Received 12 July, 2022; revised 31 August, 2022; accepted 12 September, 2022; Date of publication XX Month, 2022; date of current version XX Month, 2022.

Digital Object Identifier 10.1109/OJCS.2022.1234567

Peak-Power Aware Life-time Reliability Improvement in Fault-Tolerant Mixed-Criticality Systems

Mozhgan Navardi[1], Behnaz Ranjbar[1,2], Nezam Rohbani[3], Alireza Ejlali[1], AND Akash Kumar[2], *Senior Member, IEEE*

[1]Department of Computer Engineering, Sharif University of Technology, Tehran 11365-11155, Iran
[2]Chair of Processor Design, CFAED, Technische Universität (TU) Dresden, Dresden 01062, Germany
[3]School of Computer Science, Institute for Research in Fundamental Sciences (IPM), Tehran 11365-11155, Iran

CORRESPONDING AUTHOR: Behnaz Ranjbar (e-mail: behnaz.ranjbar@tu-dresden.de).

ABSTRACT *Mixed-Criticality Systems* (MCSs) include tasks with multiple levels of criticality and different modes of operation. These systems bring benefits such as energy and resource saving while ensuring safe operation. However, management of available resources in order to achieve high utilization, low power consumption, and required reliability level is challenging in MCSs. In many cases, there is a trade-off between these goals. For instance, although using fault-tolerance techniques, such as replication, leads to improving the timing reliability, it increases power consumption and can threaten life-time reliability. In this work, we introduce an approach named ***L**ife-time **P**eak **P**ower management in **M**ixed-**C**riticality systems* (LPP-MC) to guarantee reliability, along with peak power reduction. This approach maps the tasks using a novel metric called *Reliability-Power Metric* (RPM). The LPP-MC approach uses this metric to balance the power consumption of different processor cores and to improve the life-time of a chip. Moreover, to guarantee the timing reliability of MCSs, a fault-tolerance technique, called task re-execution, is utilized in this approach. We evaluate the proposed approach by a real avionics task set, and various synthetic task sets. The experimental results show that the proposed approach mitigates the aging rate and reduces peak power by up to 20.6% and 17.6%, respectively, compared to state-of-the-art.

INDEX TERMS Mixed-Critically Embedded Systems, Life-time Reliability, Peak Power Management, Fault-Tolerance, Multi-Core Processor.

I. Introduction

The constraint on the power consumption and fabrication cost has triggered a growing trend towards implementing embedded systems with multiple functionalities on a shared platform [1]–[3]. Such systems that have functionalities with different criticality levels concerning how important they are to the application, are called *Mixed-Criticality Systems* (MCSs) [1]–[4]. The number of processing cores is rising in such systems that results in boosting computing capacity [4]. However, increasing the number of processing cores and aggressive downscaling of feature size have resulted in higher overall power dissipation and power density, respectively, and consequently, elevate processor temperature [5], [6].

Different tasks running on an MCS require different reliability levels corresponding to their criticality [2], [7]–[9]. For example, in avionics applications, the flight management system includes high criticality (HI-Crit) tasks, like the engine operation (i.e., the task that ensures the safe operation). Failure during these task's executions may lead to system failure, and cause catastrophic damage. These tasks should be executed with higher reliability or execution time in comparison with low criticality (LO-Crit) tasks like air conditioning. By executing these LO-Crit tasks, the system accomplishes its mission successfully. There are defined standards like DO-178B [10] that provide reliability metrics for each task in each level of criticality in the MCSs [7], [8]. DO-178B considers the *Probability of Failure per Hour* (PFH) as a reliability metric based on transient faults.

A large number of previous works consider task re-execution or replication [2], [3], [7], [11] and task migration [8], [12]–[14] to confront with transient and permanent faults in MCSs, respectively. Regarding the transient fault

tolerating, when a task is executed, if an instance of the task failed due to a transient fault occurrence, another instance of the task is executed to meet the required reliability level with higher probability. On the other hand, if a core failed permanently, HI-Crit tasks are migrated and re-mapped on an active core. Hence, HI-Crit tasks are executed correctly before their deadline. Although the downside of these approaches is timing overhead, they guarantee required reliability; however, there is no improvement on life-time reliability. As a result, to the best of our knowledge, there is no proposed solution to address these issues in mixed-criticality systems: 1) considering both transient and permanent faults, 2) mitigating aging, i.e., improving life-time reliability. These approaches would be beneficial to apply in real-life avionic applications, like satellites and airplanes, where functions have different criticality levels, and LO-Crit tasks can be dropped in an emergency situation in favor of HI-Crit tasks to have a safe operation. Besides, life-time reliability management would be essential while designing such systems.

From the perspective of MC system operation, each system operates in different criticality modes. At first, the system starts its operation in the *Low* (LO) mode and in this mode, all tasks must finish their execution correctly before their deadlines. However, as an example, if a fault occurs and leads to exceeding the *Worst-Case Execution Time* (WCET) of HI-Crit tasks, these tasks overrun and consequently, the system switches from LO mode to the *High* (HI) mode. In this HI mode, we have to guarantee the correct execution of HI-Crit tasks before their deadlines to prevent catastrophic consequences [1]–[3]. In the worst-case scenario, executing all HI-Crit and LO-Crit tasks up to their WCET in the HI mode, requires higher computational demands, which may increase the processor's utilization and causes the cores to be activated for some time with the highest performance. Thus, all cores may execute tasks simultaneously to meet the task deadlines, which draw a significantly larger power than it is designed for and may increase the system instantaneous power consumption [15]–[17]. In this situation, the system is more likely to generate heat beyond its cooling capacity. Therefore, it will be more susceptible to failures [17]. To overcome this power consumption issue, *Dynamic Voltage and Frequency Scaling* (DVFS) is one of the outstanding technique [18]. However, there are some restrictions in managing the power of MCSs while using the DVFS technique. For example, applying DVFS leads to high switching time overhead due to changing the cores' voltage and frequency levels. Thus, it may cause deadline violation and also threaten the reliability of tasks [16]. Moreover, the DVFS technique increases transient fault rate probability [18], and therefore, it cannot be simply applied in different operational modes of MCSs, especially in the HI mode. On the other hand, the power budgeting strategies such as *Thermal Design Power* (TDP) [17], [19] are not efficient on permanent faults. The reason is that although these techniques reduce chip aging and prevent failures caused by extreme power density and generating unexpected heat, they do not propose any policy such as aging balancing to mitigate chip failures due to permanent faults. Note that TDP refers to the maximum power that can be safely dissipated by a chip. By exceeding the TDP (peak power constraint), a large amount of heat will be generated that may exceed the cooling capability of the chip and hence activate dynamic thermal management [15], [17]. However, although the system in which TDP constraint is met, some processing cores may experience higher temperature and age faster [5]. Based on the *International Roadmap for Devices and Systems* (IRDS) 2020 reports [20], aggressive down-scaling of feature size and inefficiency of power budgeting strategies on reliability, accelerate aging in recent and future technology sizes.

Due to the effect of temperature on aging rate and consequently on failure rate, temperature management with long-term perspective has an important effect on MCSs' reliability. This work proposes an approach named ***L**ife-time **P**eak **P**ower management in **M**ixed-**C**riticality systems* (LPP-MC) to prolong the life-time of MCSs. To balance the aging rate of processing cores, that results in system life-time extension, we propose *Reliability-Power Metric* (RPM). According to RPM, the proposed algorithm selects the appropriate region with lower aging rate and less power density to map tasks with different criticality levels. After mapping tasks, we apply re-execution fault-tolerance technique to guarantee that tasks execute correctly before their deadline by assuming permanent and transient faults. Then, *Earliest Deadline First with Virtual Deadline* (EDF-VD) [9], [21] is used to schedule the tasks. In summary, the main contributions of this article are:

- Defining a new metric to evaluate the effect of power, utilization, and reliability w.r.t. a life-time target on aging rate, and then proposing an approach to balance stress condition on the processing cores in MCSs.
- Proposing an scheme for peak power management to avoid chip failures due to TDP violations in short-term.
- Considering both permanent and transient faults in MCSs to guarantee the timing reliability and to improve the life-time reliability.
- Calculating the required number of re-executions for each task to guarantee the reliability in each criticality level, based on both permanent and transient faults in an analytical approach.

To evaluate our approach, we compare it with the state-of-the-art task mapping approaches in MCSs [1], [22]. The experimental results show that LPP-MC extends life-time up to 20.6% and reduces peak power comparing to the existing approaches up to 17.6%.

The rest of the paper is structured as follows: related work in this area is presented in Section II. A brief overview of fundamental concepts is provided in Section III. The problem definition and motivational example are presented in Section IV. Then, in Section V, we describe the proposed

TABLE 1: Summary of state-of-the-art approaches.

#		(S)/(M) -Core	RT Tasks	MC Tasks	LC tasks' QoS	Peak Power	Avg. Power	Tra. Fault	Per. Fault	Life-time
1	Huang'14 [18], Ali'15 [23], Taherin'18 [24], Zhang'21a [25], Zhang'21b [26]	(S)-Core	✓	✓	×	×	✓	×	×	×
2	Haririan'15 [27], Digalwar'17 [28]	(M)-Core	✓	✓	×	×	✓	×	×	×
3	Ranjbar'19 [29], Ranjbar'21 [16]	(M)-Core	✓	✓	✓	✓	✓	×	×	×
4	Ranjbar'22 [15]	(M)-Core	✓	✓	✓	✓	×	✓	×	×
5	Saraswat'09 [12], Saraswat'10 [13], Liu'13 [14], Al'16 [8], Alahmad'17 [30]	(M)-Core	✓	✓	×	×	×	×	✓	×
6	Ranjbar'20 [9], Huang'14 [7], Al'16 [3]	(S)-Core	✓	✓	✓	×	×	✓	×	×
7	Zeng'16 [2], Caplan'18 [11], Rambo'21 [31]	(M)-Core	✓	✓	×	×	×	✓	×	×
8	Koc'19 [32], Choi'18 [33]	(M)-Core	✓	✓	✓	×	×	✓	×	×
9	Huang'11 [34]	(M)-Core	✓	×	×	×	×	×	×	✓
10	Das'14 [35]	(M)-Core	✓	×	×	✓	✓	✓	✓	✓
11	Das'16 [36]	(M)-Core	×	×	×	×	✓	×	×	✓
12	Haghbayan'17 [5]	(M)-Core	×	×	×	✓	×	×	×	✓
13	Das'13 [37]	(M)-Core	✓	×	×	×	×	✓	✓	✓
14	Lee'10 [38]	(M)-Core	×	×	×	✓	×	×	×	×
15	Lee'14 [39], Munawar'14 [17]	(M)-Core	✓	×	×	✓	×	×	×	×
16	Our Work (LPP-MC)	(M)-Core	✓	✓	✓	✓	×	✓	✓	✓

LPP-MC approach in detail. In the end, Section VI and VII describe experimental results and conclusion, respectively.

II. Related Work

The state-of-the-art research works on multi-core MCSs focus on three scopes: *utilization bound improvement*, *power management* and *reliability management*. Since our focus is on power and reliability management, we discussed the most related works within these scopes. These works are categorized in Multi(M)/Single(S)-Core, Real-Time (RT), and Mixed-Criticality (MC) task model from the perspective of different objective optimization, such as providing Quality of Service (QoS) for the LO-Crit tasks in the HI mode, peak or average power consumption management, transient fault, permanent fault and life-time, which are summarized in TABLE 1. The research works in the scope of power management are given in rows 1-4 of TABLE 1. Rows 5-8 of the table show works on reliability management, specifically timing reliability. The works given in rows 9-15 are implemented on non-MCSs with life-time reliability and peak power objectives. A brief explanation of these works is given below.

Power Management: There are some works in the context of power management in MCSs [18], [23]–[28], [40], in which, the authors have applied DVFS to minimize energy consumption and have dropped LO-Crit tasks in HI mode (rows 1-2). Researchers in [41] give a comprehensive study in the field of power and energy-aware task scheduling in MCSs. However, there is a lack of *peak power reduction* in some research works, like what mentioned in rows 1-2 and no guarantee to meet TDP. In [16], [29], the authors have minimized peak power consumption by employing the accumulated dynamic slack and then using the DVFS technique at run-time for mixed-criticality systems with no guarantee of meeting the TDP constraint (row 3). However, since the DVFS technique increases the rate of fault occurrence [18] and there is no method in order to tolerate transient faults, reliability management cannot be guaranteed. Besides, these authors have presented a design-time approach to manage peak power and transient fault occurrence in MCSs by finding different scenarios of task mapping (row 4). However, this presented approach does not guarantee the reliability requirements and improves the life-time reliability and average power consumption.

Reliability Management: The reliability of MCSs has been considered by researchers as well. All techniques that were presented in [8], [12]–[14] have considered permanent faults as the fault model that have guaranteed reliability of the HI-Crit tasks by migration and reallocation. If there is no appropriate core to migrate HI-Crit tasks, they will drop LO-Crit tasks (row 5). Besides, researchers in [30] have proposed redundancy-based task mapping and scheduling to address the reliability improvement; however since the task's replicas are mapped to different types of processors, like safe and regular, there is no guarantee on correct execution of HI-Crit tasks in the case of failing of safe processors. Researchers in [42] give a comprehensive study in the field of reliability management for multi-core MCSs. Besides, there are some works that have considered transient faults as well [2], [3], [7], [9], [11], [32], [43]. Huang et al. [7], have applied re-execution fault-tolerance technique to MCSs. They have proposed an analytical approach to model safety requirements for both HI-Crit and LO-Crit tasks in single-core MCSs. Ranjbar et al. [9] also have used re-execution to meet reliability requirement. Moreover, they have considered maximum allowable number of drops for LO-Crit tasks in HI mode. Researchers in [3] have defined a four-mode model for single-core MCSs and optimized the LO-Crit tasks' quality of service (row 6). Later, Caplan et al. [11], have proposed an algorithm for this model in multi-core MCSs. Also, based on

the proposed approach in [7], Zeng et al. [2] have proposed a multi-core fault-tolerance scheduling algorithm. They have used replication besides re-execution (row 7). [31] has also presented a replica-aware co-scheduling method for MCSs by exploiting cross-layer fault tolerance mechanisms. This method has supported network-on-chip communication delay and replication management overheads. However, there is no guarantee on correct execution of LO-Crit tasks in the HI mode, which may lead the system not to do its mission efficiently. Researchers in [32], [33] have proposed mapping algorithm for multi-core MCSs and guarantee a level of QoS for LO-Crit tasks (row 8). The disadvantage of this set of approaches is that they can no longer be used if a core fails. Hence, life-time of each core plays a crucial role in the efficiency of these works that is one of the main drawbacks of these approaches. Some other previous works focus on single-core MCSs reliability improvement [3], [7], [9], [44]. Although scheduling on single-core architectures is more straight forward, however, since multi-core architectures are more power-efficient and flexible in task scheduling, almost all of the recent MCSs exploit multi-core architectures and many of the state-of-the-art researches are focusing on these systems.

There are previous works in the context of *life-time* (rows 9-13) and *peak power* (rows 14-15) which are implemented for non real-time systems or systems with one level of criticality. In the following, a summary of these works is mentioned: 1) *Life-time Improvement*: A task allocation and scheduling for real-time applications are presented in [34]. This work has presented an analytical model in order to estimate the life-time (row 9). Das et al. [35], have considered multi-objective optimization problem including energy and reliability for real-time applications (row 10). Also, Das et al. [36], propose a mapping and scheduling for *Synchronous Data Flow Graphs (SDFGs)* to improve life-time and power consumption (row 11). The work in [5], has applied a two-step resource management approach for dynamic applications to improve life-time while meeting a power budget. Although the work presented in [5] considers throughput, there is no guarantee to complete the correct execution of tasks before their deadline. Moreover, they are not concerned about transient faults (row 12). Besides, researchers in [37] have considered trade-off between transient and permanent faults in resource allocation. However, there is no guarantee to manage the real-time constraints (row 13). 2) *Peak Power Reduction*: The first work on peak power control has been presented in [38]. They have considered task graph model and there is no constraint on the task execution time (row 14). Later the works in [17], [39], have assumed real-time tasks and propose an scheduling algorithm to reduce the peak power (row 15).

There is no policy for the HI and LO modes in the mentioned categories due to the existence of one operational mode in the traditional systems. If these approaches are applied in MCSs, the utilization will reduce extremely. The reason is that, there is a gap between *Worst-Case Execution Time* (WCET) and *Actual Execution Time* (AET) of a task. These algorithms consider WCET; hence this gap leads the cores to be underutilized [45]. On the contrary, considering two or more criticality modes (i,e, MCSs) results in decreasing the gap due to the close proximity of WCET and AET in lower modes by defining two WCETs for HI-Crit tasks, optimistic and pessimistic, and therefore, improving the system utilization [45]–[47].

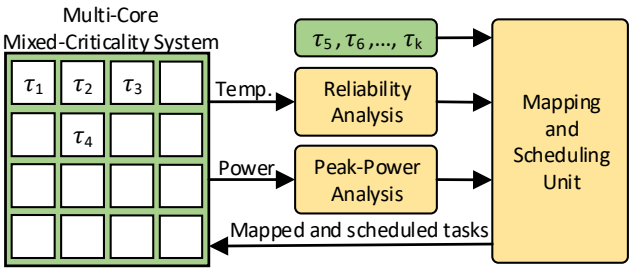


FIGURE 1: System overview

In many of the previous works on MCSs, the LO-Crit tasks are ignored in the HI mode and furthermore, none of them consider life-time and TDP. To the best of our knowledge, there is no algorithm that prolongs life-time along with considering the TDP constraints in MCSs, while LO-Crit tasks are executed in the HI mode by increasing their period (i.e., using service degradation) until all tasks are schedulable in the HI mode.

III. Preliminaries

In this section, the essential concepts to understand the proposed approach are introduced. For this aim, system and task models are explained. Then, we show how power, and reliability are modeled in this work.

A. System Overview and Application Model

FIGURE 1 presents our system overview in which tasks are executed on a multi-core system. Temperature and power consumption of the cores are utilized to analyse the cores reliability and manage peak power, respectively. The mapping and scheduling unit uses the output of the reliability analysis unit and peak power analysis unit. Then, it decides how to map and schedule the input tasks on the cores.

Application Model: Let $\tau = \{\tau_1, \tau_2, ..., \tau_n\}$ be a set of independent periodic tasks which execute on a multi-core systems with $m \times m$ square matrix layout ($\varsigma = \{\varsigma_{1,1}, \varsigma_{1,2}, ..., \varsigma_{i,j}, ..., \varsigma_{m,m}\}$ where $i, j \in \{1, 2, ..., m\}$). Each τ_k is characterized by $\{C_k, T_k, D_k, \chi\}$. C_k denotes the WCET of task τ_k. T_k is the period of task τ_k which is the minimum time amount between two released instances of the task. Note that the period amount is used to compute the task utilization bound which is used to check the task schedulability in the system ($u_k = \frac{C_k}{T_k}$). D_k is the implicit-

deadline[1] of task τ_k. Analogous to [1], [3], [22], we consider the former that deadline is equal to the period ($T_k = D_k$). Besides, χ is a criticality level of task τ_k ($\chi \in \{LO, HI\}$)[2] Also, τ_χ represents all tasks with $\chi_$Crit.

For mixed-criticality tasks, two levels of WCET are generally considered; *Low WCET* (LWCET, denoted by C_k^{LO}) and *High WCET* (HWCET, denoted by C_k^{HI}). We discuss in Section D, how these mixed-criticality tasks are analyzed and how these WCETs for different operational modes are computed.

Besides, the total utilization of χ_1_Crit tasks (τ_{χ_1}) in the χ_2 mode is denoted by $U_{\chi_1}^{\chi_2}$ and is calculated by $\sum_{\tau_k \in \tau_{\chi_1}} C_k^{\chi_2}/T_k$ [7], [18], [22].

System Operational Mode: Without loss of generality, we assume an MCS with two criticality modes: HI and LO [2], [7], [9]. At the beginning of time, all of the cores start in the LO mode. In this mode, if the execution time of any HI-Crit task exceeds its LWCET, the core enters to the HI mode [2]. It remains in this mode until there is no HI-Crit task in the execution queue of the core. To guarantee the HI-Crit tasks are executed before their deadlines, EDF-VD [21] (one of the most common scheduling algorithms for MCSs) is used to schedule tasks. Moreover, service degradation policy is considered for the LO-Crit tasks [2], [7]. The details are elaborated in Section E.

B. Power Model

The power consumed in each core $\varsigma_{i,j}$ includes static and dynamic power [18], [23], [27]:

$$P_{i,j}^{Core} = P_{i,j}^{Sta} + P_{i,j}^{Dyn} \tag{1}$$

The static power $P_{i,j}^{Sta}$ is dissipated due to sub-threshold leakage current and is approximately related to the number of transistors on the chip. The dynamic power $P_{i,j}^{Dyn}$ is akin to the switching activity α, internal nodes capacitance c, operating voltage v_{dd} and frequency f:

$$P_{i,j}^{Dyn} = \alpha_{i,j} \times c_{i,j} \times f_{i,j} \times (v_{dd})_{i,j}^2 \tag{2}$$

The total power consumption of cores at a given time is:

$$P^{Tot} = \sum_{i,j=1}^{Core\_Numbers} P_{i,j}^{Core} \tag{3}$$

A chip can withstand certain power threshold which is determined by the chip TDP. Therefore, the TDP constraint determines the total power consumption limit.

C. Fault Model and Fault Tolerance

We consider both transient and permanent faults in the proposed approach. We employ fault tolerance techniques to guarantee the reliability requirement of the MCS. To achieve this purpose, the DO-178B standard and furthermore, the *Probability-of-Failure-per-Hour* (PFH) metric are utilized to characterize the system reliability [2], [3], [9], [24]. As shown in TABLE 2, DO-178B introduces five levels of criticality A, B, C, D and E, which A provides the highest level of criticality and E is the lowest one. Usually, the HI-Crit tasks are classified in the levels A to C. The tasks that have criticality level D or level E are more LO-Crit tasks and QoS is more important for these tasks.

TABLE 2: DO-178B safety standard [7], [10]

χ	A	B	C	D	E
PFH(χ)	$<10^{-9}$	$<10^{-7}$	$<10^{-5}$	$\geqslant 10^{-3}$	-
Safety Impact	Catastrophic	Hazardous	Major	Minor	No effect

In the following, we first define reliability and *Probability-Of-Failure* (POF). Then, we present the transient and permanent fault models used in this work.

The system reliability is denoted by $R(t)$. $R(t)$ is the probability of the correct operation of the system during the period of time $[t_0, t]$ (i.e., in the operational phase), where t_0 refers to initial time that can be equal to 0. The PFH can be calculated as follows [48]:

$$PFH = POF/t,\ POF = 1 - R(t) \tag{4}$$

where t is time in hour and $1 \leqslant t \leqslant 10$ [7]. Without loss of generality, we assume $t = 1$, therefore $PFH = POF$.

Transient Fault Model: An instance of the task τ_k is not completed before its deadline, with the probability of failure f due to transient faults [7]. Note that an error detection mechanism, like ARGUS [49], is employed to check the correctness of the task's output when the task finishes its execution. ARGUS is one of the significant checker tools to detect errors, which can be applied to any embedded systems with low chip area overhead and check control flow, data-flow, computation, and memory access separately [15]. Hence, the timing overhead of error detection has been considered in the tasks' WCETs in our evaluation. We use re-execution technique for each task to deal with transient faults and meet the timing reliability requirements [2], [11]. In the case of fault occurrence, there is a timing overhead to discard the faulty result and resume the task execution. Since we have to guarantee the worst-case scenario, we consider the timing overhead in the WCETs of tasks as well. The number of re-execution task τ_k to guarantee the timing reliability requirement in χ criticality mode (LO or HI mode) is denoted by n_k^χ and is calculated in Eq. 5. FIGURE 2 shows how a task is released and re-executed in each period until duration time t. Each task instance τ_k is released in its period of T_k. In one period, we assume each instance is re-executed n_k times with the execution time of C_k, so $n_k \times C_k$ is the total execution time of task τ_k shown as a green rectangle during a time period. Also, if the probability of failure of task τ_k was f, it will be decreased to f^{n_k} in one period when re-executing the task for n_k times. On the other hand,

[1]Between two kind of deadline, implicit and constrained [22]

[2]Analogous to [3], [7], [11], we consider two criticality levels in this article.

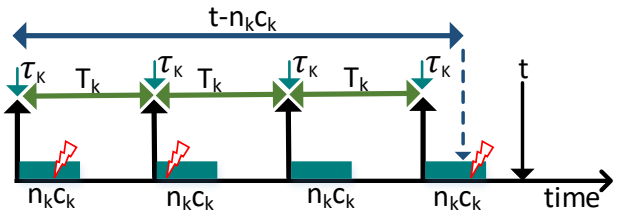


FIGURE 2: Number of instances of task τ_k until time t

if we want to calculate the probability of failure in time t, not a period, we need to take into account the total number of released instances in duration t, $(r(n_k, t))$, which is equal to four in the FIGURE 2. Hence, we can model PFH and consequently, transient reliability $R_k^{Tra}(t, \chi)$ for a LO-Crit or HI-Crit task τ_k as Eq. (5) [7]:

$$R_k^{Tra}(t,\chi) = 1 - PFH_k^{Tra}(t,\chi) = 1 - r(n_k^\chi, t) \times f^{n_k^\chi} \tag{5}$$

where n_k^χ is the re-execution number of task τ_k in criticality mode χ. t is the operating time in hour and $r(n_k^\chi, t)$ is the maximum number of instances of τ_k that can be executed until time t. With the help of FIGURE 2, this maximum number can be calculated as follows [7]:

$$r(n_k^\chi, t) = max\{\lfloor \frac{t - n_k^\chi \times C_k}{T_k} + 1 \rfloor, 0\} \tag{6}$$

The PFH of all χ_Crit tasks on a specific core $\varsigma_{i,j}$ in criticality mode χ is denoted by $PFH_{i,j}^{Tra}(t,\chi)$ and is calculated as follows [7]:

$$R_{i,j}^{Tra}(t,\chi) = 1 - PFH_{i,j}^{Tra}(t,\chi) = 1 - \sum_{k \in \chi} PFH_k^{Tra}(t,\chi) \tag{7}$$

where $PFH_k^{Tra}(t,\chi)$ is equal to $r(n_k^\chi, t) \times f^{n_k^\chi}$ based on the Eq. (5).

Permanent Fault Model: We adopt an analytical model similar to [5], [34] to calculate the permanent reliability of each core $(R_{i,j}^{Per}(t))$. Furthermore, we assume each core $\varsigma_{i,j}$ will permanently fail with probability of failure $PFH_{i,j}^{Per}$. Therefore, permanent reliability $R_{i,j}^{Per}(t)$ can be calculated as follow:

$$R_{i,j}^{Per}(t_c) = 1 - PFH_{i,j}^{Per}(t_c) = e^{-(A_{i,j})^\beta} \tag{8}$$

$$A_{i,j} = \sum_{s=1}^{t_c} t_s - t_{s-1}/\alpha(Temp) \tag{9}$$

where β is the Weibull slope parameter, t_s is a time slot, t_c is a current time and $Temp$ is the temperature in Kelvin. α is the aging rate that depends on temperature and wear-out mechanisms. In this work, we consider *Electromigration* (EM) as one of the most important aging causes [5], [34]:

$$\alpha^{EM}(Temp) = \frac{A_0 \times (J - J_{Crit})^{-n} \times e^{E_a/K.Temp}}{\Gamma(1 + \frac{1}{\beta})} \tag{10}$$

in which A_0 is material-dependent constant, J and J_{Crit} are current density and critical current density, respectively. n typically is equal to 2, E_a, K and Γ are the activation energy, Boltzmann's constant and gamma function, respectively.

The average life-time of the core $\varsigma_{i,j}$ is estimated by *Mean Time To Failure* (MTTF). By the first core failure, the system needs to be fixed or changed. Thus, the minimum of MTTF of the cores can be estimated as the system MTTF [34], [50].

$$MTTF = \int_0^\infty R_{i,j}^{Per}(t)dt \tag{11}$$

D. Mixed-Criticality Task Analysis

Analogous to [7], [9], we define the two execution time levels based on the reliability requirement level. As mentioned in previous section, Tolerating transient faults has been achieved by re-executing tasks. Therefore, this technique directly affects the execution time of task. Consequently, re-executing each task to guarantee its safety requirement would result in specific WCETs for each criticality level [7]. To guarantee the reliability, the number of re-executions (n_k) for each criticality level of tasks in each mode is obtained (n_k^{LO} for LO mode, and n_k^{HI} for HI mode), which has been discussed in previous section. Therefore, C_k^{LO} is the *Low WCET* (LWCET) of task τ_k and also C_k^{HI} is the *High WCET* (HWCET) of task τ_k. Now, the WCET of each task in each criticality level is computed as follows, where for each LO-Crit task, the LWCET C_k^{LO} is equal to the HWCET C_k^{HI} and n_k^{LO} is equal or lower than n_k^{HI} subject to the criticality level [7], [9]:

- LO_Crit tasks: $C_k^{LO} = C_k^{HI} = n_k^{LO} \times C_k$
- HI_Crit tasks: $\begin{cases} C_k^{LO} = n_k^{LO} \times C_k, \\ C_k^{HI} = n_k^{HI} \times C_k, \end{cases}$

E. Mixed-Criticality Scheduling Algorithm

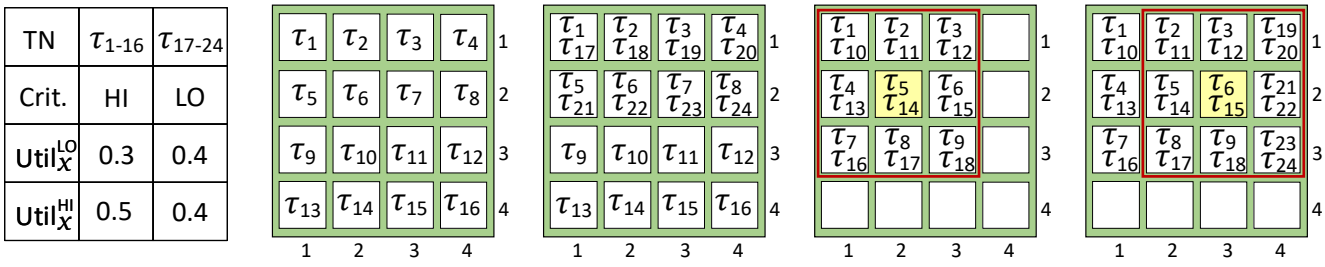
There are various algorithms for task scheduling in MCSs. One of the most efficient algorithms is EDF-VD [21]. EDF-VD has separate policies to schedule tasks in the two modes.

1) LO Mode Policy: The tasks are scheduled based on their LWCET and a *Virtual Deadline* (VD). In fact, the VD is defined to give higher priority to the HI-Crit tasks in the scheduling algorithm. The VD is obtained as follows: $VD = x \times D$, where $x = U_{HI}^{LO}/(1 - U_{LO}^{LO})$ [21].

2) HI Mode Policy: *Task Dropping* (TD) or *Service Degradation* (SD) are utilized to improve schedulability. When the system enters to the HI mode, it is possible that algorithms cannot schedule all tasks with their HWCET. In this case, EDF-VD does TD or SD for the LO-Crit tasks. The TD drops the LO-Crit tasks in the HI mode to guarantee the HI-Crit tasks meet their deadlines [47]. The SD extends the period of each LO-Crit task to reduce the number of executed LO-Crit tasks. For this aim, the period of LO-Crit task T converts to T' as follows: $T' = d_f \times T$ where $d_f > 1$ [7].

IV. Problem Definition and Motivation

This section defines the problem statement that is discussed in this work. Then, it shows the effectiveness of using *Region Selection* (RS) in solving the problem through a concrete



(a) Task set example (b) HI-Crit task mapping without RS (c) LO-Crit task mapping without RS (d) Task mapping with RS-Step1 (e) Task mapping with RS-Step2

FIGURE 3: Motivational example

example. After defining the problem and illustrating the example, we take advantage of RS to uniform aging rate of cores and power consumption in order to increase life-time and reduce peak power in mixed-criticality systems.

A. Problem Definition

This work deals with the life-time and peak power management in mixed-criticality systems. The proposed approach tries to balance the aging of cores and consequently increases the *life-time*. On the other hand, *reliability level* is satisfied by one of the fault tolerance techniques, re-execution. The problem can be formulated as follows:

Inputs: Given:

- The multi-core embedded system that the cores are assumed to be organized in a $m \times m$ square matrix layout. $\varsigma = \{\varsigma_{1,1}, \varsigma_{1,2}, ..., \varsigma_{m,m}\}$
- The mixed-criticality task set $\tau = \{\tau_1, \tau_2, ..., \tau_n\}$.
- The target reliability $R^{target}(t)$ and $t_{life-time}$.
- The system power constraint (TDP).
- The scheduling and aging parameters.

Outputs: Determining a mapping algorithm to allocate tasks to processing cores and a scheduling algorithm to schedule tasks on each processing core w.r.t the constraints.

Constraints: Reliability, power and schedulability are represented as main constraints which are introduced as follows.

- Reliability level constraint: 1) $R^{Per}(t) > R^{target}(t)$, and 2) $PFH^{Tra}(t, \chi) < PFH^{target}(\chi)$
- Power constraint: $P^{Tot} < TDP$
- Schedulability analysis: $U_{MC} \leqslant 1$

Objective: The main objective of this work is life-time improvement in MCSs.

B. Motivational Example

In this section, we give a motivational example to show the effect of RS technique. We show that selecting a region of processing cores for mapping the tasks reduces the number of active cores. Then, we take this advantage to reduce peak power and consequently avoid chip failure.

We implement CU-UDP algorithm [22] on a 4×4 multi-core platform in two different scenarios. First, we assume a scenario that we have 16 cores when implementing CU-UDP. Second, we use RS, in which, we have a subset of cores (nine cores in this example) when the tasks are mapped. The task set and timing parameters of our example are given in FIGURE 3a. CU-UDP sorts tasks unaware of task criticality. It considers U_{HI}^{HI} for HI-Crit tasks and U_{LO}^{LO} for LO-Crit tasks to sort the tasks in decreasing order of utilization. Then, it maps the HI-Crit tasks based on *Worst Fit* (WF) strategy subject to the $U_{HI}^{HI} - U_{HI}^{LO}$ and LO-Crit tasks based on *First Fit* (FF) strategy. In the first scenario, FIGURE 3b and 3c show the result of tasks mapping based on CU-UDP with 16 cores without RS that all available resources have been utilized.

In the second scenario, we apply RS technique. Each region has two properties; a central core that is named *First Node* (FN) and region radius r that $(2r + 1)^2$ cores exist around FN [5]. Therefore, there are four regions with radius $r = 1$ and $(2 + 1)^2 = 9$ cores in this example. In FIGURE 3d, CU-UDP maps the tasks on a region with nine cores until there is no available core for mapping the tasks on. Then, it switches to another region and keeps going to map all the tasks. The result in FIGURE 3e shows the task set is mapped on 12 cores by using RS. If we choose another region except for this region, we will have some idle cores too. As most of the mapping algorithms in MCSs use WF strategy, these algorithms may turn on all available cores. However, RS leads to have some idle cores and to manage peak power without time redundancy or increasing fault rate. For instance, DVFS manages peak power but it increases transient fault rate exponentially by reduction of operating voltage [48]. Although turning off four cores reduces peak power consumption, it may increase power density on the active cores. Hence, it is important to propose an algorithm to take into account the power density of the cores intelligently. If we do not utilize some cores unaware of reliability and temperature, it even can cause hotspot and accelerate aging [5]. However, if we manage RS and choose

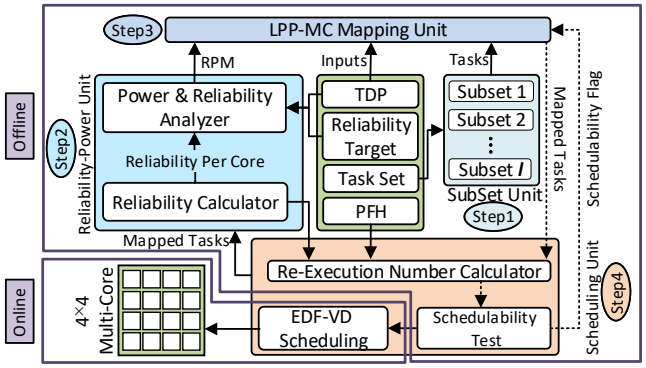


FIGURE 4: Proposed approach overview

a fresh region to map tasks, it results in an increased life-time.

V. Proposed Approach

In this section, we present an approach that uses RS to solve the problem. We first introduce our proposed approach which consists of four main units: the subset unit, the reliability power unit, the LPP-MC mapping unit and the scheduling unit. Then, we explain the implemented algorithm for each unit.

A. LPP-MC Approach

FIGURE 4 shows the overview of an MCS *operational phase* includes offline and online phases that utilizes LPP-MC. An operational phase defines as a period of time from the start of system operation to the end of it, which is usually between 1 to 10 hours due to the application (as mentioned in section C). In each operational phase, at first, the offline phase will be done then the online phase will start. In the offline phase, LPP-MC takes the task set, reliability target, PFH and TDP as inputs and generates the mapped tasks and tests the schedulability of tasks. Then, the mapped tasks are scheduled by EDF-VD in the online phase. In the first step, the subset unit divides the task set in l subsets. Then, the LPP-MC mapping unit selects an appropriate region by considering RPM and maps a subset of tasks on the selected region. Later, the scheduling unit tests schedulability and returns a flag. If the flag is set to zero (i.e., the task subset cannot be scheduled under the defined circumstances), LPP-MC will select another region to map the subset until the schedulability flag is set or LPP-MC failed. In parallel, the reliability-power unit updates RPM and the next subset is mapped by the LPP-MC mapping unit. All these calculations are done offline so the required time for task mapping is not critical. Further explanation of the units is presented as follows.

Subset Unit: In this unit given task set is divided into subsets. Since LPP-MC considers a subset of cores to map the tasks, it is necessary to map a subset of tasks on a subset of cores. Otherwise, it will be impossible to map all tasks of the task set on a subset of cores. We consider two main factors for dividing a task set into subsets: The sum of tasks utilization in task set and utilization level of a region of cores. By dividing these two factors we calculate the minimum number of subsets. For this aim, the subset unit sorts the tasks from the HI-Crit to the LO-Crit level to divide the task set into subsets. Then, it assigns one task of the task set to each subset until all subsets include one task. It repeats this algorithm for all tasks. In this situation, HI-Crit and LO-Crit tasks will distribute uniformly in different subsets. It should be mentioned that the proposed approach is applied in offline mode. Therefore, it has no knowledge about the online power trace of the system. However, we guaranty that total power consumption is always less than TDP. Hence, the task with the highest power consumption level is assumed as the power of the core. In this way, we can manage peak power in offline mode.

Reliability-Power Unit: The reliability calculator and the power and reliability analyzer are the main components of the reliability-power unit. Temperature is akin to the power consumption of each core which is given to the reliability calculator as an input to calculate core reliability. After calculating core reliability, the power & reliability analyzer uses it to calculate the region selection metric. In this work, as explained in section I, we propose a new metric named *Reliability-Power Metric* (RPM) given in Eq. (12). It uses the concept of *Reliability Factor* (RF) which has been presented for the first time in [5]. The proposed technique in [5] minimizes the communication latencies of interconnected tasks belonging to the same application and manages the life-time; while, in our proposed method (RPM), we minimize the number of active cores to reduce peak power along with life-time management. The RPM is defined for all available FN cores $\varsigma_{w,h}$ as follows:

$$RPM^{r}_{w,h} = \sum_{i=w-r}^{w+r} \sum_{j=h-r}^{h+r} r^{Per}_{i,j} \times (1-p_{i,j}) \times (1-u_{i,j}) \quad (12)$$

where, $RPM^{r}_{w,h}$ is RPM for the FN $\varsigma_{w,h}$ and radius r on a square region. This region includes $(2r+1)^2$ cores $\varsigma_{i,j}$. $u_{i,j}$ is the core $\varsigma_{i,j}$ utilization and $1-u_{i,j}$ is weight of the core. $p_{i,j}$ is a factor for power of core $\varsigma_{i,j}$ which is a binary variable.

Three factors take part in this metric. The first factor is $r^{Per}_{i,j}$ which is a factor for core aging. If $R^{Per}_{i,j}(t)$ is greater than $R^{target}(t)$, it means the core $\varsigma_{i,j}$ is fresh enough, otherwise, the core $\varsigma_{i,j}$ is aged. Therefore, the value of $r^{Per}_{i,j}$ is equal to $R^{Per}_{i,j}(t)$ if the core $\varsigma_{i,j}$ is not aged, otherwise $r^{Per}_{i,j}$ is set to zero. Hence, a region with FN $\varsigma_{w,h}$ and higher $RPM^{r}_{w,h}$ contains more fresh cores. The second factor is $p_{i,j}$ that distributes power consumption among cores. To initiate $p_{i,j}$, we compare the power of the core $\varsigma_{i,j}$ with $TDP/(m \times m)$. Since we must guarantee to meet the TDP constraint, we consider the worst-case scenario. It means among the power of tasks, mapped on the core $\varsigma_{i,j}$, the one with the highest power is assumed as the power of

core $\varsigma_{i,j}$ (further explanation in Section VI). If the power of core $\varsigma_{i,j}$ exceeds $TDP/(m \times m)$, $p_{i,j}$ is set, otherwise, it is reset. The power factor discourages the selection of regions that have a high power density. The last effective factor to select an appropriate region is the utilization factor. We map tasks on a region that has the lowest utilization to apply load balancing besides distributing power density. After calculating the RPM, it is transmitted to the LPP-MC mapping unit.

Life-time Peak Power management in Mixed-Criticality systems (LPP-MC) Mapping Unit: A task allocation algorithm must meet the required reliability level and peak power constraint, and maximize the processor's utilization. To achieve these objectives, this unit uses region selection. The region with the maximum metric level takes priority over the other regions. The LPP-MC mapping unit takes the task subset and the RPM from the subset unit and the reliability-power unit, respectively. Then, the LPP-MC mapping unit selects a region that has more fresh cores and lower power density using the defined metric. After region selection, the LPP-MC mapping unit sorts tasks of the subset and cores of the selected region. Then, it maps the tasks based on *Best Fit* (BF) strategy. The mapped tasks are given to the scheduling unit as an input.

Re-Execution Number Calculator Unit: LPP-MC must guarantee the mapped tasks to be schedulable under the reliability constraints. The re-execution number calculator unit calculates the required number of re-executions for each task τ_k to guarantee its timing reliability requirement based on DO-178B.

This unit considers both effect of transient and permanent faults to calculate the re-execution numbers. If only the transient faults are considered to calculate the re-execution time overhead, it may lead to fault rate underestimation in the long-term. Therefore, we present the following model to guarantee the reliability of tasks under permanent and transient faults to have a more accurate estimation of the required number of re-executions for each task in order to meet the timing reliability requirements.

The reliability $R_{i,j}^{Per}(t)$ and $R_{i,j}^{Tra}(t,\chi)$ are calculated based on the permanent and transient faults rate, respectively. These reliability factors are independent, therefore the total reliability of each core $\varsigma_{i,j}$, denoted by $R_{i,j}^{Tot}(t)$, can be modeled as follows:

$$R_{i,j}^{Tot}(t,\chi) = R_{i,j}^{Per}(t) \times R_{i,j}^{Tra}(t,\chi) \tag{13}$$

$PFH_{i,j}^{Tot}(t,\chi)$ is the PFH for the tasks on the core $\varsigma_{i,j}$:

$$PFH_{i,j}^{Tot}(t,\chi) = 1-(1-PFH_{i,j}^{Per}(t))\times(1-PFH_{i,j}^{Tra}(t,\chi)) \tag{14}$$

that it is used to calculate the re-execution number n_{χ}^{χ}.

Schedulability Test Unit: This unit uses re-execution number n_{χ}^{χ} to calculate $U_{LO}^{LO}, U_{HI}^{LO}, U_{HI}^{HI}$ and finally U_{MC}. It calculates total system utilization U_{MC} by using Eq. (15) [7].

$$U_{MC} = max\{U_{LO}^{LO} + U_{HI}^{LO}, \\ U_{HI}^{HI}/(1-(U_{HI}^{LO}/(1-U_{LO}^{LO}))) + U_{LO}^{LO}/(d_f-1)\} \tag{15}$$

If U_{MC} is larger than 1, the tasks cannot be scheduled. Hence, the schedulability test unit reset the schedulability flag and gives it back to the LPP-MC mapping unit to re-map the tasks. Otherwise, the schedulability test unit gives the mapped tasks to the reliability-power unit.

If a subset is mapped and scheduled, the next subset and the updated RPM are given to the LPP-MC mapping unit. The cycle of task mapping and scheduling is repeated until all tasks are mapped. Now, we explain the algorithm of the units.

B. Algorithm Description

Algorithm 1 is consisted of four main functions: *update_RPM()*, *Subset_Creation()*, *LPP_MC_Map()* and *Calculate_U_MC()*. Function *update_RPM()* and function *Subset_Creation()* are used in the reliability-power unit and the subset unit, respectively. We explain function *LPP_MC_Map()* in Algorithm 2 which is used in the LPP-MC mapping unit. Furthermore, function *Calculate_U_MC()* is used in the scheduling unit and is explained in Algorithm 3. The detail of each algorithm is as follows:

Algorithm 1: For each start of the system operation, Algorithm 1 updates the reliability $R_{i,j}^{Per}(t)$, the utilization factor $u_{i,j}$ and the power factor $p_{i,j}$ for each core $\varsigma_{i,j}$ in lines 1-4. At the first time, since all processing cores are free, $u_{i,j}$ and $p_{i,j}$ are initiated to 0. The reliability $R_{i,j}^{Per}(t)$ is set to the last reliability of core $\varsigma_{i,j}$. Then, line 5 calculates the maximum radius of the chip. In lines 6-13 the algorithm calculates the $RPM_{w,h}^{r}$ for each FN $\varsigma_{w,h}$ and radius r based on the Eq. (12). Furthermore, it sorts RPM_array in decreasing order of the RMP in line 14. In line 15 function *Subset_Creation()* divides the task set τ in l subsets. In line 16, some variables is initiated: l is the number of subsets, sc and cc count the number of subsets and cores, respectively. r is the current radius which is used in mapping the tasks.

Lines 17-49 map the subsets on the cores. These lines are repeated until all subsets are mapped successfully. In line 18, flag $sche\_suc$ is initiated to 0. This flag is set by function *Calculate_U_MC()*, when the subset sc is schedulable. Core $\varsigma_{w,h}$ with the maximum $RPM_{w,h}^{0}$ is chosen as FN. As explained in Section A, a region with the highest RPM is the most appropriate region for mapping a subset.

Function *LPP_MC_Map()* maps the tasks of the current subset (subset sc) on the selected region with center FN and radius r (line 19). If function *LPP_MC_Map()* maps the subset sc, Algorithm 1 tests the schedulability and updates the RPM (lines 22-35). If map_suc is set (line 22), function *Calculate_U_MC()* tests the schedulability for provided mapping (line 23). In this step, if the flag $sche\_suc$ is

Algorithm 1 Proposed approach (LPP-MC) description

```
1: for each core ς_{i,j} do
2:     ς_{i,j}.R^{Per} ← Calculate_Reli(); %Eq. (8)
3:     ς_{i,j}.p ← 0, ς_{i,j}.u ← 0;
4: end for
5: max_r ← (√cores_number − 1)/2;
6: for each r = 0 : max_r do
7:     cc ← 0;
8:     for each FN ς_{w,h} do
9:         RPM^r_{w,h} ← update_RPM(ς.R^{Per}, ς.p, ς.b); %Eq. (12)
10:        rpm_array.value(r, cc) ← RPM^r_{w,h};
11:        rpm_array.FN(r, cc) ← w, h; cc ← cc + 1;
12:    end for
13: end for
14: rpm_array ← Sort(rpm_array);
15: sub_set ← Subset_Creation(Tasks);
16: l ← length(sub_set); sc ← 1; cc ← 0; r ← 0;
17: while sc ∼= l do
18:    sche_suc ← 0; FN ← rpm_array.FN(r, cc);
19:    (map_suc, Q) ← LPP_MC_Map(sub_set(sc), r, FN, ς);
20:    if map_suc then
21:        sche_suc = Calculate_U_MC(Q);
22:        if sche_suc then
23:            for each r = 0 : max_r do
24:                cc ← 0
25:                for each FN ς_{w,h} do
26:                    RPM^r_{w,h} ← update_RPM(ς.R^{Per}, ς.p, ς.b); %Eq. (12)
27:                    rpm_array.value(r, cc) ← RPM^r_{w,h};
28:                    rpm_array.FN(r, cc) ← w, h; cc ← cc + 1;
29:                end for
30:            end for
31:            rpm_array ← Sort(rpm_array);
32:            sc ← sc + 1; r ← 0; cc ← 0;
33:            mapped_tasks(sc) = Q;
34:        end if
35:    end if
36:    if !map_suc || !sche_suc then
37:        if cc ! = length(RPM_array(r)) then
38:            cc ← cc + 1;
39:        else
40:            if r < r_max then
41:                r ← r + 1; cc ← 0;
42:            else
43:                return 0
44:                break; %cannot find the appropriate FN
45:            end if
46:        end if
47:    end if
48: end while
49: return mapped_tasks;
```

Algorithm 2 LPP-MC-Map function

```
1: Tasks ← Sort_Tasks(Tasks);
2: for k = 1 : length(Tasks) do
3:     Cores ← Sort_Cores(FN, r, ς);
4:     τ_k ← Tasks(k); ij ← 0; deltaR ← 0; map_suc ← 0;
       map_suc ← 0;
5:     while !map || ij ∼= length(Cores) do
6:         c = Cores(ij); c.W = 1 − c.U;
7:         if c.deltaR ⩽ 0 && τ_k.U ⩽ c.W && WCP ⩽ TDP then
8:             Q(k) ← assign(c, τ_k); map_suc ← 1; break;
9:         else if !deltaR && τ_k.U ⩽ c.W && WCP ⩽ TDP then
10:            BU_c ← c; deltaR ← 1;
11:        end if
12:        ij ← ij + 1;
13:    end while
14:    if !map_suc then
15:        if deltaR then
16:            Q(k) ← assign(BU_c, τ_k);
17:        else
18:            Q(k) ← 0; map_suc ← 0; break;
19:        end if
20:    end if
21: end for
22: return map_suc, Q;
```

set (line 22), $RPM^r_{w,h}$ is updated by considering the changes of $u_{i,j}$ and $p_{i,j}$ values for each core $\varsigma_{i,j}$ in line 23-30. Then, the counter sc is increased and the variables cc and r are reset (line 32).

If in functions *LPP_MC_Map()* or *Calculate_U_MC()*, the subset sc cannot be mapped or schedulable due to the constraints (line 36), the algorithm looks for the next appropriate region (lines 37-46). In each time, if the algorithm returns 0, there is no possible region for mapping and scheduling tasks.

Algorithm 2: This algorithm gets the subset sc, the cores ς, FN $\varsigma_{w,h}$ and radius r. FN $\varsigma_{w,h}$ and radius r are the properties of the selected region which determines its cores. Function *LPP_MC_Map()* applies a *Worst Fit* (WF) strategy based on the utilization of cores. Before mapping the tasks of the subset sc on the selected region, function *Sort_Tasks()* sorts the tasks in decreasing order of power consumption. Then, the tasks with the same power are sorted in decreasing order of utilization.

In line 3, function *Sort_Cores()* sorts all the cores in the selected region as follows; it sorts first non-busy cores in decreasing order of reliability and then, cores with equal reliability, in increasing order of utilization. Then Lines 2-21 are repeated until all tasks are mapped on the cores. Task τ_k is mapped on core c and the information of mapping is saved on a data structure Q (line 8). If the following three conditions are satisfied: **1)** Processor core c is fresh enough (i.e., $deltaR = R^{target}_{i,j}(t) - R^{Per}_{i,j}(t) < 0$), **2)** The utilization of the task τ_k is smaller than the weight of core c, and **3)** Adding the task τ_k does not violate the TDP constraint. Otherwise, the first core for which, condition 1 and 2 are satisfied, is selected for mapping the τ_k on

Algorithm 3 Schedulability test in LPP-MC

1: $PFH\_target(HI) \leftarrow initial\_PFH\_target(); \% \; TABLE \; I$
2: $PFH\_target(LO) \leftarrow initial\_PFH\_target(); \% \; TABLE \; I$
3: **for** *each core* $\in Q$ **do**
4: $ij \leftarrow Q.CoreNymber$;
5: **for** *each criticality level* χ **do**
6: $PFH\_Tot(ij, \chi) \leftarrow 1$; $n(ij, \chi) \leftarrow 0$;
7: **while** $PFH\_Tot(ij, \chi) >= PFH\_target(\chi)$ **do**
8: $n(ij, \chi) \leftarrow n(ij, \chi) + 1$;
9: $PFH\_Tot(ij, \chi) \leftarrow calculate\_PFH\_Tot(ij, \chi, Q)$;
10: **end while**
11: **end for**
12: **for** *LO_Crit task* **do**
13: **while** $PFH\_Tot(ij, SD) >= PFH\_target(LO)$ **do**
14: $n\_Prim(ij) \leftarrow n\_Prim(ij) + 1$;
15: $PFH\_Tot(ij, SD) \leftarrow calculate\_PFH\_Tot(ij, SD, Q)$;
16: **end while**
17: **end for**
18: $n_{LO}^{LO} = n_{LO}^{HI}$;
19: $U^{LO} \leftarrow calculate\_Util(n_{LO}^{LO}, n_{HI}^{LO}, ij)$;
20: $U^{HI} \leftarrow calculate\_Util(n_{LO}^{LO}, n_{LO}^{HI}, n_{HI}^{HI}, ij)$;
21: $U_{MC} \leftarrow MAX(U^{LO}, U^{HI})$;
22: **if** U_{MC} <= 1 **then**
23: $sche\_suc \leftarrow 1$;
24: **else**
25: $sche\_suc \leftarrow 0$; *break*;
26: **end if**
27: **end for**
28: **return** *sche_suc*;

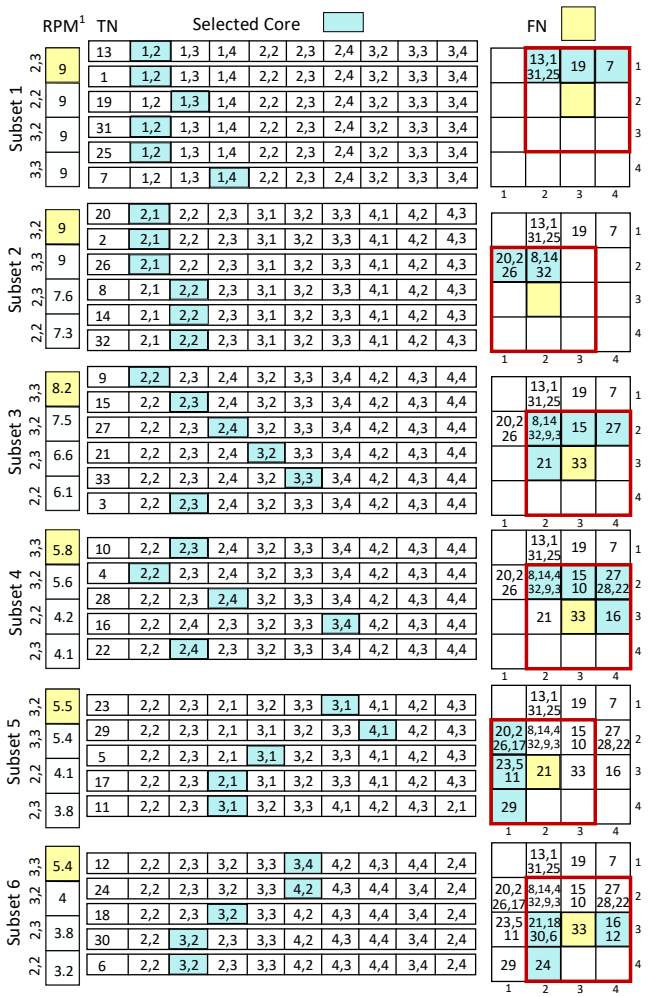


FIGURE 5: An example to illustrate the proposed mapping

it (line 10). If the variables $map = 0$ and $deltaR = 1$, Algorithm 2 maps the task τ_k on a core that satisfied the conditions 1 and 2 (lines 14-16). Function *LPP_MC_Map()* returns fail and stops task mapping whenever it cannot find an appropriate core for a task(lines 18).

Algorithm 3: It tests schedulability of tasks for the subset sc which are mapped on the cores in the mapping Q. In the first step, the algorithm initiates the target PFH (PFH^{target}) for the tasks due to the their criticality levels. Then, lines 3-27 are repeated for each core c in the current mapping. In lines 5-11, Algorithm 3 calculates the re-execution numbers of HI-Crit and LO-Crit tasks (n_{HI}^{HI} and n_{LO}^{HI}) in the HI mode by using Eq. (14). Then, it calculates the re-execution number of HI-Crit tasks in the LO mode (n_{HI}^{LO}) in lines 12-17. In lines 18-21, it uses n_{HI}^{HI}, n_{LO}^{HI} and n_{HI}^{LO} to calculate the HI and LO mode utilization U^{HI} and U^{LO}, and then the system utilization U_{MC} by using the Eq. (15). If Algorithm 3 cannot schedule at least one task on a core, it fails and returns 0.

Example: We present an example here to explain LPP-MC. A task set is considered in TABLE 3 to be mapped on a 4×4 multi-core platform. According to FIGURE 4, there are four major steps to apply LPP-MC: subsets creation, region selection, mapping the tasks, and testing the schedulability. Before these steps, the RPM is initiated to 1 and 9 for $r = 0$ and $r = 1$, respectively (lines 1-13, Algorithm 1).

In the first step, LPP-MC sorts the tasks from the HI-Crit to the LO-Crit level to divide task set in subsets (TABLE 3). Then, it assigns one task of the task set to each subset until all subsets include one task. It repeats this algorithm for all tasks (line 15, Algorithm 1). The result of this step is shown in TABLE 4. The remaining three steps are repeated for the mapping of each subset (lines 17-48, Algorithm 1). We explain these three steps for subset 1 that is similar for the rest of subsets.

In the second step, Algorithm 1 chooses a region with FN ς_{ij} and radius $r = 0$ (line 18) and goes to the third step. Function *LPP_MC_Map()* maps subset sc on the selected region. Before mapping the tasks, Algorithm 2 sorts the tasks based on the power and utilization (line 1). The result of the task sorting is shown in FIGURE 5. Then, Algorithm 2 maps the tasks τ_{13} and τ_1 on core ς_{ij} ($c.U = \tau_{13}.U + \tau_1.U = 0.55$) in lines 2-21. When Algorithm 2 tries to map the task τ_{19}, it fails due to the second condition in line 7 or 9 (($\tau_{19}.U =$

TABLE 3: Task set example

TN	τ_1	τ_2	τ_3	τ_4	τ_5	τ_6	τ_7	τ_8	τ_9	τ_{10}	τ_{11}	τ_{12}	τ_{13}	τ_{14}	τ_{15}	τ_{16}	τ_{17}	τ_{18}	τ_{19}	τ_{20}	τ_{21}	τ_{22}	τ_{23}	τ_{24}	τ_{25}	τ_{26}	τ_{27}	τ_{28}	τ_{29}	τ_{30}	τ_{31}	τ_{32}	τ_{33}
Crit	H	H	H	H	H	H	H	H	H	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L
Util	.05	.12	.12	.02	.07	.15	.2	.02	.1	.1	.2	.5	.5	.55	.55	.5	.25	.25	.55	.4	.55	.2	.55	.55	.1	.2	.55	.25	.55	.05	.2	.2	.55
Pow	.58	.61	.38	.61	.53	.37	.43	.5	.62	.63	.39	.63	.62	.49	.58	.39	.47	.61	.58	.62	.54	.36	.59	.62	.54	.57	.56	.46	.54	.4	.55	.36	.43

TABLE 4: The task set is divided in the subsets

Subset 1

TN	τ_1	τ_7	τ_{13}	τ_{19}	τ_{31}	τ_{25}
Crit	H	H	L	L	L	L
Util	.05	.2	.5	.55	.2	.1
Pow	.58	.43	.62	.58	.55	.54

Subset 2

τ_2	τ_8	τ_{14}	τ_{20}	τ_{26}	τ_{32}
H	H	L	L	L	L
.12	.02	.55	.4	.2	.2
.61	.5	.58	.62	.57	.36

Subset 3

τ_3	τ_9	τ_{15}	τ_{21}	τ_{27}	τ_{33}
H	H	L	L	L	L
.12	.1	.55	.55	.55	.55
.38	.62	.58	.54	.56	.43

Subset 4

τ_4	τ_{10}	τ_{16}	τ_{22}	τ_{28}
H	L	L	L	L
.02	.1	.5	.2	.25
.61	.63	.39	.36	.46

Subset 5

τ_5	τ_{11}	τ_{17}	τ_{23}	τ_{29}
H	L	L	L	L
.07	.2	.25	.55	.55
.53	.39	.47	.59	.54

Subset 6

τ_6	τ_{12}	τ_{18}	τ_{24}	τ_{30}
H	L	L	L	L
.15	.5	.25	.55	.05
.37	.63	.61	.62	.4

$0.55) \nleqslant (c.W = 1-0.55)$). LPP-MC cannot map the subset sc on a region with radius $r = 0$. Therefore, Algorithm 1 returns to the second step. It increases the radius r (line 41) and chooses the FN $\varsigma_{2,3}$ (line 18), then it goes to the third step again.

In the third step, function *LPP_MC_Map()* sorts tasks based on function *Sort_Tasks()*. Then, it sorts cores based on the reliability, utilization and power. It maps task τ_{13} on the core $\varsigma_{1,2}$ that is the first core in the queue in FIGURE 5. The value of the power and utilization are updated for this core and the cores are sorted again in order to map next task. Task τ_1 is mapped on the first core in the queue ($\varsigma_{1,2}$). The utilization of the core $\varsigma_{1,2}$ is equal to $0.5 + 0.05 \times 3 = 0.65$. Therefore, the weight of core $\varsigma_{1,2}$ is equal to $1-0.65 = 0.35$. The utilization of τ_{19} is 0.55 and is greater than the weight of $\varsigma_{1,2}$, hence it cannot be mapped on this core. Algorithm 2 maps task τ_{19} on the next core in the queue (core $\varsigma_{1,3}$). To map task τ_{31} and τ_{25}, it selects the first core in the queue (core $\varsigma_{1,2}$). Then, it updates the core utilization: $0.65 + 0.2 + 0.1 = 0.95 < 1$. The task τ_7 is mapped on the core $\varsigma_{1,4}$ that is the first appropriate core on the queue with utilization is $3 \times 0.2 = 0.6 \leqslant 1$.

In the last step, LPP-MC tests the schedulability of mapped tasks on the cores in Algorithm 3. In lines 1-20 of Algorithm 3, the number of re-executions is calculated for the HI-Crit and the LO-Crit tasks on each core: $n_{HI}^{HI} = 3$, $n_{HI}^{LO} = 2$ and $n_{LO}^{LO} = 1$. Then, it calculates U_{MC} by helping Eq. (15) for cores $\varsigma_{1,2}$, $\varsigma_{1,3}$ and $\varsigma_{1,4}$ that is equal to $0.85, 0.55, 1$, respectively (lines 22-24). These three steps are executed for mapping the other subsets. The result of mapping all subsets are shown in FIGURE 5.

VI. Experimental Result

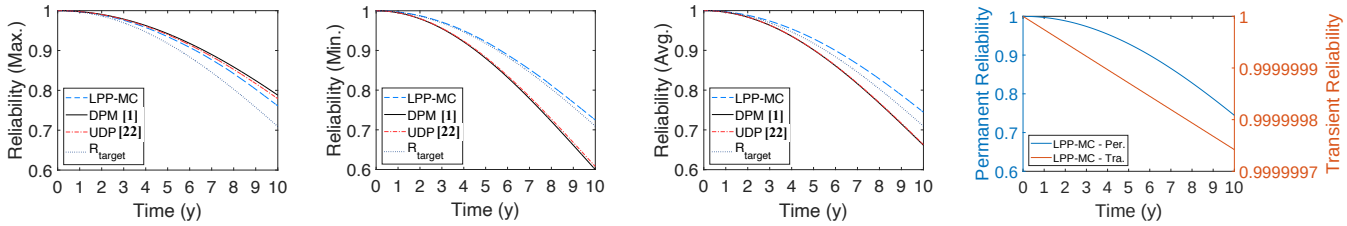
In this section, we first present the experimental setup. Then, we evaluate the LPP-MC in terms of reliability and life-time, power and temperature distribution, and utilization and compare to the state-of-the-arts.

A. Experimental Setup

To evaluate our system, we model the embedded multi-core platforms with ARM processors in our experiments. Besides, we use two types of task sets for evaluation: real task set for flight management system [2], [7] and synthetic task sets, similar to the state of the art studies such as [2], [4], [22]. The procedure described in [7], [9] has been used to generate synthetic task sets, where the following data are employed.

- The number of cores is equal to $4 \times 4 = 16$.
- System utilization is normalized by $u_{sys} = u_{tot}/16$ and the normalized utilization is bounded by $u_{sys} \in [0.01, 1]$.
- The utilization of each task is bounded by $u \in [0.01, 0.2]$.
- The period of each task is bounded by $T \in [200, 2000]$.
- HI-Crit Task Probability: a task is a HI-Crit task with the probability of $p_{ht} \in [0.1, 0.3]$.
- HI Mode Probability: the system switches to the HI mode with the probability of $p_{hm} \in [0.01, 0.2]$.

In order to have a realistic power consumption data for tasks, we use the MEET tool [51] (which is configured with ARM cores), and running several embedded benchmarks from MiBench suite [52] such as Automotive, Network and Telecommunication benchmarks on it. The benchmarks are run 1000 times to report the maximum values of power consumption. According to the measurements, the maximum power of tasks is generated randomly following the normal distribution, in the range of the minimum and maximum values, reported in experiments, which is $pow \in [456, 833]$mW. In this work, we consider the TDP value as 80% of the maximum power that a chip consumes, which is conventionally considered in embedded processors [53]. In addition, in order to obtain the temperature of cores, HOTSPOT simulator [54] is used throughout the execution for a floorplan and configuration platform which has ARM cores [16]. For the configuration file, we use the parameters reported in [55] which is for ARM processors. The ARM core (A7) has an area of $0.45mm^2$ in our experiments reported by the ARM company.



(a) Maximum (b) Minimum (c) Average (d) Transient and Permanent

FIGURE 6: System reliability over time, by utilizing LPP-MC, DPM [1] and UDP [22] approaches.

B. Experimental Results

As mentioned in previous sections, there is no approach that considers life-time and peak power in multi-core MCSs. Therefore, to evaluate LPP-MC, we compare it with two state-of-the-art mapping algorithms in multi-core MCSs which are more close to the proposed approach; DPM [1], and UDP [22]. In the following section, we first evaluate the effect of different mapping on the reliability and life-time of the cores. Furthermore, we measure the total power of cores in time slots to show the peak power. Then, we demonstrate temperature distribution among the cores. Eventually, we show the schedulability using extensive experiments by generating synthetic task sets for the proposed approach.

1) Life-time Reliability

FIGURE 6 shows system reliability in long-term (10 years) when UDP [22], DPM [1] and LPP-MC are applied on an MCS. To accelerate the life-time evaluation, we consider the worst-case scenario in power and core utilization. We extract the maximum, minimum and average reliability of 16 cores at each time (based on the Eq. (8)) to show the effect of different mapping on reliability. In the first three figures of FIGURE 6, although the reliability of cores is approximately equal to 1 in the first year when applying three approaches the distance between the reliability of cores is increasing over time (up to 0.12). The reason is that, the effect of various mapping on the reliability against permanent faults determines after spending a long time.

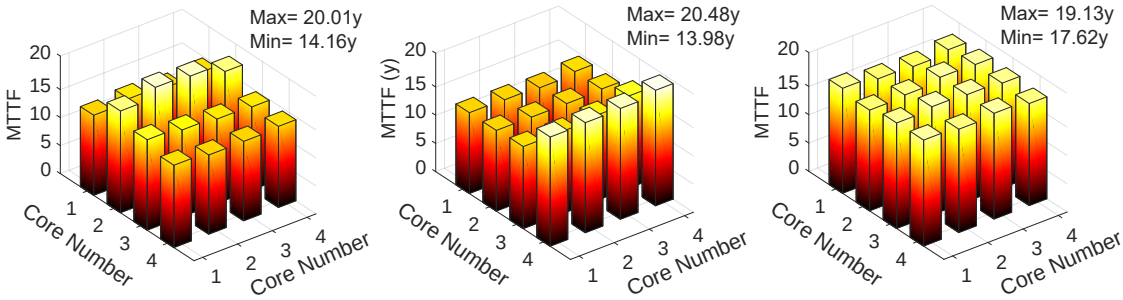
FIGURE 6a shows that the maximum reliability of cores in UDP and DPM is higher than LPP-MC, while FIGURE 6b shows that applying the two former approaches results in lower reliability of cores than LPP-MC. Therefore, using the LPP-MC approach reduces the variance between the maximum and minimum reliability of cores, while some of the cores are more reliable than other cores in UDP and DPM. Moreover, it is reasonable that the maximum reliability of cores in LPP-MC is lower than other approaches. The reason is that it increases the minimum reliability of cores compared to two other approaches to balance the reliability of cores. Additionally, it can be extracted from FIGURE 6 that since the maximum, average, and minimum reliability of cores are almost equal, the reliability is balanced among cores when applying LPP-MC. Balancing the reliability of cores not only reduces the probability of failure of MCSs, but also increases the average reliability of cores due to the results presented in FIGURE 6c.

FIGURE 6d shows system reliability considering the transient and permanent faults which are calculated by Eq. (7) and Eq. (8), respectively. System reliability by applying LPP-MC is 0.9999997 and 0.7444 in 10 years by considering the transient fault and permanent fault, individually. It is worth mentioning that UDP [22] and DPM [1] do not consider any fault tolerant techniques and since failure in the execution of HI-Crit tasks may lead to system failure [16], the system may fail with the first transient fault on those tasks, which causes irreparable damage to the system. This is while, LPP-MC schedules tasks by applying re-execution at the presence of transient faults. LPP-MC schedules tasks by EDF-VD after applying re-execution fault tolerance technique in order to meet the deadlines of tasks. On the other hand, LPP-MC postpone permanent faults by re-mapping tasks in each operational phase which leads to better load balancing over the cores.

In this work, MTTF is reported as a metric to evaluate the life-time of the system. FIGURE 7 compares MTTF of the LPP-MC mapping approach to UDP and DPM. FIGURE 7c shows that LPP-MC has a more balanced life-time of the cores compared to UDP and DPMs. This is because the former approaches use static mapping, while LPP-MC changes the mapping of tasks due to the aging parameter in each start of operational phase. Besides, the MTTF of 12 cores is equal to about 14 years in the UDP (FIGURE 7a) and DPM (FIGURE 7b) approaches, while it is approximately equal to 19 years for LPP-MC (FIGURE 7c) approach. The system MTTF is 13.98, 14.16 and 17.62 when applying DPM, UDP and LPP-MC approach, respectively. LPP-MC increases the life-time of the system by up to 26% based on the reported MCS MTTF.

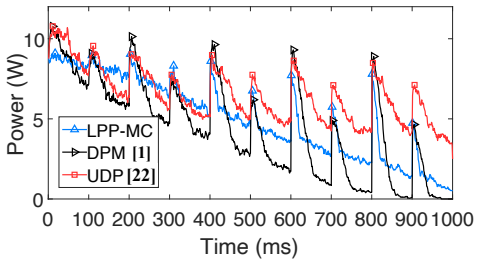
2) Peak Power Management

FIGURE 8a shows the worst-case power consumption trace of 1000 task sets during a hyper-period (1000ms). As shown, the total power consumption has the peak value when time instants located at integer multiples of 100ms. The reason

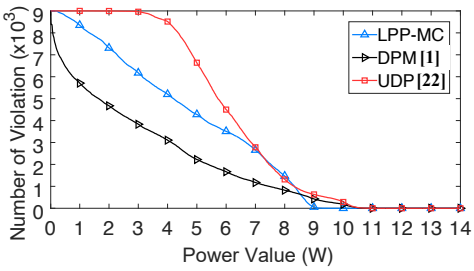


(a) UDP [22] (b) DPM [1] (c) LPP-MC

FIGURE 7: The effect of mapping on MTTF of cores



(a) Power trace during 1000 ms



(b) Number of violation of a specific power

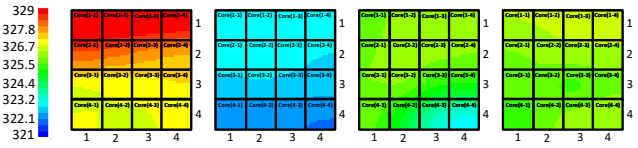
FIGURE 8: Power consumption during a hyper-period

is that, there are many tasks that have period equal to time instants, which are integer multiples of 100ms (100, 200, 300, etc.). Therefore, the system consumes more power to execute many tasks that execute in this time instants. The curve of total power consumption has a decreasing trend in three approaches, without considering the peaks (FIGURE 8a). This means the period of some tasks is bigger than 100ms. For instance, there are some tasks with the period of 1000ms that are executed once in this hyper-period. As the value of TDP is one of LPP-MC inputs, TDP=10W (which is acceptable for embedded processors) is given to the algorithm as the input of the experiment. As FIGURE 8a shows, LPP-MC succeeds to meet given TDP, while UDP and DPM violate TDP. The maximum power consumption for the UDP, DPM, and LPP-MC approaches is 11.02W, 10.78W, and 9.08W, respectively. Hence, LPP-MC reduces peak power consumption up to 17.6% compared to others.

In addition to considering a constant value for TDP, FIGURE 8b counts how many times systems have a power consumption higher than a specific power. It is worth mentioning that the better mapping is the one that has no violation in lower power consumption because it can be applied to map tasks on a chip with lower TDP. The number of violation is non-zero for power consumption lower than 11.02W in curve related to UDP, lower than 10.78W in curve related to DPM and lower than 9.08W for LPP-MC. Therefore, LPP-MC can be used for systems with lower TDP.

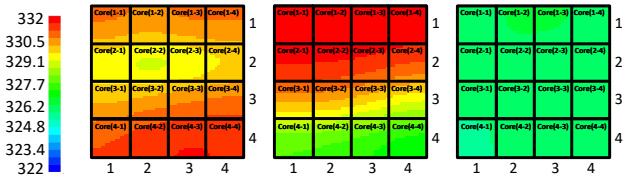
3) Temperature Distribution

We first measure temperature in an short term operational phase, equal to 10 hours. Note that since the UDP and DPM approaches have static mapping during run-time, the result for any short term operational time has the same trend and there is no matter we take which operational time result. The steady-state temperature output of HOTSPOT [54] is shown in FIGURE 9. FIGURE 9b, 9c and 9d show that the temperature is well distributed. However, four processing cores $\varsigma_{1,1}$, $\varsigma_{1,2}$, $\varsigma_{1,3}$ and $\varsigma_{1,4}$ of UDP [22], have reasonably higher temperature, which are equal to 327.74 K, 327.67 K, 327.71 K and 327.76 K, respectively (FIGURE 9a). Although DPM has lower maximum temperature that other approaches and also distributes temperature, it activates most of processor cores, which may causes to not meet TDP constraint. On the other hand, as shown in FIGURE 9b, the temperature of upper half of the cores are higher than the temperature of cores $\varsigma_{4,1}$, $\varsigma_{4,2}$, $\varsigma_{4,3}$ and $\varsigma_{4,4}$, hence they would age faster than other cores in long term. The reason is that keeping running these cores in long-term and being active causes higher temperature. We show and explain the temperature distribution in long term in the next paragraph. FIGURE 9c and FIGURE 9d shows system temperature of two different task mappings in two operational phase by applying LPP-MC. In FIGURE 9c, LPP-MC activates 14 cores in addition to distributing temperature among them. Therefore, the proposed approach by applying region selection not only meets the TDP constraint due to activating



(a) UDP [22] (b) DPM [1] (c) LPP-MC (Re-map 1) (d) LPP-MC (Re-map 2)

FIGURE 9: Temperature profile for an operational time



(a) UDP [22] (b) DPM [1] (c) LPP-MC

FIGURE 10: Temperature profile in long-term

fewer cores but also, it avoids hotspots in the chip due to the thermal distribution. On the other hand, the difference between the temperatures of free cores like $\varsigma_{3,4}$ or $\varsigma_{4,4}$ with other cores such as $\varsigma_{3,2}$, does not cause unbalanced aging. The reason is that the thermal distribution changes in long-term in LPP-MC, due to the task re-mapping in each operational phase. For instance, FIGURE 9d shows that in another operational phase, tasks are mapped on the younger processors ($\varsigma_{4,4}$).

In order to show the effect of different mapping on the distribution of temperature in the long-term, we evaluate temperature with the assumption that the system has continuously run for different operational times (FIGURE 10). As shown in FIGURE 10a and 10b, some cores have a higher temperature than other cores in UDP and DPM. This result is also matched with FIGURE 7b in which the first 12 cores also have lower MTTF due to experiencing higher temperature. However, FIGURE 10c shows that LPP-MC distributes temperature among cores since the task mapping is changed during time by considering mapping tasks on younger cores.

4) Scheduling

LPP-MC considers transient and permanent faults to calculate required re-execution number to guarantee predefined reliability level; This can affect schedulability and utilization, Eq. 15. FIGURE 11 shows the effect of LPP-MC on MCS utilization. We generate 1000 task sets for each total utilization $u_{tot} \in [0.01, 16]$. Then, we report normalized utilization $u_{tot}/16 = u_{sys} \in [0.01, 1]$. The work in [7], named *Safety Quantification with Service Degradation* (SQSD), uses re-execution technique to tolerate the faults and SD in MCSs.

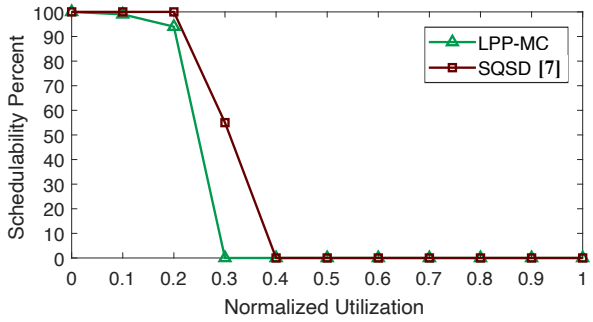


FIGURE 11: Task sets schedulability

We compare the schedulability of our approach with the SQSD. Schedulability is the percent of the schedulable task sets in comparison with all examined task sets on the MCSs. FIGURE 11 shows that SQSD guarantees 100% schedulability until the system utilization is smaller than 0.2 while LPP-MC imposes 0.1 utilization penalty. The reason of penalty is that SQSD does not consider peak power and life-time while applying peak power management approaches reduces the number of active cores at a time. This limits LPP-MC to achieve higher utilization.

VII. Conclusion and Future Work

In this work, we propose an approach called LPP-MC to prolong the life-time and reduce the peak power of mixed-criticality systems (MCSs). Besides, we present a reliability model for MCSs that considers both transient and permanent faults. For this aim, a new metric called RPM is proposed and it includes three factors; power, reliability, and utilization to select an appropriate cores' region for task mapping. Besides, the schedulability of the tasks is evaluated using EDF-VD. The results show peak power reduction up to 17.6% and life-time improvement up to 20.6%, compared to state-of-the-art works, along with reliability guarantee against permanent and transient faults.

To guarantee the worst-case scenario of system operation, we design the system for objectives' improvement by considering the worst-case scenario of task execution and fault occurrence at design-time. However, since it does not always exhibit the worst-case behavior at run-time, the core's capacity may be wasted, which can be a limitation/drawback of the proposed scheme. Therefore, as prospective future work, we would first employ the more efficient mixed-criticality task scheduling algorithm than EDF-VD at design-time. Then, we would consider the run-time behavior and employ the accumulated dynamic slack of cores in order to better manage the objectives, like improving the life-time or reducing the peak power consumption. Besides, in multi-core systems, to avoid manufacturing yield losses and increase reliability, designers embed redundant cores in the system. As another future work, those redundancies can be modeled and considered to improve the MCS's life-time.

REFERENCES

[1] Z. Al-bayati, Q. Zhao, A. Youssef, H. Zeng, and Z. Gu, "Enhanced partitioned scheduling of mixed-criticality systems on multicore platforms," in *Proc. of Asia and South Pacific, Design Automation Conference (ASP-DAC)*, 2015, pp. 630–635.

[2] L. Zeng, P. Huang, and L. Thiele, "Towards the design of fault-tolerant mixed-criticality systems on multicores," in *Proc. of International Conference on Compilers, Architectures and Synthesis for Embedded Systems*, 2016, pp. 6:1–6:10.

[3] Z. Al-bayati, J. Caplan, B. H. Meyer, and H. Zeng, "A four-mode model for efficient fault-tolerant mixed-criticality systems," in *Proc. of Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2016, pp. 97–102.

[4] C. Gu, N. Guan, Q. Deng, and W. Yi, "Partitioned mixed-criticality scheduling on multiprocessor platforms," in *Proc. of Design, Automation & Test in Europe (DATE)*, 2014, pp. 292:1–292:6.

[5] M.-H. Haghbayan, A. Miele, A. M. Rahmani, P. Liljeberg, and H. Tenhunen, "Performance/reliability-aware resource management for many-cores in dark silicon era," *IEEE Transaction on Computers (TC)*, vol. 66, no. 9, pp. 1599–1612, 2017.

[6] S. S. Sahoo, T. D. Nguyen, B. Veeravalli, and A. Kumar, "Lifetime-aware design methodology for dynamic partially reconfigurable systems," in *Proc. of Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2018, pp. 393–398.

[7] P. Huang, H. Yang, and L. Thiele, "On the scheduling of fault-tolerant mixed-criticality systems," in *Proc. of Design Automation Conference (DAC)*, 2014, pp. 1–6.

[8] Z. Al-bayati, B. H. Meyer, and H. Zeng, "Fault-tolerant scheduling of multicore mixed-criticality systems under permanent failures," in *Proc. of International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*, 2016, pp. 57–62.

[9] B. Ranjbar, B. Safaei, A. Ejlali, and A. Kumar, "Fantom: Fault tolerant task-drop aware scheduling for mixed-criticality systems," *IEEE Access*, vol. 8, pp. 187 232–187 248, 2020.

[10] L. A. Johnson, "Do-178b, software considerations in airborne systems and equipment certification," *Crosstalk, October*, vol. 199, pp. 11–20, 1998.

[11] J. Caplan, Z. Al-Bayati, H. Zeng, and B. H. Meyer, "Mapping and scheduling mixed-criticality systems with on-demand redundancy," *IEEE Transaction on Computers (TC)*, vol. 67, no. 4, pp. 582–588, 2018.

[12] P. K. Saraswat, P. Pop, and J. Madsen, "Task migration for fault-tolerance in mixed-criticality embedded systems," *ACM SIGBED Review*, vol. 6, no. 3, pp. 6:1–6:5, 2009.

[13] P. K. Saraswat, P. Pop, and J. Madsen, "Task mapping and bandwidth reservation for mixed hard/soft fault-tolerant embedded systems," in *Proc. of Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2010, pp. 89–98.

[14] G. Liu, Y. Lu, and S. Wang, "An efficient fault recovery algorithm in multiprocessor mixed-criticality systems," in *Proc. of High Performance Computing and Communications & Embedded and Ubiquitous Computing (HPCC_EUC)*, 2013, pp. 2006–2013.

[15] B. Ranjbar, A. Hosseinghorban, M. Salehi, A. Ejlali, and A. Kumar, "Toward the design of fault-tolerance-and peak-power-aware multi-core mixed-criticality systems," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 41, no. 5, pp. 1509–1522, 2022.

[16] B. Ranjbar, T. D. A. Nguyen, A. Ejlali, and A. Kumar, "Power-aware run-time scheduler for mixed-criticality systems on multi-core platform," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 40, no. 10, pp. 2009–2023, 2021.

[17] 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. of International Conference on Parallel and Distributed Systems (ICPADS)*, 2014, pp. 200–209.

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

[19] Intel Corporation, *Desktop 3rd generation intel core processor family - Thermal Mechanical Specifications and Design Guidelines (TMSDG)*, January 2013.

[20] (2020) International roadmap for devices and systems. [Online]. Available: https://irds.ieee.org/

[21] 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. of Euromicro Conference on Real-Time Systems (ECRTS)*, 2012, pp. 145–154.

[22] S. Ramanathan and A. Easwaran, "Utilization difference based partitioned scheduling of mixed-criticality systems," in *Proc. of Design, Automation & Test in Europe (DATE)*, 2017, pp. 238–243.

[23] I. Ali, J.-h. Seo, and K. H. Kim, "A dynamic power-aware scheduling of mixed-criticality real-time systems," in *Proc. of International Conference on Computer and Information Technology; Ubiquitous Computing and Communications; Dependable, Autonomic and Secure Computing; Pervasive Intelligence and Computing (CIT/IUCC/DASC/PICOM)*, 2015, pp. 438–445.

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

[25] Y.-W. Zhang, "Energy-aware non-preemptive scheduling of mixed-criticality real-time task systems," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, pp. 1–1, 2021.

[26] ——, "Energy-aware mixed-criticality sporadic task scheduling algorithm," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 40, no. 1, pp. 78–86, 2021.

[27] P. Haririan and A. Garcia-Ortiz, "A framework for hardware-based dvfs management in multicore mixed-criticality systems," in *Proc. of International Symposium on Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC)*, 2015, pp. 1–7.

[28] M. Digalwar, B. K. Raveendran, and S. Mohan, "Lamcs: A leakage aware dvfs based mixed task set scheduler for multi-core processors," *Sustainable Computing: Informatics and Systems*, vol. 15, pp. 63–81, 2017.

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

[30] B. Alahmad and S. Gopalakrishnan, "Work-in-progress: Isochronous execution models for mixed-criticality systems on parallel processors," in *Proc. of IEEE Real-Time Systems Symposium (RTSS)*, 2017, pp. 354–356.

[31] E. A. Rambo and R. Ernst, "Asteroid and the replica-aware co-scheduling for mixed-criticality," in *Dependable Embedded Systems*. Springer, Cham, 2021, pp. 57–84.

[32] H. Koc, V. K. Karanam, and M. Sonnier, "Latency constrained task mapping to improve reliability of high critical tasks in mixed criticality systems," in *Proc. of IEEE Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)*, 2019, pp. 0320–0324.

[33] J. Choi, H. Yang, and S. Ha, "Optimization of fault-tolerant mixed-criticality multi-core systems with enhanced wcrt analysis," *ACM Transaction Des. Autom. Electron. Syst. (TODAES)*, vol. 24, no. 1, dec 2018.

[34] L. Huang, F. Yuan, and Q. Xu, "On task allocation and scheduling for lifetime extension of platform-based mpsoc designs," *IEEE Transaction on Parallel and Distributed Systems (TPDS)*, vol. 22, no. 12, pp. 2088–2099, 2011.

[35] A. Das, A. Kumar, B. Veeravalli, C. Bolchini, and A. Miele, "Combined dvfs and mapping exploration for lifetime and soft-error susceptibility improvement in mpsocs," in *Proc. of Conference on Design, Automation & Test in Europe (DATE)*, 2014, pp. 61:1–61:6.

[36] A. Das, A. Kumar, and B. Veeravalli, "Reliability and energy-aware mapping and scheduling of multimedia applications on multiprocessor systems," *IEEE Transaction on Parallel and Distributed Systems (TPDS)*, vol. 27, no. 3, pp. 869–884, 2016.

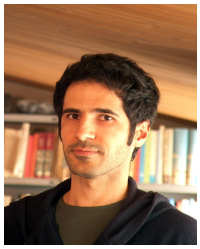
[37] A. Das, A. Kumar, and B. Veeravalli, "Aging-aware hardware-software task partitioning for reliable reconfigurable multiprocessor systems," in *Proc. of on Compilers, Architecture and Synthesis for Embedded Systems (CASES)*, 2013, pp. 1–10.

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

[39] J. Lee, B. Yun, and K. G. Shin, "Reducing peak power consumption inmulti-core systems without violatingreal-time constraints," *IEEE Transaction on Parallel and Distributed Systems (TPDS)*, vol. 25, no. 4, pp. 1024–1033, 2014.
[40] L. Behera and P. Bhaduri, "An energy-efficient time-triggered scheduling algorithm for mixed-criticality systems," *Design Automation for Embedded Systems*, vol. 24, no. 2, pp. 79–109, 2020.
[41] Y.-W. Zhang and R.-K. Chen, "A survey of energy-aware scheduling in mixed-criticality systems," *Journal of Systems Architecture*, vol. 127, p. 102524, 2022.
[42] S. S. Sahoo, B. Ranjbar, and A. Kumar, "Reliability-aware resource management in multi-/many-core systems: A perspective paper," *Journal of Low Power Electronics and Applications*, vol. 11, no. 1, p. 7, 2021.
[43] L. Behera, "A fault-tolerant time-triggered scheduling algorithm of mixed-criticality systems," *Computing*, vol. 104, no. 3, pp. 577–599, 2022.
[44] K. Cao, G. Xu, J. Zhou, M. Chen, T. Wei, and K. Li, "Lifetime-aware real-time task scheduling on fault-tolerant mixed-criticality embedded systems," *Future Generation Computer Systems*, vol. 100, pp. 165–175, 2019.
[45] B. Ranjbar, A. Hosseinghorban, S. S. Sahoo, A. Ejlali, and A. Kumar, "Bot-mics: Bounding time using analytics in mixed-criticality systems," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, pp. 1–1, 2021.
[46] Z. Guo, L. Santinelli, and K. Yang, "Edf schedulability analysis on mixed-criticality systems with permitted failure probability," in *Proc. of Embedded and Real-Time Computing Systems and Applications (RTCSA)*, 2015, pp. 187–196.
[47] B. Ranjbar, A. Hoseinghorban, S. S. Sahoo, A. Ejlali, and A. Kumar, "Improving the timing behaviour of mixed-criticality systems using chebyshev's theorem," in *Proc. of Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2021, pp. 264–269.
[48] I. Koren and C. M. Krishna, *Fault-tolerant systems*. Morgan Kaufmann, 2020.
[49] A. Meixner, M. E. Bauer, and D. Sorin, "Argus: Low-cost, comprehensive error detection in simple cores," in *Proc. of Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2007, pp. 210–222.
[50] A. K. Das, A. Kumar, B. Veeravalli, and F. Catthoor, *Reliable and Energy Efficient Streaming Multiprocessor Systems*. Springer, 2018.
[51] M. Bazzaz, M. Salehi, and A. Ejlali, "An accurate instruction-level energy estimation model and tool for embedded systems," *IEEE Transaction on instrumentation and measurement (TIM)*, vol. 62, no. 7, pp. 1927–1934, 2013.
[52] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown, "Mibench: A free, commercially representative embedded benchmark suite," in *Proc. of IEEE International Workshop on Workload Characterization. WWC-4*, 2001, pp. 3–14.
[53] Y. Liu, H. Yang, R. P. Dick, H. Wang, and L. Shang, "Thermal vs energy optimization for dvfs-enabled processors in embedded systems," in *Proc. of International Symposium on Quality Electronic Design (ISQED)*, 2007, pp. 204–209.
[54] K. Skadron, M. R. Stan, K. Sankaranarayanan, W. Huang, S. Velusamy, and D. Tarjan, "Temperature-aware microarchitecture: Modeling and implementation," *Transaction on Architecture and Code Optimization (TACO)*, vol. 1, no. 1, pp. 94–125, 2004.
[55] Y.-H. 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, 2018.

Mozhgan Navardi received the MSc degree in computer engineering from the Sharif University of Technology, Tehran, Iran, in 2019. From 2016 to 2019, he was a full-time research assistant in the Embedded Systems Research Laboratory (ESRLab.) at Sharif University of Technology under the supervision of Prof. Alireza Ejlali. Her research interests include Mixed-Criticality Systems Design, Low Power Design and Life-time Reliability Management.

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



Nezam Rohbani received the B.Sc. and M.Sc. degrees in computer engineering from University of Mazandaran, Mazandaran, Iran, and Sharif University of Technology, Tehran, Iran, in 2010 and 2012, respectively, and the Ph.D. degree in computer engineering from the Sharif University of Technology, Tehran, Iran, in 2018. His Ph.D. dissertation is focused on aging assessment and mitigation of processors in the presence of process variation in nanoscale technologies. He is currently a Post-Doctoral Researcher with the Institute for Research in Fundamental Sciences (IPM), Tehran.

Alireza Ejlali received the PhD degree in computer engineering from Sharif University of Technology, Tehran, Iran, in 2006. He is currently an associate professor of computer engineering at SUT. 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, 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.