S_{dyn} + \frac{WCET_i}{f_i / f_{max}} \leq\ a_i + WCET_i \leq d_i$. However, as mentioned at the beginning of this section, selecting the proper task and the core, and changing the *V-f* level have overheads[2]. If we ignore them while selecting the optimum frequency, it may cause a deadline violation. Therefore, S_{dyn} is reduced by $TO_{Sch.}$ and TO_{Vf}. After selecting the optimum frequency the start time of the appropriate task τ_i (a_i) is updated for the static schedule.

*2) **Re-Mapping Technique**:* In order to manage the maximum temperature of the system and have better thermal control, it is possible to re-map the selected task to the other cores without changing its deadline. Therefore, to decide about re-mapping the task and selecting the appropriate core to re-map, we use the cost function in Eq. 6.

$$CF_c = \Gamma \times \sum_{t=1}^{t_f} E_c(t) \tag{6}$$

In this cost function, instead of using actual core temperature, we predict their temperature according to the accumulated energy. Based on our observation, a core tends to have a lower temperature when its accumulated energy is less than the other cores[3]. However, the difference between the accumulated energy of the base core and the selected core should be large enough. Therefore, we define a coefficient (Γ), which is equal to 0.9 in our experiments. In this equation, t_f is the time when any particular task is finished. Besides, in order to not affect the tasks' deadline mapped on other cores, cores are examined for re-mapping that have free slack at the same period to execute the appropriate task. Since we consider the clustered multi-core platform (ODROID XU3) for our experiment, each application's execution time and power consumption will be different when running on different clusters. Hence, we use the re-mapping technique within each cluster. The reason is that although re-mapping from a little core to a big core reduces a task's execution time, it causes the system peak power consumption to increase, which is not acceptable based on our targets. Therefore, to not change the system peak power consumption, we use the re-mapping technique within each cluster. It should be noted, since the re-mapping technique is applied to a task that is not started yet, and also, the technique is done in parallel with changing the frequency, the migration overhead does not affect the deadline constraints. The reason is that the latency of re-mapping is much less than the latency of changing the frequency, which we study in detail in Section VII.

C. Run-Time Management Algorithm

The pseudo-code of our proposed algorithm is outlined in Algorithm 1. At first, the algorithm gets the set of precedence constraint tasks, the number of tasks looking ahead (k), scheduling table for each mode, and available *V-f* levels for cores as inputs. Then it gives start time and the *V-f* level assignment for each task at runtime. At the initialization step, the system starts its operation in the LO mode, and also, the voltage and frequency of each core are set to the maximum value (lines 1-3). The proposed online peak power reduction algorithm is presented in (lines 4-45). At first, the algorithm checks that whenever the execution time of a task exceeds its *WCET*, the system switches to the HI mode (lines 5-9). If any task execution exceeds its C_i^{LO} and the output of this task is not ready, the system switches to the HI mode and remains in this mode till the end of the period. In this situation, at the beginning, the *V-f* level of each core is set to the maximum value to meet the deadline of HC tasks (lines 7-8). The rest of the algorithm is executed in both modes.

If there is a dynamic slack during runtime, the algorithm selects the appropriate task to assign slack, which has more impact on instantaneous power consumption (lines 10-39). This dynamic slack is generated if a task finishes its execution

[2]We discuss in Section VII, how theses timing overheads ($TO_{Sch.}$ and TO_{Vf}) are measured.

[3]We show the observation about the relevance between core temperature and its accumulated energy consumption in Section VII

Algorithm 1 Online Peak Power Reduction Algorithm

Input: Task Graph (G_T), Cores, Scheduling Tables of each Mode (Sch_L and Sch_H), Number of Tasks Looking Ahead (k).

1: mode ← 0 , MO ← LO; // the system starts from the LO mode and Sch_L is used to schedule the tasks
2: **for each** core j **do** initialize the *V-f* level to maximum;
3: **end for**
4: **procedure** MCS ONLINE PPREDUCTION
5: **if** each Task executes more than C^{MO} **then**
6: mode ← 1, MO ← HI; // System switches to the HI mode and task scheduling is done by Sch_H
7: **for each** core j **do** initialize the *V-f* level to maximum;
8: **end for**
9: **end if**
10: **if** each Task finishes its execution earlier than its deadline **or** there is an idle time in a core **then**
11: **if** $Task_i$ has already finished its execution **then**
12: S_{dyn} ← Extract_Dynamic_Slack(); //C_i^{MO} - ACT_i
13: **elseif** there is an idle time in a core **then**
14: S_{dyn} ← Extract_Dynamic_Slack(); //idle time
15: **end if**
16: T_S, T_P ← 0
17: S_{dyn} -= $TO_{sch} + TO_{Vf}$;
18: **for** n = 1 **to** k **do**
19: T_P ← $\tau_{n_{th}}$ after generated slack;
20: **if** $CF_{T_S} < CF_{T_P}$ **and** T_P can start earlier **then**
21: T_S ← T_P, n_s ← n;
22: **end if**
23: **end for**
24: **if** $n_s > 0$ **then**
25: $Freq_{T_S}^{MO}$ ← max (f_{min}, $\frac{C_{T_S}^{MO}}{C_{T_S}^{MO}+S_{dyn}}$)
26: **for** n = 1 **to** n_s **do**
27: *Update* the $St_{Task_{LA-n}}^{MO}$ & $d_{Task_{LA-n}}^{MO}$
28: **end for**
29: /*Re-Mapping Checking*/
30: $Core_S$ ← $Core_{T_S}$, $Flag_{remap}$ ← 0
31: **for** each core j in the cluster **do**
32: **if** $CF_j < \Gamma \times CF_{Core_S}$ **and** free slack exists **then**
33: $Core_S$ ← C_j, $Flag_{remap}$ ← 1;
34: **end if**
35: **end for**
36: **if** $Flag_{remap}$ == 1 **then** Re-Map (T_S, $Core_S$);
37: **end if**
38: **end if**
39: **end if**
40: **for each** task i **do**
41: **if** St_i^{MO} == $Time_{sys}$ or a task finishes its execution **then**
42: DVFS(Ready and Running Tasks, Cores); //Update cluster *V-f* level (Algorithm 2)
43: **end if**
44: **end for**
45: **end procedure**

Algorithm 2 DVFS governor

1: **function** DVFS(Ready and Running Tasks, Cores)
2: **if** $Core_{Task_i} \leq 3$ **then** C_{ID} = 0; //Cluster with LITTLE cores
3: **else** C_{ID} = 4; //Cluster with big cores
4: **end if**
5: SetFreq = 0;
6: **for** c = C_{ID} **to** C_{ID}+3 **do**
7: **if** $SetFreq < Freq_{Run/ReadyTaskonCore_{C_{ID}}}^{MO}$ **then**
8: SetFreq = $Freq_{Run/ReadyTaskonCore_{C_{ID}}}^{MO}$;
9: **end if**
10: **end for**
11: **if** $SetFreq$! = $Freq_{Cluster}$ **then**
12: `cpufreq-set -c` $Core_{Task_i}$ `-f` $SetFreq$
13: **end if**
14: **end function**

before its defined WCET (C_i^{LO} or C_i^{HI} due to the system mode). In addition, since we use static scheduling of tasks for both modes and do not change the order of task execution in each core, there may be some idle time in a core that can be used. Therefore, if there is dynamic slack, we first compute the amount of available slack (lines 11-15). Hence, we have to consider the timing overheads of the scheduler and speed changing. Therefore we deduct these latencies from the slack to guarantee the deadline (line 17). Now, we select the appropriate task among k tasks that can be released early due to the slack time after reclaimable slack (lines 18-23) based on the cost function (Eq. 5). Besides, Fig. 2 details this process in the flow chart. After determining the appropriate task, according to the system mode situation, the frequency of the core to execute the appropriate task is obtained according to the amount of slack (line 25). Hence, the selected frequency must be rounded to the nearest *V-f* level of the cluster that is greater. If there is at least one task between the generated slack and selected task, we change their start time. Therefore, their deadline would be changed (lines 26-28). Now, the re-mapping technique is applied if the core in which the task has been allocated, has a higher temperature than other cores (lines 29-37). As a result, it is possible to re-map the selected task to a core according to cost function (Eq. 6). As mentioned in Section V-B, we just re-map a task between the cores of each cluster. Further, the algorithm checks regularly that if a task is ready to start based on the static schedules, the *V-f* level of the core that task has been mapped on it, is changed according to defined frequency scaling factor (lines 40-44). The detail is discussed in the following sub-section.

D. Update V-f Levels in Clustered Multi-Core Platform (DVFS governor)

After finishing a task execution, there might be a free slack or a task in the core queue that is ready to start its execution. Here, Algorithm 2 is executed to change the *V-f* level if needed. As mentioned, all cores within a cluster operate at the same *V-f* level in clustered multi-core platforms. Since the *V-f* levels of both clusters are different, it is checked on which cluster the recently completed task was running (lines 2-4). Then, we check the assigned *V-f* level of running or ready tasks on all cores of the cluster. Since all cores run with the same speed, we find the best frequency to set to the frequency cluster (lines 5-10). The reason for selecting the greatest minimum frequency is to ensure that all tasks finish their execution before their deadline. In the end, if the chosen frequency (*SetFreq*) is different from cluster frequency, we change the speed of the cluster by assigning the new speed to one core of the cluster by using ⟨cpufreq-set⟩ program (lines 11-13). It should be mentioned that by changing the frequency of a cluster, its voltage will be changed automatically based on the table setting of the kernel.

VI. RUN-TIME SCHEDULER ALGORITHM OPTIMIZATION–ANALYSIS AND IMPLEMENTATION

In the presented approach of our previous paper [6], the timing overheads of run-time scheduling and changing the frequency have not been considered. However, these overheads

TABLE II: Run-time scheduler cache misses report

	Look-Ahead Unit			Re-mapping Unit		
	Cortex A7	**Cortex A15**	**Intel Core i5**	**Cortex A7**	**Cortex A15**	**Intel Core i5**
L1 Data Read Miss	3.318%	3.318%	2.946%	0.163%	0.163%	0.303%
L1 Data Write Miss	3.584%	3.584%	2.168%	0.352%	0.352%	0.764%
LL Data Read Miss	0.076%	0.081%	0.0%	0.028%	0.034%	0.0%
LL Data Write Miss	0.063%	0.0%	0.0%	0.036%	0.0%	0.0%

can have a profound impact on power-aware run-time scheduling of tasks and must be considered in the respective analysis. Neglecting them may lead to missing deadlines for MC tasks, which may cause catastrophic consequences. Two sources of generating overheads that deal with the online scheduler are the Look-Ahead Unit to select the appropriate tasks and the Re-mapping Unit to find the appropriate core. In the following, we use the online scheduler phrase for both units to make it easy to follow. The other source of causing overhead is the DVFS governor Unit for changing frequency during runtime. Now, we first analyze the scheduler function from the timing overhead aspect. Then, we focus on optimizing the code and reducing the overheads.

In order to evaluate the scheduler and analyze the overhead on a real platform, we first convert Matlab code to C code. Then, we detect the main parts of the code, which have more latency and attempt to optimize it. To analyze the main functions, we first get a strict upper bound of the latency in different parts of the online scheduler on a real platform. We use the Kcashegrind tool [40] to measure the worst-case time. Kcashegrind is a visualization tool that uses a technique called profiling, which gives you the time distribution among the scheduler code at runtime. Now, we focus on the functions code and its timing analysis and endeavor to reduce the estimation cycles and the delay caused by cache misses in shared cache levels. Both Look-Ahead and Re-mapping unites in the online scheduler are called frequently during runtime, and the apparent improvement and optimization should be performed. The run-time phase of Fig. 2 shows the flow chart of these two units in detail. As shown in this figure, some functions play a critical role in timing overhead of power-aware run-time scheduler, which is indicated by white color. As discussed in the previous section, the Look-Ahead unit chooses k tasks after generated dynamic slack and find a task that has the most effect on peak power and maximum temperature. Checking the k tasks is done in a for-loop, in which each task is investigated that can release early to use the dynamic slack. Therefore, all predecessors of it must be checked whether they can finish their execution soon or not. Investigating the execution status of all predecessors need more cycles to be done and then causes latency and more cache misses. Therefore, introducing an entity that shows the estimated finish time of a task would be useful, and instead of checking the status of all predecessors, just that entity can be checked. Besides, due to the having different *V-f* levels, we must ensure that the dynamic slack is large enough to include the timing overhead of changing the *V-f* level. This check prevents the over-calling of the Re-mapping Unit. In addition, there are two functions of calculating the costs, in which there are some math calculations with high timing overhead. Hence, optimizing these calculations by pre-defining them to avoid dynamic memory allocation during computation would help reduce timing overhead.

From the perspective of cache hit/miss, one of the ideas is to optimize the code to reduce the estimation cycles in the online scheduler by focusing on calculations and memory access latency. There are some tips to optimize C/C++ code to run it faster; reducing functions calls and the number of function parameters, how to define variables and objects, how to use operators, using prefix instead of postfix in objects, avoiding unnecessary data initialization and so many other techniques that we must use for code optimization. Apart from using these techniques, due to data access latency, we have effective timing overhead in the online scheduler. We optimize code by changing the representation of the data structure manipulated by the algorithms. We have defined two types of task classes: 1) defining a task class that uses vectors in the class for each task entity, 2) defining a task class with vectors of task class in the number of tasks. Each has its advantages and disadvantages under certain circumstances. However, due to the checking of limited tasks (k) in the run-time scheduler, the use of the second task class has less timing overhead, and cache misses. As a result, Table II shows the percent of cache (L1 and LL (last level)) read and write misses for Look-Ahead and Re-mapping Units after optimization on three different platforms, Cortex A7, Cortex A15, and Intel Core i5. As mentioned in previous sections, most of the embedded systems use ARM processors, not Intel. Therefore, we target the ARM processors, such as the ODROID board. However, this table shows that we have less than 3.584% and 0.081% cache L1 and LL data misses in ARM processors, respectively, which are admissible compared to all cache misses and also in comparison with Intel processor that has less cache misses.

VII. Evaluation

A. Experimental Setup

1) Hardware Platform: To evaluate our system, we conducted experiments on the ODROID XU3/XU4 board powered by ARM, which has big.LITTLE architecture, four big (Cortex A15), and four LITTLE (Cortex A7) cores. As explained in Section III-B, the ODROID XU3 board supports DVFS and can operate at 13 different *V-f* levels between $[0.9V, 200MHz]$ and $[1.3V, 1.4GHz]$ on LITTLE cores, while the last four frequency levels have the same voltage levels and 19 different *V-f* levels between $[0.9V, 200MHz]$ and $[1.3625V, 2GHz]$ on big cores. Therefore, the effect of changing *V-f* levels is done by scaling the frequency within the range of available levels.

2) Task Set Generation: In the experiments, we use random applications (task graphs) generated by the tool in [4]. An example of a real-life application is already given in the motivational example. In these applications, there are four basic parameters, c (number of cores), U (system utilization), d (outgoing edge percentage) and n (number of tasks), which are presented in Table III. d represents the probability of having outward edges from one task to the others. In addition, U/c is

TABLE III: Experiment Configurations

Param.	Varying c	Varying U/c	Varying n	Varying d
c (#core)	2, 4, 8, 16	8	8	8
U/c (utilization)	[0.5, 0.75]	[0, 1]	[0.5, 0.75]	[0.5, 0.75]
d (edge percentage)	10%	10%	10%	1%, 10%, 20%
n (#task)	50	50	30, 40, 50, 80	50

the normalized system utilization that refers to both LC and HC tasks with their predefined C^{HI}. As the results are presented in both simulation and real platform (with eight cores), we show the results with 16 cores in simulation in addition to 2, 4 and 8 cores. We provide different configurations by changing the value of these parameters for different scenarios used in the experiments.

3) Tasks' Power Consumption: In order to have a realistic possible range of power values, we run several embedded benchmarks from MiBench suite [41], e.g., automotive, network and Telecomm benchmarks on two configurations, ARM Cortex A7 and A15 on the ODROID XU3 with maximum frequency and read data from power sensors on the board. Hence, since the DVFS is applied to the whole processor, the power consumption at other lower frequencies can be obtained using Eq. 2 in Section III-C by considering frequency scaling [42]. In addition, we examined different scenarios of activating one core to all cores by running different benchmarks. We run each benchmark 1000 times and report the maximum value of power consumption. We select the maximum power of tasks in the range of these minimum and maximum values in our experiments, which is [484, 940]mW in Cortex A7 and [3.891, 7.622]W in Cortex A15. The power that the tasks may consume is generated randomly following the normal distribution within this range. Besides, we consider the power consumption of the system as the sum of the power consumption of all cores [9].

4) Thermal Analysis: As presented in the proposed method section, we assume that our approach does not have to probe the core temperature to make a decision. Therefore, during the scheduling of the tasks, the power values of cores depending on the running tasks are recorded. In addition, for validating on the real platform, since there are just temperature sensors for big cores on the ODROID XU3, the HOTSPOT tool [43] is used to obtain the core temperature throughout the execution for the specific floorplan and configuration platform which we use. For the configuration file, we use the parameters reported in [44], which is for ARM big.LITTLE processors. The ARM core (A7) has an area of 0.45 mm^2 in our experiments reported by the ARM company.

5) Comparison: In this paper, we analyze our modified proposed method and compare against [4], [26], and the previous work, [6]. The work [4] proposes an offline scheduling algorithm for an MC system where most of the LC tasks are not dropped in the HI mode to improve the QoS of the system. However, they ignore the peak power and temperature aspect of the system. Additionally, researchers in [26] suggest an online energy minimization algorithm for hard real-time systems where they use the dynamic slack just for the immediately available task to decrease the *V-f* level. We compare with the method of this paper by considering the latency to have a fair comparison.

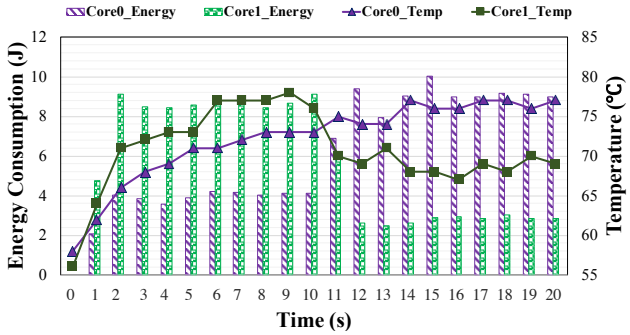
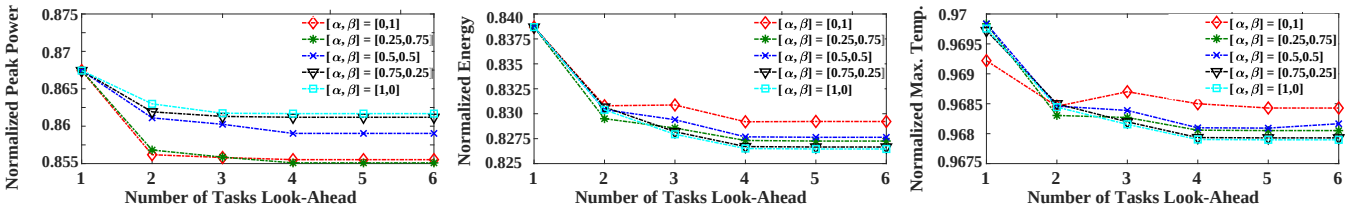


Fig. 4: The relevance between a core temperature and accumulated energy consumption.

B. Experimental Results, Observation and Discussion

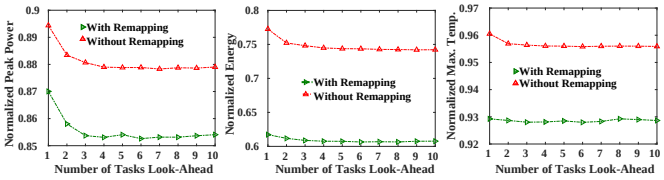
*1) **Analyzing the relevance between a core temperature and energy consumption**:* At first, we represent the relevance between core temperature and its accumulated energy consumption. In Section V-B2, our algorithm was based on the assumption that a core tends to have a lower temperature when its accumulated energy is less than the other cores. Fig. 4 studies the validity of the assumption. Since we do not have a power sensor for each big core on the ODROID XU3, we run the same task on all the big cores to have the same power consumption. This task is executed several times periodically in cores with different execution times. Therefore, we have different energy consumption in each period of cores. After finishing the execution of the task on each core, the core goes to sleep until the end of the task's period. Fig. 4 shows energy consumption and the temperature of two big cores during the time for a window of energy monitoring equal to two seconds. In this figure, first, the task runs with larger execution time on Core1 in comparison to Core0. Thus, the temperature of Core1 rises more rapidly than Core0. After 10s, the accumulated energy of Core1 is reduced, and Core0 is increased. As shown, the Core0 that has more energy consumption tends to have a higher temperature.

*2) **The effect of varying** $\langle\alpha,\beta\rangle$:* Now, we evaluate the results for different values of α and β in Eq. 5. The experiments are carried out for a system with $c = 8$, $U/c \in [0.5, 0.75]$, $d = 1\%$ and $n = 30$. The average results (Fig. 5) are obtained for a set of 100 task graphs with different $\langle\alpha,\beta\rangle$ = $\langle 0,1\rangle$, $\langle 0.25, 0.75\rangle$, $\langle 0.5, 0.5\rangle$, $\langle 0.75, 0.25\rangle$ and $\langle 1, 0\rangle$. The results are normalized to [4]. In this section, to show the effect of varying these two parameters, tasks are executed with their actual execution time, and task re-mapping is not exploited. It can be seen that, in every case, utilizing our approach would lead to a system with lower peak power, energy as well as peak temperature. Besides, the expected effect of varying $\langle\alpha,\beta\rangle$ is confirmed in the experiments. For example, the average normalized peak power is progressively reduced when β increases from 0 to 1, as presented in Fig. 5a. Similarly, in Fig. 5b, the higher the α, the lower the energy consumption and peak temperature. Finally, as the algorithm looks further ahead in the future to find the best tasks to



(a) Normalized Peak Power (b) Normalized Energy (c) Normalized Peak Temperature

Fig. 5: Impact of varying α and β on peak power, energy and max. temperature.



(a) Peak Power (b) Energy (c) Max. Temperature

Fig. 6: Normalized improvement in peak power, energy and max. temperature for all scenarios.

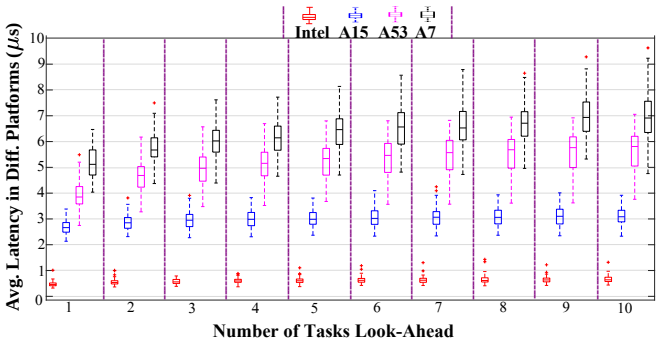


Fig. 7: Analyzing the timing overhead of run-time scheduler on four platforms.

assign the dynamic slack, the results are generally getting better, up to 1.25%, 1.25%, and 0.22% more reduction in peak power, energy and peak temperature. It is worth noting that, in this experiment, we intentionally disable the task re-mapping technique to ensure that the effect of $\langle \alpha, \beta \rangle$ is not skewed by another optimization.

For the other experiments in the paper, we consider $\langle \alpha, \beta \rangle = \langle 0.5, 0.5 \rangle$ that balances both peak power and temperature average reduction in comparison with other values of $\langle \alpha, \beta \rangle$.

*3) **The optimum number of tasks to look ahead and the effect of task re-mapping**:* In this subsection, we analyze the optimum number of tasks to look ahead (k) by evaluating the respective average quality of results without considering the overheads. The number of look-ahead tasks is varied from 1 to 10. The results presented in Fig. 6 are obtained from some scenarios of changing parameters in Table III with running on a homogeneous multi-core system. As a result, looking 4 tasks ahead provides a significant reduction in peak power and also in maximum temperature and energy consumption with and without task re-mapping. Hence, looking four tasks ahead is the average result of varying all parameters. The detail of finding the optimum k by varying the properties of tasks is discussed in Appendix B. Besides, when task re-mapping is used, the temperature, on average, is reduced by 2.7% compared to the case where task-remapping is disabled. In general, by looking ahead 4 tasks and enabling task re-mapping, the proposed method reduces the peak power, energy consumption, and maximum temperature on average by 14.6%, 39%, and 7.1%, respectively compared to [4] and 4.2%, 16%, and 3.1%, respectively compared to [26].

*4) **The analysis of scheduler timings overhead on different real platforms**:* To investigate the timing overhead of the proposed run-time scheduler, we analyze it on three real platforms, Intel Core i5, ARM big core (A15), and ARM LITTLE core (A7) on ODROID XU3/4 and ARM core (A53) in Xilinx Zynq UltraScale+ MPSoC board. Fig. 7 shows the overheads in each platform for different numbers of tasks looking ahead. Each boxplot shows the average latency for normal values of parameters in Table. III (d= 10%, c= 8, U/c= [0.5, 0.75], n= 80) with 100 task graphs. The following observation can be seen from the figure. First of all, the run-time timing overhead in the Intel platform is extremely small as compared to ARM processors. In the second observation, as the number of tasks look-ahead is increased, the latency is increased in all ARM processors. However, this latency increase is more evident in the A7 processor, while it is almost constant after looking seven tasks ahead in the A53 processor and four tasks ahead in the A15 processor. Furthermore, since big (A15) cores have high performance as compared to LITTLE (A7) cores, this timing overhead would be less. However, since a platform has lower performance, the range of latency between the minimum and maximum value is more significant. This fact is due to its lower performance and access to the memory and cache miss/hit.

To have a real implementation of our proposed method, we obtain the observed worst-case timing overhead of run-time scheduler, in which the appropriate task is selected for slack assignment and also a proper core for the re-mapping. Since many embedded systems use ARM processors, we evaluate our method on the ARM processor. We analyze the overhead on a LITTLE core of the ODROID XU4 platform. We examine both the Look-Ahead Unit and Re-mapping Unit in the run-time scheduler, separately and obtain the maximum observed timing overhead. To determine this overhead, we ran several applications (200 task graphs) with their various inputs on ARM LITTLE Core (A7). Based on these overheads, we set the observed worst-case of the Look-Ahead Unit to the maximum value. In addition, the scheduler in Re-mapping Unit checks other cores, whether it is possible to re-map a task for thermal management. In the worst-case, all cores are checked. Therefore, to have a close to accurate observed worst-

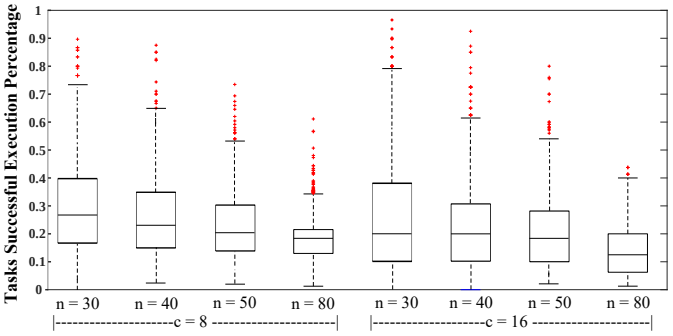
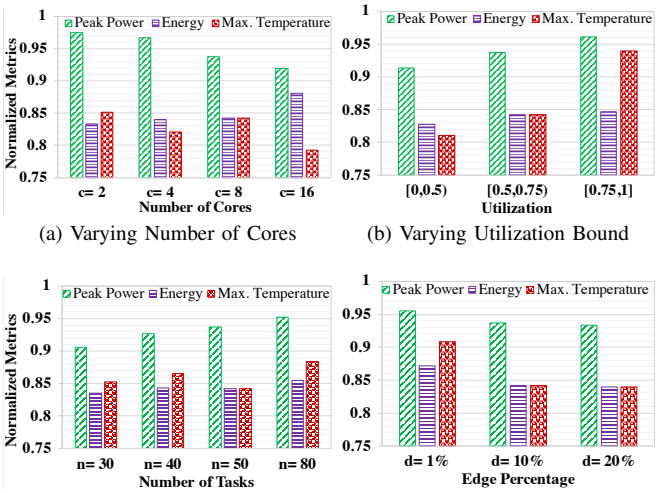


Fig. 8: Percent of successful executed tasks before their deadline in task graph based on the proposed method in [6] without considering timing overheads.



(a) Varying Number of Cores (b) Varying Utilization Bound

(c) Varying Number of Tasks (d) Varying Edge Percentage

Fig. 9: The improvements in peak power, energy and max. temperature in different scenarios normalized to [4].

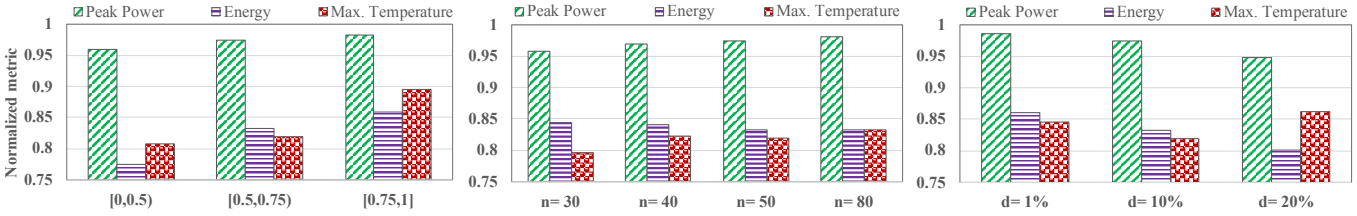
case timing overhead of the Re-mapping Unit, we obtain the worst timing overhead for each core and multiply it by the number of cores. Based on our observation and this measurement, the maximum timing overhead for Look-Ahead Unit is 56.417μs, and for Re-mapping Unit per core is 64.54μs. In order to ensure that the timing guarantees provided by the static schedule are not violated, we deduct these overheads from slack before assigning it to an appropriate task.

*5) **The latency of changing frequency in real platform**:* The main unit, DVFS governor, adjusts the frequency, which has a significant timing overhead. The ODROID XU3/4 board has a frequency range of $\langle 0.2, 1.4\rangle$ GHz for LITTLE cores and $\langle 0.2, 2\rangle$ GHz for big cores, with the step of 0.1 GHz. We change the frequency by using $\langle$cpufreq-set$\rangle$ program in two scenarios of scaling-down and scaling-up. Hence, the voltage is adjusted automatically according to the selected frequency. The maximum latency for all scenarios is at most 12.025*ms*. Besides we observe in our experiments that the latency of scaling-down transition is 342μs less than the scaling-up transition, on average. Regardless of the frequency scaling-down or up, we consider the latency of changing *V-f* level to be equal to 12.025*ms*. Due to this timing overhead, we deduct this latency from available dynamic slack before assigning it to a task to guarantee the correct execution of tasks before their deadlines. Since the re-mapping latency is 3.75*m*s [45] in the worst-case scenario and remapping is done in parallel with changing the frequency, it has no impact on the overall deadline.

*6) **The effect of latency on system schedulability**:* As discussed, considering the latencies of the run-time scheduler and changing frequency are critical in analyzing the system, which has not been considered in our previous work [6]. If these timing overheads are not studied, it may cause deadline miss of tasks and then catastrophic consequence. Our proposed method's effectiveness depends on the available slacks at runtime and the possibility of assigning them to the tasks. Therefore, if the latencies are not properly accounted for, some tasks may not be executed successfully before their deadline. Fig. 8 shows the percentage of successfully executed task sets before their deadlines during run-time phase in different scenarios in the method of [6]. The results are obtained for the normal scenario of some parameters (d= 10%, U/c= [0.5,0.75], c = 8, 16 and n = 30, 40, 50 and 80) and 1000 task graphs for each scenario. In this figure, we observe that as the number of tasks in the system with the same number of cores is increased, fewer task sets can be scheduled and meet their deadline. When there are more tasks in the system with the same *U/c*, the dynamic slacks that are incurred when the tasks finish earlier than their WCETs are smaller. The reason is that as the expected execution times of the tasks are decreased, the absolute differences between their actual execution time and WCETs are inherently small. Therefore, the possibility of missing a deadline is increased, and fewer tasks would be executed successfully before their deadline. In addition, if the number of cores in the system is increased, more task sets miss their deadlines. Since the re-mapping technique is used to manage temperature, all cores are checked in the worst-case. Therefore, by increasing the number of cores, the timing overhead of selecting a proper core for task re-mapping is increased. Since this latency has not been considered in [6] while a dynamic slack is assigned, using the re-mapping technique at runtime, may cause more deadline violation by increasing the number cores. In general, as shown in this figure, a high percentage of task sets miss their deadline, which is not acceptable in MC systems. Therefore, it is critical to consider timing overheads of run-time scheduler and changing frequency.

*7) **The analysis of the proposed method on peak power, energy and temperature improvement**:* In order to illustrate how effective our proposed method is with different parameters, we analyze the results under four separate scenarios of Table III, shown in Fig. 9, in which the results are normalized to [4]. These results are obtained for multi-core systems, in which there are homogeneous cores based on ARM A7. In general, as the applications get more complicated (e.g., having a large number of tasks or system utilization), it is harder to achieve significant savings in peak power, energy, and maximum temperature. Thanks to our task re-mapping technique where the tasks are redistributed more evenly to the cores at run-time based on their accumulated energy.

For the case of varying the number of cores, since our



(a) Varying Utilization Bound (b) Varying Number of Tasks (c) Varying Edge Percentage

Fig. 10: Normalized metrics running on the clustered heterogeneous multi-core platforms.

method only tries to optimize the peak power for each core individually to reduce the time overhead, it is more difficult to maintain a similar system peak power reduction when c is low. Nevertheless, as illustrated in Fig. 9a, the difference in peak power is significant by increasing the number of cores. In addition, as the temperature of each core is affected by the temperatures of neighboring cores, the reduction in maximum temperature is less by increasing the number of cores. On average, the peak power, maximum temperature, and energy consumption in the system is reduced by 5.015%, 17.32%, and 15.073%, respectively.

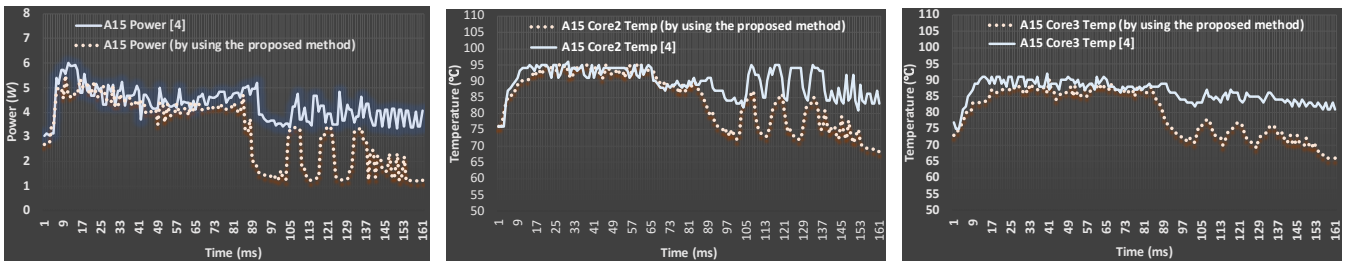
The effectiveness of our method depends on the available slacks at run-time and the possibility of assigning them to the tasks. Therefore, if there is less slack due to the nature of the application in terms of the number of tasks and system utilization, the reduction in peak power, energy consumption, and maximum temperature is less. For instance, in Fig. 9b, when the system utilization is getting higher, the idle time of the core between two consecutive tasks is getting smaller. The tasks also tend to execute longer. Thus, the amount of slacks that can be exploited at run-time is limited. But, overall, the peak power is reduced by at least 3.905%, and up to 8.59% in this scenario. Similarly, when there are more tasks in the system with the same U/c, the dynamic slacks incurred when the tasks finish earlier than their WCETs are smaller. The reason is that as the expected execution times of the tasks are decreased, the absolute differences between their actual execution time and WCETs are inherently small. However, as seen in Fig. 9c, our method manages to reduce the peak power, energy, and maximum temperature on average by 6.96%, 15.61% and 13.91%, respectively.

Besides, the possibility of releasing the tasks earlier than their presumed start times also affects the outcomes. When the dependency between the tasks is high, a significant amount of them cannot be released earlier. This behavior can either have a positive or negative impact on the system. For the former, the cores might have more idle time because the tasks have to wait longer for their predecessors to finish. For the latter, our method has less opportunity to apply DVFS to tasks. However, at run-time, these idle periods might overlap with the other tasks with the already reduced *V-f* level. The peak power of the whole system is consequently reduced. It can be seen in Fig. 9d that, when $d = 20\%$, the best peak power and maximum temperature reduction is achieved compared the cases where $d = 1\%$ and $d = 10\%$.

*8) **The analysis of the proposed method on peak power, energy and temperature improvement in a multi-core platform based on the ODROID-XU3 architecture**:* In this section, we analyze the improvement of peak power, energy consumption, and maximum temperature in a clustered heterogeneous multi-core processor in which there are four big (A15) and four LITTLE (A7) cores. Here, we have a common *V-f* level for all cores within the same cluster (cluster with big cores or cluster with LITTLE cores), while in the previous results, the frequency of each core was changed individually. We show the results in Fig. 10, in which the improvements in peak power, energy consumption and maximum temperature in the clustered heterogeneous multi-core system across all experiments are up to 5.25%, 22.44%, and 20.33%, respectively in comparison with [4]. The trends for the clustered heterogeneous multi-core architecture are similar to that obtained for the homogeneous architecture in the previous sub-section. The only notable difference is in the peak power, which is on average 3.461% worse than the tasks on a homogeneous multi-core system, due to enforcing of common *V-f* level for the entire cluster. Therefore, the power improvement is somewhat lower.

*9) **Evaluation of running real MC task graph (Unmanned Air Vehicle) on real platform**:* Now, we validate the proposed online technique with a real-life application task graph, presented in Fig. 1a, running on the ODROID XU3. In particular, we evaluate the impacts of changing frequency on the system power and temperature in this section. The UAV application consists of seven dependent tasks executed on two cores, which has been presented in Fig. 1a. Since there are no available real benchmarks for the tasks of the graph, we used different benchmarks of Mibench [41] as tasks of the graph. Then, we obtain the WCETs and maximum power consumption of each task with running on the ODROID XU3. Since WCET analysis is a complicated task [46], we used an existing WCET estimation tool called OTAWA [47], to capture the highest WCETs (C_i^{HI}). In addition, we run each benchmark, 10,000 times and select the maximum of the measured execution time as the lowest WCET (C_i^{LO}). To analyze the system temperature, we run the application on Core2 and Core3 that in general have a higher temperature due to their proximity to the memory and other components. Hence, there is a temperature sensor for each big core and a power sensor for each cluster of ODROID XU3. Therefore, the power and temperature data of this section are exploited from the board sensors.

Fig. 11 shows the power trace of the cluster of big cores and temperature trace of two cores during runtime in two scenarios of using our DVFS-based proposed method and presented method of [4]. At runtime, to analyze a task execution time, we select a docker container to run a task and check the time to be aware of the exact start and finish times of the task. Then, the dynamic slack is computed, the slack between



(a) Power trace of the big cluster (b) Temperature trace of A15-core2 (c) Temperature trace of A15-core3

Fig. 11: Power and temperature sensor data of the ODROID XU3, by running the real MC task graph (Unmanned Air Vehicle).

task's actual completion time and its WCET. Fig. 11a shows the power traces of the method of [4] and our proposed method by considering two tasks looking ahead that the DVFS has been used from almost 90*ms*. In addition, as shown in Fig. 11b and 11c, in general, the average core temperature has been decreased by using the DVFS-based proposed method and looking two tasks ahead. Based on the scheduling of tasks in Fig. 1, one of the cores is active until near the middle of the period. However, the temperature of each core is affected by the temperature of neighboring cores. In addition, after executing two tasks in each core and using the dynamic slack to reduce the speed, the cores temperatures are decreasing. Besides, in a part of the task graph period, only Core2 is active but still has high temperatures. Therefore, after applying the proposed method and reducing the *V-f* levels, the cores temperatures are reduced. The proposed method will be more effective and have a significant improvement if there are more tasks that are run on a system with more cores.

VIII. Conclusion and Future Work

In this article, we studied online peak power and peak temperature reduction in mixed-criticality embedded systems and analyzed the proposed run-time power-aware scheduler on clustered multi-core real platforms. Our presented method uses re-mapping technique and DVFS at runtime whenever there is a dynamic slack. We also proposed the associated cost functions to select the most appropriate task to assign the dynamic slacks to decrease its *V-f* level or to re-map it to another core. We showed that by increasing the number of tasks to look ahead, more peak power and maximum temperature reduction are achieved. In addition, the proposed power-aware scheduler was analyzed in terms of run-time timing overhead in different multi-core platforms. We focused on reducing the run-time scheduler latency to have more usage of dynamic slack and, consequently, more peak power and maximum temperature reduction, while the tasks' deadlines are guaranteed. The results show up to 5.25%, 20.33%, and 22.44% reduction in peak power, peak temperature, and average energy consumption, respectively, compared to recent studies. The proposed solution can manage the system peak power consumption by considering greedily optimizing individual cores. Therefore, as future work, we would consider the management of peak power and thermal cycling issues by considering the whole system to have better improvement in peak power consumption.

References

[1] S. Baruah, V. Bonifaci, G. DAngelo, H. Li, A. Marchetti-Spaccamela, S. van der Ster, and L. Stougie, "The preemptive uniprocessor scheduling of mixed-criticality implicit-deadline sporadic task systems," in *Proc. Euromicro Conference on Real-Time Systems (ECRTS)*, July 2012, pp. 145–154.

[2] H. Su and D. Zhu, "An elastic mixed-criticality task model and its scheduling algorithm," in *Proc. Design, Automation & Test in Europe Conference & Exhibition (DATE)*, March 2013, pp. 147–152.

[3] A. Burns and R. I. Davis, "A survey of research into mixed criticality systems," *ACM Computing Surveys*, vol. 50, no. 6, Nov. 2017.

[4] R. Medina, E. Borde, and L. Pautet, "Availability enhancement and analysis for mixed-criticality systems on multi-core," in *Proc. Design, Automation Test in Europe Conference Exhibition (DATE)*, March 2018, pp. 1271–1276.

[5] R. Ernst and M. Di Natale, "Mixed criticality systems—a history of misconceptions?" *IEEE Design & Test*, vol. 33, no. 5, pp. 65–74, 2016.

[6] B. Ranjbar, T. D. A. Nguyen, A. Ejlali, and A. Kumar, "Online peak power and maximum temperature management in multi-core mixed-criticality embedded systems," in *Proc. Euromicro Conference on Digital System Design (DSD)*, Aug 2019, pp. 546–553.

[7] S. Baruah, "The federated scheduling of systems of mixed-criticality sporadic dag tasks," in *Proc. IEEE Real-Time Systems Symposium (RTSS)*, Nov 2016, pp. 227–236.

[8] M. A. Awan, D. Masson, and E. Tovar, "Energy efficient mapping of mixed criticality applications on unrelated heterogeneous multicore platforms," in *Proc. IEEE Symposium on Industrial Embedded Systems (SIES)*, May 2016, pp. 1–10.

[9] W. Munawar, H. Khdr, S. Pagani, M. Shafique, J. Chen, and J. Henkel, "Peak power management for scheduling real-time tasks on heterogeneous many-core systems," in *Proc. IEEE International Conference on Parallel and Distributed Systems (ICPADS)*, Dec 2014, pp. 200–209.

[10] Z. Li, X. Hua, C. Guo, and S. Ren, "Empirical study of energy minimization issues for mixed-criticality systems with reliability constraint," in *Proc. Workshop on Low-Power Dependable Computing (LPDC)*, 2014, pp. 3–5.

[11] P. Huang, P. Kumar, G. Giannopoulou, and L. Thiele, "Energy efficient dvfs scheduling for mixed-criticality systems," in *Proc. International Conference on Embedded Software (EMSOFT)*. ACM, 2014, pp. 1–10.

[12] Z. Li, C. Guo, X. Hua, and S. Ren, "Reliability guaranteed energy minimization on mixed-criticality systems," *Journal of Systems and Software*, vol. 112, pp. 1–10, 2016.

[13] A. Taherin, M. Salehi, and A. Ejlali, "Reliability-aware energy management in mixed-criticality systems," *IEEE Transactions on Sustainable Computing (TSUSC)*, vol. 3, no. 3, pp. 195–208, 2018.

[14] T. Li, T. Zhang, G. Yu, Y. Zhang, and J. Song, "Ta-mcf: Thermal-aware fluid scheduling for mixed-criticality system," *Journal of Circuits, Systems and Computers*, vol. 28, no. 02, p. 1950029, 2019.

[15] J. Lee, B. Yun, and K. G. Shin, "Reducing peak power consumption in multi-core systems without violating real-time constraints," *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, vol. 25, no. 4, pp. 1024–1033, April 2014.

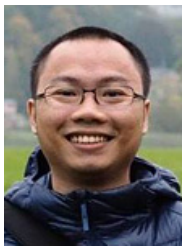
[16] B. Lee, J. Kim, Y. Jeung, and J. Chong, "Peak power reduction methodology for multi-core systems," in *Proc. International SoC Design Conference (ISOCC)*, Nov 2010, pp. 233–235.

[17] M. Ansari, S. Safari, A. Yeganeh-Khaksar, M. Salehi, and A. Ejlali, "Peak power management to meet thermal design power in fault-tolerant embedded systems," *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, vol. 30, no. 1, pp. 161–173, Jan 2019.

[18] V. Chaturvedi, A. K. Singh, W. Zhang, and T. Srikanthan, "Thermal-aware task scheduling for peak temperature minimization under periodic constraint for 3d-mpsocs," in *Proc. IEEE International Symposium on Rapid System Prototyping (RSP)*, Oct 2014, pp. 107–113.

[19] R. Kabir and B. Izadi, "Temperature and energy aware scheduling of heterogeneous processors," in *Proc. International Conference on Contemporary Computing (IC3)*, Aug 2016, pp. 1–7.
[20] M. Bao, A. Andrei, P. Eles, and Z. Peng, "On-line thermal aware dynamic voltage scaling for energy optimization with frequency/temperature dependency consideration," in *Proc. ACM/IEEE Design Automation Conference (DAC)*, July 2009, pp. 490–495.
[21] M. Qiu, J. Niu, F. Pan, Y. Chen, and Y. Zhu, "Peak temperature minimization for embedded systems with dvs transition overhead consideration," in *Proc. International Conference on High Performance Computing and Communication (HPCC) & International Conference on Embedded Software and System ICESS*, June 2012, pp. 477–484.
[22] H. Hong, J. Lim, H. Lim, and S. Kang, "Thermal-aware dynamic voltage frequency scaling for many-core processors under process variations," *IEICE Electronics Express*, vol. 10, no. 14, 2013.
[23] T. Chantem, X. S. Hu, and R. P. Dick, "Temperature-aware scheduling and assignment for hard real-time applications on mpsocs," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 19, no. 10, pp. 1884–1897, Oct 2011.
[24] Jian-Jia Chen, Chuan-Yue Yang, and Tei-Wei Kuo, "Slack reclamation for real-time task scheduling over dynamic voltage scaling multiprocessors," in *Proc. on Sensor Networks, Ubiquitous, and Trustworthy Computing (SUTC)*, vol. 1, 2006, pp. 1–8.
[25] J. Kang and S. Ranka, "Dynamic slack allocation algorithms for energy minimization on parallel machines," *Journal of Parallel and Distributed Computing (JPDC)*, vol. 70, no. 5, pp. 417–430, 2010.
[26] D. Zhu, R. Melhem, and B. R. Childers, "Scheduling with dynamic voltage/speed adjustment using slack reclamation in multiprocessor real-time systems," *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, vol. 14, no. 7, pp. 686–700, 2003.
[27] A. K. Singh, A. Das, and A. Kumar, "Energy optimization by exploiting execution slacks in streaming applications on multiprocessor systems," in *Proc. on Design Automation Conference (DAC)*, 2013.
[28] L. Zhang, K. Li, K. Li, and Y. Xu, "Joint optimization of energy efficiency and system reliability for precedence constrained tasks in heterogeneous systems," *International Journal of Electrical Power & Energy Systems*, vol. 78, pp. 499–512, 2016.
[29] A. Martins, M. Ruaro, A. Santana, and F. G. Moraes, "Runtime energy management under real-time constraints in mpsocs," in *Proc. of IEEE International Symposium on Circuits and Systems (ISCAS)*, 2017, pp. 1–4.
[30] M. Chisholm, N. Kim, S. Tang, N. Otterness, J. H. Anderson, F. D. Smith, and D. E. Porter, "Supporting mode changes while providing hardware isolation in mixed-criticality multicore systems," in *Proc. International Conference on Real-Time Networks and Systems (RTNS)*, 2017, p. 58–67.
[31] L. Sigrist, G. Giannopoulou, P. Huang, A. Gomez, and L. Thiele, "Mixed-criticality runtime mechanisms and evaluation on multicores," in *Proc. IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, April 2015, pp. 194–206.
[32] J. L. Herman, C. J. Kenna, M. S. Mollison, J. H. Anderson, and D. M. Johnson, "Rtos support for multicore mixed-criticality systems," in *Proc. IEEE Real Time and Embedded Technology and Applications Symposium*, April 2012, pp. 197–208.
[33] R. Trüb, G. Giannopoulou, A. Tretter, and L. Thiele, "Implementation of partitioned mixed-criticality scheduling on a multi-core platform," *ACM Transactions Embedded Computing System (TECS)*, vol. 16, no. 5s, Sep. 2017.
[34] Z. Guo, A. Bhuiyan, D. Liu, A. Khan, A. Saifullah, and N. Guan, "Energy-efficient real-time scheduling of dags on clustered multi-core platforms," in *Proc. IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, April 2019, pp. 156–168.
[35] A. K. Singh, S. Dey, K. R. Basireddy, K. McDonald-Maier, G. V. Merrett, and B. M. Al-Hashimi, "Dynamic energy and thermal management of multi-core mobile platforms: A survey," *IEEE Design & Test*, 2020.
[36] G. Giannopoulou, N. Stoimenov, P. Huang, and L. Thiele, "Mapping mixed-criticality applications on multi-core architectures," in *Proc. Design, Automation Test in Europe Conference Exhibition (DATE)*, March 2014, pp. 1–6.
[37] M. Chisholm, B. C. Ward, N. Kim, and J. H. Anderson, "Cache sharing and isolation tradeoffs in multicore mixed-criticality systems," in *Proc. IEEE Real-Time Systems Symposium (RTSS)*, Dec 2015, pp. 305–316.
[38] M. Chisholm, N. Kim, B. C. Ward, N. Otterness, J. H. Anderson, and F. D. Smith, "Reconciling the tension between hardware isolation and data sharing in mixed-criticality, multicore systems," in *Proc. IEEE Real-Time Systems Symposium (RTSS)*, Nov 2016, pp. 57–68.
[39] N. Kim, S. Tang, N. Otterness, J. H. Anderson, F. D. Smith, and D. E. Porter, "Supporting i/o and ipc via fine-grained os isolation for mixed-criticality real-time tasks," in *Proc. International Conference on Real-Time Networks and Systems (RTNS)*, 2018, pp. 191–201.
[40] Kcachegrind tool. [Online]. Available: http://kcachegrind.sourceforge.net/html/Home.html
[41] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown, "Mibench: A free, commercially representative embedded benchmark suite," in *Proc. IEEE International Workshop on Workload Characterization. WWC-4*, Dec 2001, pp. 3–14.
[42] M. Salehi and A. Ejlali, "A hardware platform for evaluating low-energy multiprocessor embedded systems based on cots devices," *IEEE Transactions on Industrial Electronics (TIE)*, vol. 62, no. 2, pp. 1262–1269, 2015.
[43] Wei Huang, S. Ghosh, S. Velusamy, K. Sankaranarayanan, K. Skadron, and M. R. Stan, "Hotspot: a compact thermal modeling methodology for early-stage vlsi design," *IEEE Transactions on Very Large Scale Integration Systems (TVLSI)*, vol. 14, no. 5, pp. 501–513, May 2006.
[44] Y. Gong, J. J. Yoo, and S. W. Chung, "Thermal modeling and validation of a real-world mobile ap," *IEEE Design & Test*, vol. 35, no. 1, pp. 55–62, Feb 2018.
[45] K. R. Basireddy, A. K. Singh, B. M. Al-Hashimi, and G. V. Merrett, "Adamd: Adaptive mapping and dvfs for energy-efficient heterogeneous multi-cores," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, 2019.
[46] M. Bazzaz, A. Hoseinghorban, and A. Ejlali, "Fast and predictable non-volatile data memory for real-time embedded systems," *IEEE Transactions on Computers (TC)*, pp. 1–1, 2020.
[47] C. Ballabriga, H. Cassé, C. Rochange, and P. Sainrat, "Otawa: an open toolbox for adaptive wcet analysis," in *IFIP International Workshop on Software Technolgies for Embedded and Ubiquitous Systems*. Springer, 2010, pp. 35–46.

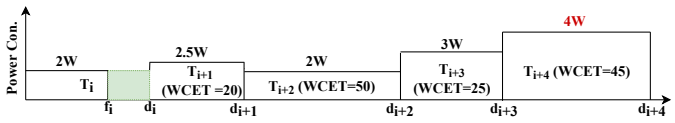
Behnaz Ranjbar received the B.S. degree in computer engineering from the Amirkabir University of Technology, in 2012, and the M.S. degree from the Sharif University of Technology, Tehran, Iran, in 2014. She is currently pursuing the joint Ph.D. degree with the Sharif University of Technology and the Chair for Processor Design, Technische Universität (TU) Dresden, Dresden, Germany. Her research interest includes real-time systems, fault-tolerant, and low-power embedded systems design.



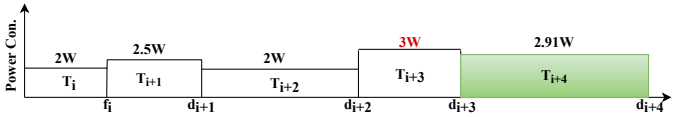
Tuan D. A. Nguyen has obtained the B.Eng. in computer engineering from Ho Chi Minh City University of Technology, Vietnam, in 2011, and the PhD degree in computer engineering from National University of Singapore, Singapore, in 2018. His interests are in Partially Reconfigurable Heterogeneous Multi-processor System-on-Chip. He was with the Technische Universität Dresden, Dresden, Germany, where he worked as PostDoctoral researcher at Chair for Processor Design from Jan. 2018 to Dec. 2019.

Alireza Ejlali received the PhD degree in computer engineering from Sharif University of Technology in Tehran, Iran, in 2006. He is currently an associate professor of computer engineering at Sharif University of Technology. From 2005 to 2006, he was a visiting researcher in the Electronic Systems Design Group, University of Southampton, Southampton, United Kingdom. He is currently the Director of the Embedded Systems Research Laboratory, Department of Computer Engineering, Sharif University of Technology. His research interests include low power design, real-time systems, and fault-tolerant embedded systems.

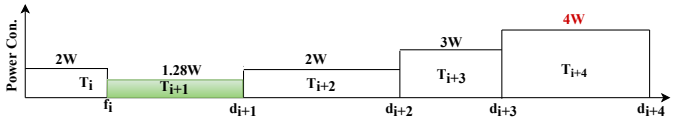
Akash Kumar (SM'13) received the joint Ph.D. degree in electrical engineering and embedded systems from the Eindhoven University of Technology, Eindhoven, The Netherlands, and the National University of Singapore (NUS), Singapore, in 2009. From 2009 to 2015, he was with NUS. He is currently a Professor with Technische Universität Dresden, Dresden, Germany, where he is directing the Chair for Processor Design. His current research interests include the design, analysis, and resource management of low-power and fault-tolerant embedded multiprocessor systems.



(a) Power trace of the core



(b) Power trace of the core after assigning the slack to the task T_{i+4}



(c) Power trace of the core after assigning the slack to the task T_{i+1}

Fig. 12: Power trace of a core

Appendix A
Optimum Solution for Minimizing the Peak Power

In this paper, we focus on peak power minimization in individual cores during the run-time phase in addition to system peak power reduction. Here, we prove that our proposed algorithm is optimal when $\langle \alpha, \beta \rangle = \langle 0, 1 \rangle$ in Eq. 5 to select the task solely based on its peak power consumption. Let us assume that the task T_i finishes its execution at time f_i, ahead of its deadline d_i, and a dynamic slack ($S_{dyn} = d_i - f_i$) is generated. The algorithm looks k tasks after generated slack to select the appropriate task and use the generated slack to reduce its *V-f* level and, consequently, decrease its power consumption. Without loss of generalization, assume that task T_{i+l} consumes the highest peak power in the core within the k tasks looking ahead, presented in Eq. 7. This equation can be rewritten as Eq. 8, in which $Max(T_{i+1}^{pow}, ..., T_{i+l-1}^{pow}, T_{i+l+1}^{pow}, ..., T_{i+k}^{pow}) < T_{i+l}^{pow}$.

$$Pow_{core}^{max}|_{[d_i, d_{i+k}]} = Max(T_{i+j}^{pow})|_{j=1:k} = T_{i+l}^{pow} \quad 1 \le l \le k \tag{7}$$

$$Pow_{core}^{max}|_{[d_i, d_{i+k}]} = Max(Max(T_{i+1}^{pow}, ..., T_{i+l-1}^{pow}, T_{i+l+1}^{pow}, ..., T_{i+k}^{pow}), T_{i+l}^{pow}) \tag{8}$$

If $T_{i+l}^{pow'}$ is the maximum power consumption of task T_{i+l} after reclaiming the slack and reducing the *V-f* level, then $T_{i+l}^{pow'} < T_{i+l}^{pow}$, therefore, the core's maximum power consumption can be written as follows, which is less than T_{i+l}^{pow}:

$$Pow_{core}^{max}|_{[d_i, d_{i+k}]} = Max(Max(T_{i+1}^{pow}, ..., T_{i+l-1}^{pow}, T_{i+l+1}^{pow}, ..., T_{i+k}^{pow}), T_{i+l}^{pow'}) \tag{9}$$

If we select one of the other task between $\{T_{i+1}, ..., T_{i+l-1}, T_{i+l+1}, ..., T_{i+k}\}$, reduce its *V-f* level and consequently, its power consumption, then the peak power of the core is still limited by T_{i+l}^{pow} according to Eq. 8. Hence, this power consumption is more than the optimum power consumption obtained by Eq. 9. T_{i+l} is, therefore, the optimum task to which the slack should be assigned (given the constraint that all the slack is assigned to *one* of the following k tasks). We conclude that whenever a dynamic slack is generated, the proposed approach for selecting the appropriate task provides the optimum solution to minimize the peak power consumption of individual cores in the run-time phase.

Fig. 12 shows a part of a static schedule of tasks on a core. Based on the peak power consumption of tasks in Fig. 12a, task T_{i+4} is the appropriate task which consumes the highest peak power in the core in the time interval $[d_i, d_{i+4}]$. Therefore, assigning the slack to this task will lower the peak power to below 4W (If we have a dynamic slack ($S_d = d_i - f_i = 5$), then $Power_{core}^{max} = 3W$ after slack assignment, shown in Fig. 12b). If we assign the generated slack to one of the other tasks (e.g., T_{i+1}) instead, then the peak power of the core is still limited by T_{i+4}, i.e. 4W, as can be seen in Fig. 12c.

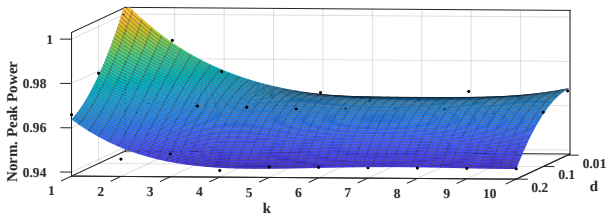


Fig. 13: Impact of number of look-ahead tasks and edge percentage on normalized peak power consumption.

Appendix B
Finding the Optimum Number of Look-Ahead Tasks by Varying the Tasks' Properties

Here, we show the relation between the number of look-ahead tasks (k) and the task property, edge percentage (d%) to model the system capability such as peak power minimization, energy consumption, and maximum temperature. For this analysis, the data from the experiments with four cores (c), and the system utilization per core (U/c) in the range of [0.5,0.75) is used. Average data of 100 task set runs has been used.

We use the Matlab Curve Fitting Tool to derive the polynomial functions of various system parameters. Fig. 13 shows the curve of the system peak power consumption by varying k and d normalized to the result for $k = 1$ and the corresponding equation is shown in Eq. 10. This equation is the polynomial function with the maximum degree of four with the minimum Root Mean Square Error (RMSE), equal to 0.0024.

$$\begin{aligned} Pow_{peak}^{Norm.}(k, d) = 1.033 - 0.5082d - 0.03374k + 1.332d^2 + \\ 0.1799dk + 0.006184k^2 - 0.7267d^2k - 0.01158dk^2 - \\ 0.0005375k^3 + 0.05294d^2k^2 - 0.0001314dk^3 + 1.912 \times 10^{-5}k^4 \end{aligned} \tag{10}$$

The equation above can also be used to mathematically derive the optimal k for a particular task property to optimize the various metrics. For example, if d is kept as 20% in Eq. 10, the minimum value of the curve is obtained when $k = 5$ which is shown in Fig. 14. In addition, by deriving the corresponding equations for the normalized maximum temperature (Eq.11) and energy consumption (Eq.12), the optimum value of k for

the system's max. temperature is $k = 3$ and for the energy consumption is $k = 4$, as shown in Fig. 14.

$$Energy_{peak}^{Norm.}(k,d) = 0.9347 + 1.176d - 0.05964k - 3.304d^2 - 0.01962dk + 0.01355k^2 + 0.3278d^2k + 0.005128dk^2 - 0.001431k^3 - 0.02466d^2k^2 - 0.000231dk^3 + 5.616 \times 10^{-5}k^4 \quad (11)$$

$$T_{peak}^{Norm.}(k,d) = 0.9925 + 0.07826d - 0.006975k - 0.08123d^2 - 0.008346dk + 0.001644k^2 + 0.05333d^2k + 0.001238dk^2 - 0.0001762k^3 - 0.003814d^2k^2 - 10^{-5}(4.653dk^3 + 0.6901k^4) \quad (12)$$

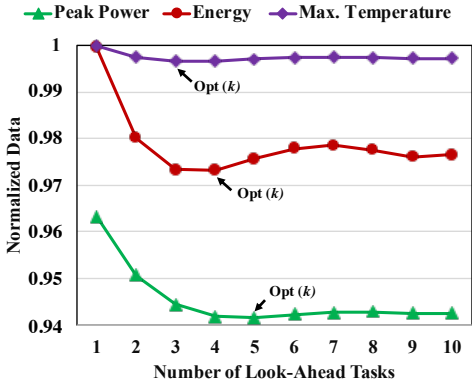


Fig. 14: Impact of number of look-ahead tasks on normalized peak power, energy and max. temperature while $d = 20\%$.