Online Peak Power and Maximum Temperature Management in Multi-Core Mixed-Criticality Embedded Systems

1st Behnaz Ranjbar , 2nd Tuan D. A. Nguyen, 3rd Alireza Ejlali , 4th Akash Kumar
[1,2,4]*Chair for Processor Design, Technical University of Dresden, Dresden, Germany*
[1,3]*Embedded System Research Laboratory (ESRLab.), Sharif University of Technology, Tehran, Iran*
[1]*branjbar@ce.sharif.edu,*[2,4]*{tuan_duy_anh.nguyen1,akash.kumar}@tu-dresden.de,*[3]*ejlali@sharif.edu*

***Abstract*—In this work, we address peak power and maximum temperature in multi-core Mixed-Criticality (MC) systems. In these systems, a rise in peak power consumption may generate more heat beyond the cooling capacity. Additionally, the reliability and timeliness of MC systems may be affected due to excessive temperature. Therefore, managing peak power consumption has become imperative in multi-core MC systems. In this regard, we propose an online peak power 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 Dynamic Voltage and Frequency Scaling (DVFS). Specifically, our approach examines multiple tasks ahead to determine the most appropriate one for slack assignment instead of just one task as in the literature. The selection is based on the impact of the tasks on peak power and temperature of the system. The DVFS is then applied to that task to reduce the system peak power and maximum temperature. Further, a re-mapping technique is proposed to further improve the results. Our experimental results show that our heuristic achieves up to 18.2% reduction in system peak power consumption and 8.1% reduction in maximum temperature compared to an existing method. The inherent energy consumption is also reduced by up to 50%.**

***Keywords*-Mixed-Criticality System; Multi-Core System; Dynamic Slacks; Peak Power Consumption; Run-Time Phase;**

I. Introduction

In most of the safety-critical real-time applications (in medical, flight control, etc. devices), tasks are classified into multiple criticality levels in order to maintain the predictability of the applications under different unexpected behaviors. The classification is done based on the functionality of the tasks with respect to how important they are to the application [1], [2]. These tasks have to be analyzed at design-time to obtain their Worst-Case Execution Time (WCET) [1]. After that, proper mapping and scheduling strategies are derived to satisfy the real-time constraints and to optimize the processor capacity usage [3]. The WCET of tasks are not considered as a robust metric and tasks can exceed their WCET due to unexpected behavior or internal defects [1], [4]. Such scenarios and unpredictability cannot be tolerated in safety-critical real-time applications. Therefore, Mixed-Criticality (MC) systems are designed to tackle this issue to avoid catastrophic consequences. In these systems, High-Criticality (HC) tasks are then analyzed with different assumptions, pessimistic and optimistic, in order to obtain two WCETs for them. For these MC tasks, if the execution time of at least one HC task exceeds its optimistic WCET, i.e., the system switches from low-criticality (LO) mode to high-criticality (HI) mode, all HC tasks continue their execution with their pessimistic WCET.

MC systems are getting more complicated due to the growth in number of tasks, therefore multi-core platforms are utilized to execute the tasks in parallel, thereby improving the systems performance [4]. As the degree of freedom (in terms of availability of the cores) increases, it is not trivial to guarantee the real-time constraints while managing the system peak power. Systems with high peak power are more likely to generate unexpected heat that is beyond the intended cooling capacity [5]. These systems will be more susceptible to failures and instability [5]. In other words, the reliability, lifetime, and timeliness of these systems will be undesirably affected [6]. As a result, minimizing peak power in multi-core MC systems is a major issue that should be addressed.

There are research works that propose methods to reduce the average power consumption in MC systems with *only independent* tasks [4], [7]–[10]. Most of them apply the Dynamic Voltage and Frequency Scaling (DVFS) technique at design time. Thus, they may miss the opportunity to reduce the power further at runtime when the tasks may finish sooner than their WCET. Furthermore, in these methods, when the system switches from LO to HI mode, the Low-Criticality (LC) tasks are dropped completely to guarantee the correct execution of the HC tasks to meet their deadlines. For some applications e.g., mission-critical applications [11], completely dropping the LC tasks reduces the quality of service (QoS) of the system which could have been avoided [12]. On the other hand, [5], [6], [13], [14] also try to reduce peak power at design time but they are only for hard real-time systems without MC.

In this paper, we propose a heuristic to manage peak power consumption in MC systems during runtime. In order to achieve this, we exploit dynamic slacks, the gap between tasks' actual completion time and their WCET, along with DVFS. There are two phases in our approach: At design time, the static mapping and scheduling tables of the tasks at LO and HI modes are obtained from [12]. In this case, the number of LC tasks that have to be dropped in the HI mode is minimized; which has a positive effect on the system QoS. After that, at runtime, we propose an approach that examines multiple tasks in the future (look-ahead) to select the most appropriate one to assign the currently available dynamic slack to. The selection is based on the impact of the tasks on the power and temperature of the system which is quantified by a weighted multi-objective cost function. The voltage level of the core that runs the task is decreased accordingly using DVFS. The effects of the number of future tasks evaluated in the look-ahead approach are carefully studied and analyzed in this work. Additionally, besides exploiting the dynamic slacks, we propose a task re-mapping technique at runtime to further improve the system temperature profile.

In summary, the main contributions of this work are:

- An online peak power and maximum temperature management in multi-core MC systems while respecting deadline requirements of tasks in both LO and HI modes.
- A multi-task look-ahead approach to make sure that dy-

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 Universitaet at Dresden.

Table I: Summary of State-of-the-Art Approaches

	Single=Multi-Core	Peak Power	Average Power	Temperature	DVFS (online=offline)	MC Tasks	DAG Model
[4], [7]–[10]	Single/Multi-Core	×	✓	×	offline	✓	×
[15]	Single-Core	×	✓	✓	offline	✓	×
[5], [6]	Multi-Core	✓	×	×	×	×	×
[13]	Multi-Core	✓	×	×	×	×	✓
[14]	Multi-Core	✓	✓	×	offline	×	✓
[16]–[21]	Single/Multi-Core	×	✓	✓	offline	×	✓
[22], [23]	Multi-Core	×	✓	×	online	×	✓
Our Method	Multi-Core	✓	✓	✓	online	✓	✓

namic slacks are assigned to the tasks that have the most impact on the system.

- An online task re-mapping technique that also tries to exploit dynamic slacks to remap the tasks to other cores which can lower the system temperature.

We ran simulations with HOTSPOT [24] and MEET [25] to compare our method with state-of-the-art methods. Experiments in Section V show that our method provides significant peak power reduction, peak temperature reduction and average energy consumption up to 18.2%, 8.1% and 50%, respectively, compared to recent previous works.

The rest of the paper is organized as follows. In Section II, we review related works. In Section III, we introduce the models. The problem and our method in detail are presented in Section IV. Finally, we analyze and conclude experiments in Sections V and VI 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. Since our focus is on online power and thermal management, we only consider the works presented for MC or non-MC systems with the similar scope.

Generally, the related works on power and thermal management for real-time systems can be classified based on the assumed platform, single or multi-core, MC or non-MC systems. Table I summarizes the recent works with different target optimization objectives of peak power, average power or maximum temperature. As can be seen, some of the works present methods to minimize average power in MC systems which are single or multi-core [4], [7]–[10]. In general, they only optimize the average power in the LO mode. 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. In addition, the other papers consider thermal management in MC systems (second column) [15], [26]. However, the presented method in [26], CONTREX, focuses on methodology to analyze the MC systems rather than proposing any mapping and scheduling algorithm. Some studies concentrate on peak power management in multi-core systems at design time (row 3-5). They only consider hard real-time tasks with one criticality level which are not practical for MC tasks. It should be mentioned that authors in [13] work on the dependent task model in which the execution of some tasks are 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 energy or thermal management that use DVFS by considering the dependent task model are shown in Table I (row 6-9). Some of these have used the online DVFS to reduce the energy or temperature [21]–[23]. In [21], 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 runtime 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. In addition, Kang et al. [22] propose an algorithm which uses DVS to minimize energy without concerning about the tasks' deadlines; which is not suitable for MC systems. Zhu et al. [23] 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.

In this work, we study online peak power and maximum temperature management for dependent MC tasks that are executed on a multi-core processor, which is not considered in existing MC works.

III. Models and Measurement Methods

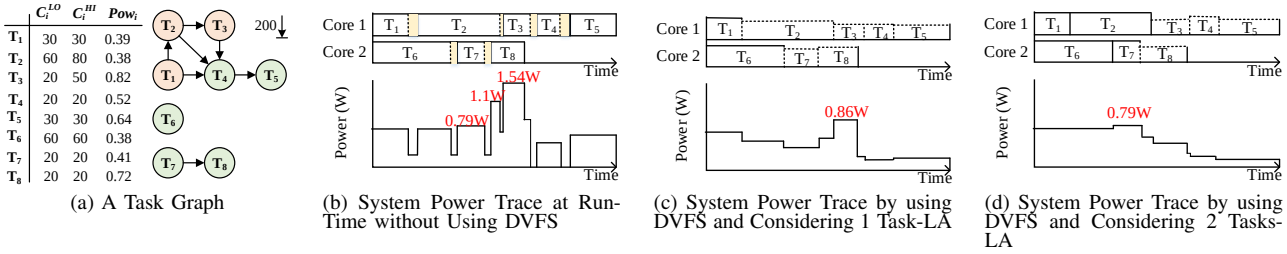
A. Task Model

We consider real-time applications consisting of dependent periodic MC tasks, such that, each task τ_i is represented as $\{\tau_i; C_i^{LO}; C_i^{HI}; d_i; Su_i; Pr_i\}$ [12], [27]. We consider dual-criticality system where each MC task can be either high-critical (τ_i = HC) or low-critical (τ_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 $C_i^{LO} \leq C_i^{HI}$. The criticality levels of tasks in a graph are determined similarly to [12], [27]. If a task is a predecessor of an HC task, then it is considered as an HC task. 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 should 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. In this case, 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 [12], [27].

B. Hardware Architecture Model

We consider a multi-core processor comprising of m cores $\{C_1; C_2; :::; C_m\}$. We assume that the system is DVFS-enabled and the cores can operate at multiple voltage (V) and frequency (I) levels. Furthermore, each core is equipped with a thermal sensor to measure the temperature.



(a) A Task Graph

(b) System Power Trace at Run-Time without Using DVFS

(c) System Power Trace by using DVFS and Considering 1 Task-LA

(d) System Power Trace by using DVFS and Considering 2 Tasks-LA

Figure 1: A Motivational example for a **real-life application** and scenarios with different number of tasks look-ahead.

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}) [9], [10]. P_{ind} refers to the power related to the memory and I/O activities. As mentioned in Section III-B, each core can operate at different *V-f* levels so that, its 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.

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

in which: (is the scaling factor),

$$f_{min} \leq f = \times f_{max} \leq f_{max}; V_{min} \leq V = \times V_{max} \leq V_{max}$$

IV. Proposed Method

In this section, we first formulate the problem, have an example to motivate our goal and then propose the solution.

A. Problem Statement

We target peak power consumption and maximum temperature issues in a multi-core MC system. Although there are works that try to manage or minimize the power consumption of MC systems as previously discussed in Section II, they do not consider the instantaneous peak power consumption in both HI and LO modes. In this regard, we propose a heuristic to manage peak power consumption and maximum temperature in both HI and LO modes of MC systems. The energy consumption, as an inherent effect, is also reduced. In order to achieve the goal, the most common approach is to exploit the dynamic slack resulted at runtime (due to the difference in the tasks' actual execution time and their WCET) to apply DVFS to the cores. In this case, the operating *V-f* level of each core can be changed based on the available dynamic slack to reduce peak power consumption. Nevertheless, the crucial research questions are (1) how to select the most appropriate tasks to assign the dynamic slack to; (2) if it is possible to re-map the tasks to other cores for better thermal control, where and when the tasks should be re-mapped to; (3) the algorithm execution time should be predictable and be as light-weight as possible to avoid interference with the actual tasks; and (4) the cost function used to select the task should not only be simple for calculation, but also sufficiently good to cover various metrics (peak power and temperature) and predict the possible impact of the task to the system in the near future. Our work presented here tries to address those.

B. Motivational Example

In this section, we are going to give a motivational example to clarify the problem and our solution in Fig. 1. As shown in Fig. 1a, the task graph is composed of eight tasks mapped on two cores. This graph represents a real-life application called Unmanned Air Vehicle control [12]. In this graph, the HC tasks are shown in light orange. We obtain the task mapping and scheduling table using the algorithm presented in [12]. In this example, we suppose that the system is only in the LO mode. It does not switch to the HI mode during runtime. Fig. 1b shows the system power trace at runtime without power management. In this example, we assume that the tasks consume their maximum power continuously during their executions. As shown in this figure, since the tasks may finish earlier than their WCET, the incurred slack can be exploited to assign 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 *V-f* level 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. In literature, there are some works that target such approach such as [22], but they target only the energy consumption; the tasks' deadlines are ignored. 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. In addition, we have 48.7%, 20.12% and 7.94% reduction in peak power, energy consumption and peak temperature compared to Fig. 1b.

C. Our Method

The goal of our proposed method is to manage the peak power consumption and maximum temperature of MC systems during runtime (see Eq. 2) in both criticality modes while guaranteeing tasks' deadlines (Eq. 3). In Eq. 3, for each mode, the execution time of each task i on the core j and at the *V-f* level l should not exceed the task deadline.

$$\forall \; timeslot \rightarrow optimize(\sum_{j \in Cores} P_j \;\&\; T_{max}) \quad (2)$$

$$\frac{C_i}{f_{jl}} \leq d_i \rightarrow \begin{cases} C_i = C_i^{LO} & \text{if } mode = LO \\ C_i = C_i^{HI} & \text{if } mode = HI \end{cases} \quad (3)$$

It is worth noting that the proposed method takes advantages of the run-time phase 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 as shown in Fig. 2. Details of the approach are provided below.

Figure 2: Overview of our proposed method.

Design-time Phase: The input to the algorithm is a task graph of which each task has two WCETs for LO and HI modes, maximum power consumption during its execution and the other parameters for the tasks explained in Section III-A and the number of cores available on the system. The power of the tasks can be obtained by MEET simulator [25]. More information about the usage of this simulator is given in Section V. Then, the static mapping and scheduling tables in two LO and HI modes are created by the algorithm presented in [12]. Researchers in this paper use the EDF algorithm to schedule tasks in each table which is related to the mode. Their goal is to execute more LC tasks in the HI mode and to compute the probability of LC tasks execution in this mode. Therefore, they try to present the best mapping of tasks on cores and also preempt the tasks to maximize the QoS by dropping as few LC tasks as possible. Initially, the system operates in the LO mode according to scheduling table of LO mode and if the system switches to the HI mode, the HI mode scheduling table is used to continue the execution of MC tasks till the end of the task set period. Eventually, these tables and the info associated with the tasks are then used at the run-time phase by our algorithm to manage the system.

Run-time Phase: Our proposed method consists of several function control units as shown in Fig. 2. The Scheduler Unit is the main unit which is communicating with the other units. Two main functions are supported in this unit: 1) Schedule the tasks according to the tables; 2) Change the scheduling and mapping of the tasks according to our proposed cost functions. When there is any free slack 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 not found, this unit will be called again in the next slot should there is any slack. If a proper task is selected, according to the core temperature and temperature of other cores at the considered time slot, the Scheduler Unit decides to re-map the task or not. After that, the obtained V-f level for the core at that time slot is stored (i.e., when the selected task will be executed). Due to the behavior of MC systems, the system switches to the HI mode if the execution of at least one task exceeds its defined C_i^{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 HI scheduling table which is generated at design-time [27]. The proposed cost functions and the algorithm are described as follows.

1) Selecting the Appropriate Task to Assign Slack to: Look-Ahead Unit, we consider an approach named look-ahead in which our algorithm chooses tasks after generated dynamic slack and also mapped on the same core in which the dynamic slack is generated. It should be mentioned that finding the optimum value for k will be discussed in Section V. Note, Eq. 4 is applied to a set of tasks that can release early. This equation is the general cost function denoted as CF_i which is computed for each task i. In this function, Pow_i and E_i are the maximum instantaneous power and maximum energy that a task consumes to execute during its execution time. In addition, α and β are in the range of [0,1]. Note that, due to this cost function, we expect that by considering $h(\alpha = 0, \beta = 1)$, the system has less peak power and more energy consumption as compared to considering $h(\alpha = 1, \beta = 0)$. Besides, energy reduction leads to significant decreasing in chip temperature [8]. Thus, we should have more temperature reduction by more energy saving. After selecting the task to use a specified slack time and decrease its V-f level, the start time, maximum power consumption and its WCET (C_i^{LO} or C_i^{HI}) are changed based on the size of generated slack time and its V-f level. As a result, start time and the deadline of tasks which are executed between the generated dynamic slack and selected task are changed based on the amount of slack.

$$CF_i = \alpha E_i + \beta Pow_i \quad (4)$$

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. Therefore, to decide about re-mapping the task and selecting the appropriate core to remap, we use the following cost function (Eq. 5). In this cost function, instead of using actual core temperature, we try to predict their temperature according to the accumulated energy. It is based on our observation that, a core tends to have lower temperature when its accumulated energy is less than the other cores. However, the difference between 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. Therefore, the proposed approach may change mapping during runtime.

$$CF_c = \sum_{t=1}^{t_f^*} \gamma E_c(t) \quad (5)$$

3) The Algorithm: The pseudo code of our proposed algorithm is outlined in Algorithm 1. At first, the algorithm gets the set of precedence constraints of 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 is set to the maximum value (line 1-3). This algorithm is done at each time slot (line 4-42). In each slot, the system checks whether switches to the HI mode or not (line 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 end of the period. In this situation, the V-f level of each core should be at first set to the maximum value to meet the deadline of HC tasks (line 7-8). The rest of the algorithm is executed in both modes.

If there is a dynamic slack during run-time, the algorithm should select the appropriate task to assign slack to that has more effect on instantaneous power consumption (line 10-35). This dynamic slack is generated if a task finishes its execution 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 which can be used. Therefore, if there is dynamic

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 PPR REDUCTION
 5:     if each Task executes more than C_i^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 finish its execution then
12:             Slack ← C_i^MO - ACT_i;
13:         else if there is an idle time in a core then
14:             Slack ← amount of the idle time
15:         end if
16:         T_S, T_P ← 0
17:         for n = 1 to k do
18:             T_P ← n_th after generated slack;
19:             if CF_T_S < CF_T_P and T_P can start earlier then
20:                 T_S ← T_P, n_s ← n;
21:             end if
22:         end for
23:         if n_s > 0 then
24:             Update the Freq_T_S^MO, C_T_S^MO and Pow_T_S^MO
25:             for n = 1 to n_s do Update the St_Task_LA_n^MO
26:             end for
27:             Core_S ← Core_T_S, Flag_remap ← 0
28:             for j = 1 to #Cores do
29:                 if CF_Core_S < CF_j then
30:                     Core_S ← C_j, Flag_remap ← 1;
31:                 end if
32:             end for
33:             if Flag_remap == 1 then Re-Map T_S on Core_S;
34:             end if
35:         end if
36:     end if
37:     for each task i do
38:         if St_i^MO = Tslot+1 then
39:             Update V-f level of the core based on Freq_i^MO
40:         end if
41:     end for
42: end procedure
```

slack, we first compute the amount of available slack (line 11-15). Now, we should select the appropriate task between k tasks that can be released early due to the slack time after reclaimable slack (line 17-22) based on the cost function. After determining the proper task, according to the system mode situation, frequency, WCET and maximum power consumption of the optimum task are changed according to the amount of slack (line 24). 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 (line 25-26). Now, the re-mapping technique is applied if the core in which the task will be run, has a higher temperature than others (line 27-34). As a result, it is possible to re-map the selected task to a core according to cost function (Eq. 5). At the end of each slot, if the start time if a task is the next slot, the V-f level of the core that task is mapped on it will be changed for the next time slot according to defined frequency scaling factor (line 37-41).

D. Complexity Analysis

In this subsection, we describe the time complexity overhead of our algorithm. We define n, c, and k as the number of tasks, the number of cores and the number of tasks looking ahead, respectively. In Algorithm 1, at first, the system is checked if needed to switch to the HI mode and change the frequency of all cores. Therefore, in the worst case situation, due to changing the frequency of cores only once, this operation is performed in $O(c)$. If there is dynamic slack, we select the proper task between k numbers of tasks look ahead in which, this search can be done in $O(k)$. After selecting the proper task, the start time of the selected task and the non-executed tasks if existed should be changed, which is done in $O(k)$ in the worst case. Finally, the algorithm changes the frequency of cores before each ready task starts its execution. This step is also done in $O(c)$. Therefore, the order of the algorithm is $\max\{O(c); O(k); O(n)\}$.

V. Experiments

In the experiments, we use random applications (task graphs) generated by the tool in [12]. An example of a real-life application is already given in our motivational Section IV-B. There are five basic parameters: c (number of cores), U (system utilization), d (edge percentage, the probability of having outward edges from one task to the others) and n (number of tasks). U=c is the normalized system utilization that refers to both LC and HC tasks with their predefined C^{HI}. Table II provides the configurations for different scenarios used in the experiments (unless stated otherwise in some specific ones). There are four scenarios where each of the previously described parameters is varied while the others are fixed. The purpose is to properly analyze how the proposed approach reacts to each parameter. In the case of varying U=c, we restrict the number of cores to 4 and the number of tasks to 40. The reason is that, when U=c 0:75, [12] fails to schedule the task graphs with higher c and n. Similarly, for the fourth scenario (varying d), [12] is not able to schedule a large number of tasks and cores with high dependency, i.e., d = 20%. For the case when the number of tasks is varied, if we use a larger number of cores, the number of tasks mapped to each core would be less. The effect of looking multiple tasks ahead cannot be examined fully.

The Pow_Range parameter is the range of power that the tasks may consume. These power values are generated randomly following the normal distribution within each task graph. In order to have a realistic possible range of power values, we use the MEET Simulator [25] with several embedded benchmarks from MiBench suite [28]. The MEET simulator is configured to model the ARM core (ARM7TDMI) running at 66 MHz. We consider the power consumption of the system as the sum of the power consumption of all cores [5].

Since our proposed method works at run-time phase, the randomly generated applications are executed in our Matlab-

Table II: Experiment Configurations

Param.	Varying c	Varying U/c	Varying n	Varying d
c (#core)	2, 4, 8, 16	4	4	4
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)	80	40	20, 30, 40, 50, 80	40
Pow_Range	[456.77, 833.33]mW			

based simulator. The scheduling tables for LO and HI modes determined at design-time and the task graphs are taken as inputs by the simulator. The actual run time of a task follows the normal distribution of which the mean and standard deviation is $\frac{3 \cdot C^{LO}}{4}$ and $\frac{C^{LO}}{12}$ respectively. The transition from LO to HI mode is simulated by forcing a randomly selected HC task to execute beyond its C^{LO}, i.e, within the range $(C^{LO}; C^{HI}]$. The scheduler is called whenever a task finishes or when its execution time exceeds its C^{LO}. After switching to the HI mode, the simulation continues until the end of the application period. These behaviors may change in some experiments for different comparison purposes.

The processor with 2, 4, 8 and 16 cores are arranged in the $2 \times 1, 2 \times 2, 2 \times 4$ and 4×4 floorplans respectively. Our approach does not have to probe the cores temperature to make decision. Therefore, during the simulations of the task graphs, the power values of cores depending on the running tasks are recorded. After that, HOTSPOT [24] is used to obtain the cores temperature throughout the execution. In addition, the effect of changing V-f levels is modeled by scaling the frequency within the range $[0.4; 1]$ [9].

The results are compared against [12] and [23]. The work [12] 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 [23] 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.

A. The effect of varying α, β

At first, we evaluate the results for different values of α and β in Eq. 4. The experiments are carried out for a system with $c = 8$, $U = c \cdot [0.5; 0.75]$, $d = 1\%$ and $n = 30$. The average results (Fig. 3) are obtained for a set of 100 task graphs with different $\alpha, \beta = \{0, 1\}, \{0.25, 0.75\}, \{0.5, 0.5\}, \{0.75, 0.25\}$ and $\{1, 0\}$. The results are normalized to [12]. In this section, to show the effect of varying these two parameters, tasks are executed with their actual 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 α or β is confirmed in the experiments. For example, the average normalized peak power is progressively reduced when α increases from 0 to 1 as presented in Fig. 3a. Similarly, in Fig. 3b, the higher the β, the lower the energy consumption and peak temperature. Finally, as the algorithm looks further ahead in the future to find the best tasks to 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 make sure that the effect of α, β is not skewed by another optimization.

In the rest of this paper, we consider $\alpha, \beta = \{0.5, 0.5\}$ that balances both peak power and temperature average reduction in comparison with other values of α, β.

B. Our proposed method in comparison with different task execution scenarios

In this experiment, we evaluate the quality of our proposed method against the situations where no online management techniques is used. We derive three cases with three assumptions about the task actual execution time at run-time phase and how dynamic slack, if any, are exploited.

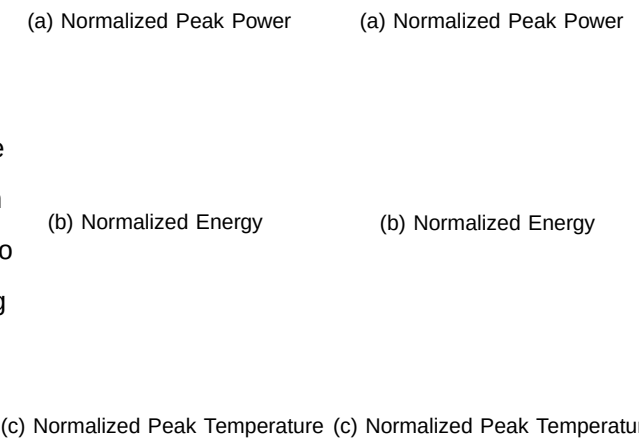


Figure 3: Impact of varying α and β on peak power, energy and MAX temperature.

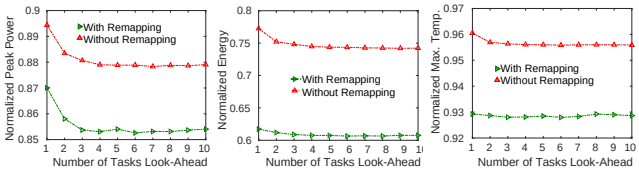
Figure 4: Ours normalized to [12] under scenarios WCNR, ACNR and ACR.

WCNR (Worst-Case, No-early-Release): All tasks are executed until their WCETs (C^{LO} in the LO mode and C^{HI} in the HI mode).
ACNR (Actual-Case, No-early-Release): All tasks are executed with their actual execution times. However, the incurred dynamic slack is not exploited, i.e., the tasks are not released earlier than their start times determined at design-time phase.
ACR (Actual-Case, early-Release): All tasks are executed with their actual execution times. The immediately following tasks which are ready can be released earlier than their presumed start times but without DVFS.

The system parameters used in this experiment are $c = 8$, $U = c \cdot [0.5; 0.75]$, $d = 1\%$ and $n = 30$. The results obtained from our approach are normalized against the ones from WCNR, ACNR and ACR. They are illustrated in Fig. 4. As WCNR, the system profile in terms of power, energy and temperature is expected to be the worst due to the pessimistic analysis on task execution time. When a more realistic task execution time is taken into account as in the cases of ACNR and ACR, the system profile is better. However, it can be seen that our approach, even without task re-mapping, does improve the system profile further quite significantly. The best result is achieved when our algorithm looks five tasks ahead when the peak power, energy and peak temperature are reduced by 12.7%, 25.1% and 4.7% compared to ACR. In the rest of this paper, due to not exploiting dynamic slacks at run-time phase in presented method of [12], we use ACNR scenario to have a fair comparison.

C. The optimum number of tasks to look ahead and the effect of task re-mapping

In this subsection, we would like to analyze the optimum number of tasks to look ahead by evaluating 1) the respective average quality of results; and 2) the latency due to extra processing for more tasks looking ahead. The number of tasks look-ahead is varied from 1 to 10. The effect of task re-mapping is also assessed in this section.



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

Figure 5: Normalized improvement in peak power, energy and MAX temperature for all scenarios shown in Table II.

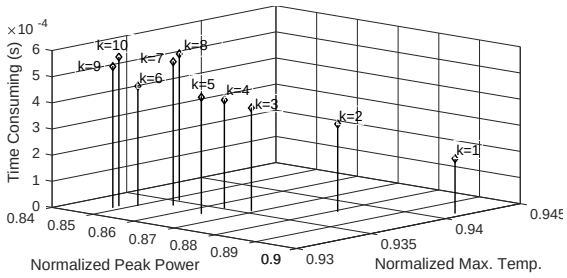


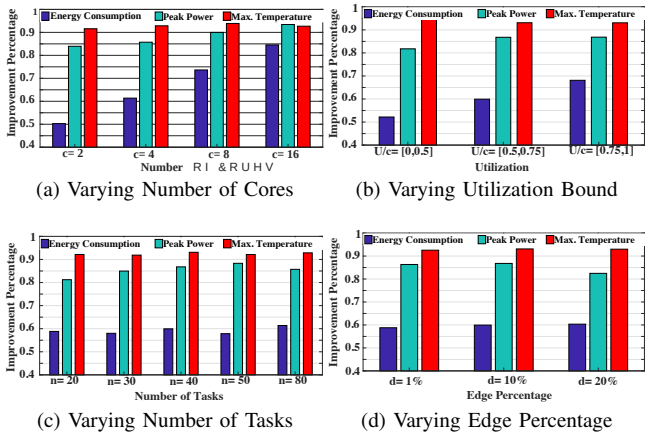
Figure 6: Time overhead when using different numbers of task look-ahead (k).

The results presented in Fig. 5 are obtained from all scenarios described in Table II. As a result, looking ahead 4 tasks provides a significant reduction in peak power and also in maximum temperature and energy consumption with and without task re-mapping. 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 [12] and 4.2%, 16% and 3.1%, respectively compared to [23].

Besides, we evaluate the time overhead for different number of tasks look-ahead. We measure this time by using the timer provided by Matlab. The values obtained by this way of measurement may not be applicable for an actual embedded system. However, the relative relationship between different numbers of task look-ahead can still be considerable. As an example, we show the average time for d= 1%, c= 4, U/c= [0.5,0.75] and n= 40 with 100 task graphs. The results are given in Fig. 6 where k represents the number of tasks look-ahead. It can be seen that, when k = 4, despite some increment in time overhead, the peak power and temperature is reduced significantly compared to when $k \leq 3$. The improvements are less pronounced when $k > 4$, especially when the time overhead is taken into account. Therefore, in the following experiments, we decide to use k = 4 with task re-mapping.

D. The analysis of the proposed algorithm for the scenarios shown in Table II

In order to illustrate how effective our proposed method is with different parameters, we analyze the results under four separate scenarios described in Table II. The results, which are normalized to [12], are provided in Fig. 7. In general, as the applications are getting more complicated (e.g., having a large number of tasks or cores, high inter-task dependency or system utilization), it is harder to achieve significant saving in peak power, energy. However, there is



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

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

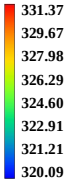
Figure 7: The improvements in peak power, energy and MAX temperature for each scenario described in Table II.

only a slight variation in the reduction of the maximum temperature across all scenarios. It is thanks to our task re-mapping technique where the tasks are re-distributed more evenly to the cores at run-time based on their accumulated energy.

For the case of varying the number of cores, since our method only tries to optimize the peak power for each core individually to reduce the time overhead, it is more difficult to maintain similar system peak power reduction when c is low. Nevertheless, as illustrated in Fig. 7a, the difference is quite marginal. The reduction is worse by only about 2% as c increases gradually from 2 to 16 cores. On average, the peak power and energy consumption is reduced by 11:5% and 32:5% 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. 7b, 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 13%, and up to 18%. 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 run-time and WCETs are inherently small. However, as seen in Fig. 7c, our method manages to reduce the peak power, energy and maximum temperature by 14:6%, 40:6% and 7:4% correspondingly.

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 precedence 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



(a) [12] with d= 20% (b) d= 20% and k= 1 (c) d= 20% and k= 4

Figure 8: Temperature profile for different edge percentage.

peak power of the whole system is consequently reduced. It can be seen in Fig. 7d that, when $d = 20\%$, the best peak power reduction is achieved compared the cases where $d = 1\%$ and $d = 10\%$.

Fig. 8 shows an example stead-state heat map of the systems with different edge percentage parameter. The result is obtained for system with $c = 16$, $U{=}c = 0{:}5$ and $n = 80$. We show the results of looking one and four tasks ahead as compared to [12]. It can be observed that, our approach not only reduces the maximum temperature, but also helps in balancing the difference in temperature between the cores, especially when $k = 4$.

VI. Conclusion and Future Work

In this paper, we study online peak power and peak temperature reduction in multi-core mixed-criticality embedded systems. Our presented method uses re-mapping technique and DVFS at runtime whenever there is a dynamic slack. We also propose 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 have evaluated the method in both low and high-criticality modes.

As future research, we would consider the management of peak power consumption and thermal cycling issues by considering the whole system instead of greedily optimizing individual cores. Furthermore, we will try to have a real implementation of proposed method to evaluate it.

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. ECRTS*, 2012.

[2] H. Su and D. Zhu, "An elastic mixed-criticality task model and its scheduling algorithm," in *Proc. DATE*, 2013.

[3] Z. Guo, L. Santinelli, and K. Yang, "Edf schedulability analysis on mixed-criticality systems with permitted failure probability," in *Proc. RTCSA*. IEEE, 2015.

[4] M. A. Awan, D. Masson, and E. Tovar, "Energy efficient mapping of mixed criticality applications on unrelated heterogeneous multicore platforms," in *Proc. SIES*, 2016.

[5] W. Munawar, H. Khdr, S. Pagani, M. Shafique, J.-J. Chen, and J. Henkel, "Peak power management for scheduling real-time tasks on heterogeneous many-core systems," in *Proc. ICPADS*, 2014.

[6] J. Lee, B. Yun, and K. G. Shin, "Reducing peak power consumption in multi-core systems without violating real-time constraints," *IEEE TPDS*, vol. 25, no. 4, 2014.

[7] Z. Li, X. Hua, C. Guo, and S. Ren, "Empirical study of energy minimization issues for mixed-criticality systems with reliability constraint," in *Proc. LPDC*, 2014.

[8] P. Huang, P. Kumar, G. Giannopoulou, and L. Thiele, "Energy efficient dvfs scheduling for mixed-criticality systems," in *Proc. of EMSOFT*, 2014.

[9] Z. Li, C. Guo, X. Hua, and S. Ren, "Reliability guaranteed energy minimization on mixed-criticality systems," *Elsevier JSS*, vol. 112, 2016.

[10] A. Taherin, M. Salehi, and A. Ejlali, "Reliability-aware energy management in mixed-criticality systems," *IEEE TSUSC*, vol. 3, no. 3, 2018.

[11] S. Baruah, H. Li, and L. Stougie, "Towards the design of certifiable mixed-criticality systems," in *Proc. RTAS*, 2010.

[12] R. Medina, E. Borde, and L. Pautet, "Availability enhancement and analysis for mixed-criticality systems on multi-core," in *Proc. DATE*, 2018.

[13] B. Lee, J. Kim, Y. Jeung, and J. Chong, "Peak power reduction methodology for multi-core systems," in *Proc. ISOCC*, 2010.

[14] 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 TPDS*, vol. 30, no. 1, 2019.

[15] T. Li, T. Zhang, G. Yu, Y. Zhang, and J. Song, "Ta-mcf: Thermal-aware fluid scheduling for mixed-criticality system," *JCSC*, vol. 28, no. 02, 2019.

[16] 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.

[17] T. Chantem, X. S. Hu, and R. P. Dick, "Temperature-aware scheduling and assignment for hard real-time applications on mpsocs," *IEEE TVLSI*, vol. 19, no. 10, 2011.

[18] M. Qiu, J. Niu, F. Pan, Y. Chen, and Y. Zhu, "Peak temperature minimization for embedded systems with dvs transition overhead consideration," in *Proc. HPCC & ICESS*, 2012.

[19] 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. RSP*, 2014.

[20] R. Kabir and B. Izadi, "Temperature and energy aware scheduling of heterogeneous processors," in *Proc. IC3*, 2016.

[21] 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. DAC*, 2009.

[22] J. Kang and S. Ranka, "Dynamic slack allocation algorithms for energy minimization on parallel machines," *Elsevier JPDC*, vol. 70, no. 5, 2010.

[23] D. Zhu, R. Melhem, and B. R. Childers, "Scheduling with dynamic voltage/speed adjustment using slack reclamation in multiprocessor real-time systems," *IEEE TPDS*, vol. 14, no. 7, 2003.

[24] W. 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 TVLSI*, vol. 14, no. 5, 2006.

[25] M. Bazzaz, M. Salehi, and A. Ejlali, "An accurate instruction-level energy estimation model and tool for embedded systems," *IEEE TIM*, vol. 62, no. 7, 2013.

[26] R. Grgen *et al.*, "CONTREX: design of embedded mixed-criticality CONTRol systems under consideration of EXtra-functional properties," in *Proc. DSD*, 2016.

[27] S. Baruah, "The federated scheduling of systems of mixed-criticality sporadic dag tasks," in *Proc. RTSS*, 2016.

[28] 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 WWC-4*, 2001.