TherMa-MiCs: Thermal-Aware Scheduling for Fault-Tolerant Mixed-Criticality Systems

Sepideh Safari, Heba Khdr, Pourya Gohari-Nazari, Mohsen Ansari,
Shaahin Hessabi, *Member, IEEE*, Jörg Henkel, *Fellow, IEEE*

Abstract— Multicore platforms are becoming the dominant trend in designing Mixed-Criticality Systems (MCSs), which integrate applications of different levels of criticality into the same platform. A well-known MCS is the dual-criticality system that is composed of low-criticality and high-criticality tasks. The availability of multiple cores on a single chip provides opportunities to employ fault-tolerant techniques, such as N-Modular Redundancy (NMR), to ensure the reliability of MCSs. However, applying fault-tolerant techniques will increase the power consumption on the chip, and thereby on-chip temperatures might increase beyond safe limits. To prevent thermal emergencies, urgent countermeasures, like Dynamic Voltage and Frequency Scaling (DVFS) or Dynamic Power Management (DPM) will be triggered to cool down the chip. Such countermeasures, however, might not only lead to suspending low-criticality tasks, but also it might lead to violating timing constraints of high-criticality tasks. In order to prevent such severe scenarios, it is indispensable to consider a temperature constraint within the scheduling process of fault-tolerant MCSs. Therefore, this paper presents, for the first time, a thermal-aware scheduling scheme for fault-tolerant MCSs, named TherMa-MiCs. In particular, TherMa-MiCs, satisfies the temperature constraint jointly with the timing constraints of the high-criticality tasks, while attempting to maximize the QoS of low-criticality tasks under the predefined constraints. At the same time, a reliability target is satisfied by employing the well-known N-Modular Redundancy (NMR) fault-tolerant technique. Experimental results show that our proposed scheme meets the temperature and timing constraints, while at the same time, improving the QoS of low-criticality tasks, with an average of 44%.

Index Terms— Multicores, N Modular Redundancy (NMR), Mixed-Criticality Systems, QoS, Temperature.

1 Introduction

In Mixed-Criticality Systems (MCSs) a large number of tasks of different criticality levels are integrated to execute on the same computing platform, to meet stringent non-functional requirements relating to the area, cost, and power [1][2]. A well-known MCS is a dual-criticality system, in which low-criticality and high-criticality tasks are considered. Low-criticality (LC) tasks have one Worst-Case Execution Time (WCET) which is specified by the system designer, while high-criticality (HC) tasks have two instances of WCETs: W^{LO} is estimated by system designers, and W^{HI} is estimated by certification authorities and is more pessimistic. Dual-criticality systems will operate in two system operational modes; *normal mode* and *overrun mode*. The system starts its execution in the normal mode, where both HC and LC tasks will be executed normally based on their W^{LO}. Whenever an HC task exceeds its W^{LO}, the system switches to the overrun mode, in which the execution priority will be given to the HC tasks to guarantee completing their execution considering their W^{HI} before their timing constraints. Thus, in the overrun mode, the state-of-the-art scheduling policies either suspend LC tasks [3][4], or guarantee a minimum service level for LC tasks [5][6], in order to guarantee the timing requirements for HC tasks. Intuitively, the various criticality levels and operational modes need to be considered by the scheduling policies of MCSs [7][8].

Mixed-criticality systems, like all other electronic systems, are susceptible to transient faults, which are considered as one of the severe reliability concerns which are increasing along with technology scaling [9][10]. Existence of a fault in an HC task might lead to catastrophic consequences [7][11]. Therefore, these systems must properly detect the faults and mitigate the effects of faults and provide recovery mechanisms when faults occur through exploiting fault-tolerant techniques. Several studies have started to employ fault-tolerant techniques within MCSs [7][12]. However, employing fault-tolerant techniques results in several challenges in MCSs.

The first challenge is that fault-tolerant techniques come with an additional timing overhead and this needs to be taken into account within the scheduling process to prevent violating timing constraints of HC tasks, especially in the overrun mode. Besides the timing challenge, fault-tolerant techniques will increase the power consumption of the cores, and thereby on-chip temperatures might increase beyond safe limits, as will be demonstrated in the following motivational example.

It is worthy to mention that the automotive indus-

- *S. Safari, P. Gohari-Nazari, M. Ansari, S. Hessabi, are with the Department of Computer Engineering, Sharif University of Technology, Tehran 14588, Iran. E-mails: {ssafari, gohary, mansari}@ce.sharif.edu, hessabi@sharif.edu.*
- *S. Safari, H. Khdr, M. Ansari, and J. Henkel are with the Karlsruhe Institute of Technology, Karlsruhe 76131, Germany. E-mails: {Sepideh.safari, Heba.khdr, Mohsen.ansari, Henkel}@kit.edu.*

Manuscript received x Aug. 2020; revised x m y; accepted x m y. Date of publication X Y Z; date of current version X Y Z.
(Corresponding author: Shaahin Hessabi)

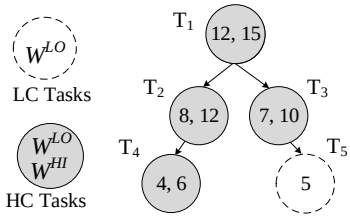


Fig. 1. An example task graph of motivational example.

try [21] is an example of integrated mixed-criticality applications running on a single chip, and therefore, applications will share resources and heat can transfer between cores. As a result, considering thermal issues within the scheduling process is relevant for the emerging real-world mixed-criticality applications.

Typically, chip manufacturers implement Dynamic Thermal Management (DTM) [13] unit on the hardware to take some urgent countermeasures like DVFS or DPM to throttle down the chip, when the temperature of any core exceeds the safe limit. However, throttling down the chip might lead to violating the timing constraints of HC tasks. *Therefore, it is indispensable to consider both temperature and timing constraints within the scheduling process of fault-tolerant multicore MCSs. It should be noted that none of the previous works have jointly considered timeliness, fault-tolerance, and thermal management in MCSs, while this paper considers, for the first time, all of these aspects jointly.* For fault-tolerance, we employ the well-known N Modular Redundancy (NMR) fault-tolerant technique, which is suitable to be applied in multicore systems, since it exploits the inherent redundancy of the multicores by executing replicas of the tasks in parallel [11]. Contrarily, other fault-tolerant techniques (e.g., re-execution) rely on time-redundancy and execute the task replica consecutively on the same core, leading to significant time overhead, and eventually, timing constraints might be violated.

1.1 Motivational Example

This example demonstrates how exploiting fault-tolerant techniques leads to violating the temperature constraint of MCSs. Let us consider a quad-core chip with a temperature constraint, T_{crit}, equal to 60 °C [14]. It should be noted that temperature constraint is the input of the system. The application task graph with five tasks {T_1, T_2, T_3, T_4, T_5}, and deadline D=75ms will be executed on the chip. Fig. 1 shows dependencies between these tasks and the two numbers at each node represent the low-level and high-level worst-case execution times of the corresponding task, i.e., W^{LO} and W^{HI}, respectively. Note that HC tasks (shown in gray color) have two instances of WCETs, while the LC task (T_5) which is shown in white color has just one WCET. Fig. 2 shows the scheduling of the given task graph (shown in Fig. 1) in four scenarios; the first one (Fig. 2a) shows the normal mode of the given MCS, where the execution time of each HC task is equal to its low-level WCET (W^{LO}). The second one (Fig. 2b) shows the overrun mode, where the execution time of all HC tasks exceed their low-level WCETs and reached W^{HI}. The third and the fourth scenarios (Fig. 2c, Fig. 2d) show also the normal and the overrun modes of the MCS, respectively, but with considering N Modular Redundancy (NMR) as the fault-tolerant technique. Here, N is considered equal to three (N=3), i.e., each task has three copies. Note that the mixed-criticality graph is scheduled based on the state-of-the-art scheduling policy for MCSs [18]. Besides the task scheduling, this figure shows the resulting peak temperature on the cores at 1ms granularity similar to [15][37]. It can be noticed in Fig. 2a, and Fig. 2b, the peak temperature does not exceed the temperature constraint. In Fig. 2b all HC tasks exceed their low-level WCETs and the system is switched to the overrun mode. However, in Fig. 2c, and Fig. 2d, the temperature constraint has been violated. The reason is that adding the task replicas to the system increases the number of simultaneous active cores, and thereby the temperature has increased beyond the safe limit. Typically, to handle such scenarios, the DTM on the chip will be triggered to throttle down the chip, and thereby cool down the chip. That, however, might lead to violating the timing constraints of the HC tasks, leading ultimately to catastrophic consequences.

In summary, *considering fault-tolerant techniques within MCSs might lead to thermal violations which have severe impact on executing the high-criticality tasks*.

1.2 Our Novel Contributions

As it can be deduced from the motivational example, it is indispensable to consider the temperature when scheduling the tasks in fault-tolerant MCSs. However, the scheduling of MCSs is known to be an NP-hard problem in the strong sense [1][6], and considering temperature constraints directly by the scheduling policy will complicate the problem further, because of the heat transfer between

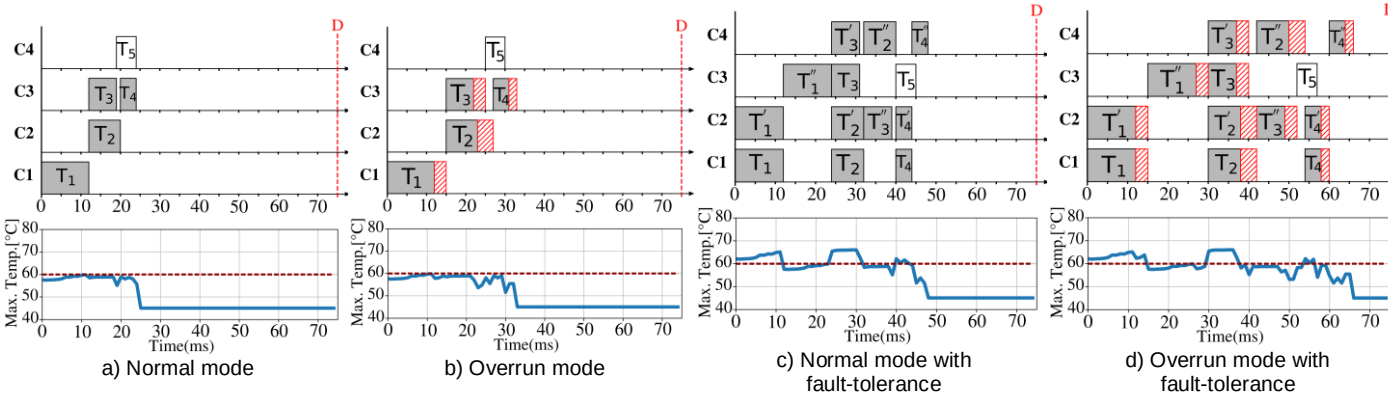


Fig. 2. A motivational example of temperature violation in fault-tolerant mixed-criticality systems.

the cores. Particularly, executing a task on a core might increase the temperatures of other cores [16][17]. That implies, for scheduling a task on one core, the temperatures of all other cores need also to be checked.

To tackle this challenge, we consider the Thermal Safe Power (TSP) which is an abstraction that provides thermally safe power constraint as a function of the number of simultaneously active cores. Executing cores at any power consumption below TSP ensures that thermal violations are avoided. That implies, for scheduling a task on one core, only the power consumption of executing the given task on that core needs to be considered, and no need to know all the power consumptions of other cores. Obviously, the complexity of considering TSP by the scheduling policy is less than considering the core temperature directly. However, since TSP is a function of the number of simultaneously active cores, we need to consider another TSP value at each time, a new task needs to be scheduled and that will affect the task scheduled previously. To solve this challenge, we propose to calculate the Maximum Safe Simultaneous Active Cores (MSSAC) factor for each task (before scheduling), based on the task's maximum power consumption and TSP. In particular, considering the power consumption of the task and the TSP values for all possible numbers of active cores, we calculate how many cores can be simultaneously active with each task. Hence, for scheduling a set of tasks at a specific time point, the MSSAC factors for all of those tasks need to be satisfied, in order to satisfy temperature constraints.

In summary, this paper presents the first thermal-aware scheduling for fault-tolerant mixed-criticality systems, named as TherMa-MiCs, which satisfies timing, temperature, and reliability constraints. The temperature constraint is satisfied by enforcing the MSSAC factors, which have been derived for each task based on TSP. The reliability target has been met by employing the *well-known N-Modular Redundancy (NMR) fault-tolerant technique*. However, in contrast to classic NMR where all N copies are executed in parallel, the MSSAC factor determines the concurrency of executing N copies to prevent thermal violations. Under the predefined constraints, our proposed TherMa-MiCs aims to maximize the QoS of the LC tasks. In particular, after scheduling all HC tasks, TherMa-MiCs, attempts to schedule as many LC tasks as possible at design time under the predefined constraints and considering the MSSAC factors. Moreover, at runtime, TherMa-MiCs, exploits the dynamically-released slack time resulting from either the cancellation of executing task replicas (when no fault occurs) or the cancellation of overrun parts (when overrun mode has not been activated), in order to execute additional LC tasks, while at the same time MSSAC factors are satisfied to keep the temperature below the predefined constraint.

In summary, the main contributions in this paper are:

- Employing the MSSAC factor derived based on TSP, to consider the temperature constraint within the offline and online scheduling.
- Proposing thermal-aware offline scheduling to schedule all HC tasks and their corresponding replicas (considering NMR fault-tolerant technique), and as many LC tasks as possible, while satisfying the predefined timing, temperature, and reliability constraints.
- Proposing thermal-aware online scheduling that cancels the execution of task replicas or overrun parts, if no fault or overrun occurs on the corresponding tasks, in order to schedule more LC tasks in the released dynamic slack times, while still considering the MSSAC factors to prevent thermal violations.

2 Related Work

Different scheduling algorithms have been proposed for MCSs to satisfy timing constraints [19]. Earliest Deadline First with Virtual Deadline (EDF-VD) [3][4][25], and Early-Release Earliest Deadline First (ER-EDF) [5][6] are the most popular ones for periodic/sporadic task model. In the EDF-VD algorithm, all LC tasks are immediately discarded after switching to the overrun mode. The ER-EDF algorithm provides a minimum acceptable service level for LC tasks in the overrun mode by increasing the period of LC tasks in the overrun mode to reduce their execution frequency and competition with HC tasks. The reference [18] presented a new scheduling method for mixed-criticality task graphs without fault-tolerant provisions. The references [26] have proposed a method to improve the QoS of LC tasks in the event of an overrun occurrence. None of the proposed methods in this category have considered reliability and temperature constraints.

Some previous work explores the scheduling problem in the context of fault-tolerant MCSs without considering power/energy or temperature constraints. The proposed methods in [7][12][27][28][29] exploited the re-execution fault-tolerant technique. The study in [30] addresses fault occurrence and overrun with separate modes in single-core and multiprocessor systems. However, it selectively chooses LC tasks to continue their execution in each mode. Exploiting fault-tolerant techniques will increase the power and temperature of the system and this has not been considered in the aforementioned works.

Few works like [3][4][31][32] cope with the energy management problem in MCSs, without considering reliability and temperature constraints. In order to minimize the dynamic energy consumption of single-cores, Huang *et al*. [3] have proposed a DVFS-based optimal solution with EDF-VD scheduling algorithm which is applied to the normal operational mode of the system, where tasks of the same criticality level share the same frequency. The proposed method in [3] has been extended to multicores in [4]. The study in [31] has proposed an optimal solution for reducing the static energy consumption by applying the DPM technique in single-core MCSs. Volp *et al.* [32] have considered an energy budget for multi-core MCSs, which discard LC tasks whenever is needed. As mentioned before, the proposed methods in this category did not exploit fault-tolerant techniques. Therefore, the reliability constraints of a safety-critical system based on prominent standards are not satisfied.

The studies in [20] and [33] propose schemes that simultaneously support energy management, fault tolerance, and guaranteed service level for LC tasks in the mixed-criticality multicore systems that execute sporadic/periodic task models. Tasks in [20] are scheduled based on the Early Release Preference Oriented Earliest Deadline First (ER-POED) scheduling method, and the new Demand Bound Function (DBF) analysis is developed for checking the schedulability of periodic tasks. The purpose of the LETR-MC paper is to minimize the total energy consumption of the fault-tolerant system. The study in [33] proposed two schemes that exploit the standby-sparing technique to tolerate permanent faults and maintain the system reliability against transient faults with low energy overhead. Moreover, a Demand Bound Function schedulability analysis is proposed to guarantee timeliness and energy management. The reference [34] proposed a scheme that applies the standby-sparing technique for fault-tolerance while guaranteed an acceptable service level for LC tasks in overrun mode. However, in this category, all tasks can be executed concurrently which may violate the temperature constraints.

As it is mentioned, none of the previous works did consider satisfying thermal constraints for fault-tolerant MCSs. However, different system-level techniques consider power/temperature constraints for real-time systems or high-performance systems. In [16], without considering DVFS, the authors proposed a scheduling algorithm that finds a set of concurrent executable tasks, such that the design-time chip-level peak power consumption is minimized and all timing requirements are satisfied. The reference [35] has proposed a method that manages peak power overlaps between concurrently executing tasks to meet TDP in the fault-tolerant system. The references [36] and [38] have proposed a peak-power-aware energy management approach that satisfies TDP constraint, reliability requirements, and real-time constraints in the standby-sparing systems for frame-based and periodic applications, respectively. The aforementioned papers consider TDP to reduce the thermal violations. However, comprehensive studies [39][40] have demonstrated how satisfying the TDP constraint does not guarantee to avoid thermal violations. Once thermal violation occurs, DTM must be triggered to throttle down the cores. That is, however, not acceptable in real-time systems, because that might lead to violating timing constraints [41]. As a response, a new power budget concept, called Thermal Safe Power (TSP), has been presented in [17][42] which is an abstraction that provides safe power and power density constraints as a function of the number of simultaneously active cores. Several techniques have been then proposed to maximize the performance under TSP constraint, e.g., [37][43], while the technique proposed in [41] employs TSP to satisfy both timing and temperature constraints. The reference [45] has proposed a peak-power-aware reliability management scheme that meets the chip-level and core-level power constraints by exploiting code version programming and determines the number of replicas for each task to keep the system reliability at an acceptable level.

Apart from the proposed methods in the embedded system community, the works in [46] and [47] are examples of exploiting TDP in high-performance systems. The study in [46] controls the reliability by a temperature-aware module that cools the system to run below the TDP and reconfigures the hardware without sacrificing performance, at runtime. The work in [47] has focused on optimizing the energy consumption of power-budgeted data centers while minimizing its impact on application performance. Power capping makes it possible to add more nodes to the data center, each node running below its TDP value, while staying within the overall power budget of the data center. It is worthy to mention that the existing solutions that already consider reliability and timing constraints cannot be extended to consider temperature constraint. The reason is that the temperature constraint cannot be separately checked on individual cores that execute the tasks, while the other two constraints, i.e., reliability and timing, can be checked for each task separately from other tasks. More specifically, any potential scheduling decision, in which a task needs to be scheduled on a core, will have impacts on the temperatures of the cores that execute other tasks and might lead to thermal violations on those cores. Moreover, the number of active/idle cores on the chip will affect the temperatures of all cores, and thus, this information is also needed while considering temperature through the scheduling process. In contrast, considering reliability and timing constraints can be achieved on the level of the task, without affecting any other tasks on different cores, and there is no requirement for information about the active and idle cores. Therefore, considering temperature constraint is not straight forward and needs a new scheduling policy that takes into account the heat transfer between the different cores and the number of active/idle cores, in order to be able to satisfy temperature constraint besides timing and reliability constraint. Therefore, the purpose of this paper is to propose a scheme that satisfies timing, reliability, and temperature constraints while maximizes the QoS of low-criticality tasks in graph-based mixed-criticality multicore systems.

3 Models and Assumptions

In this section, we introduce the models and assumptions which are used throughout the rest of the paper.

3.1 System Model

We consider a multicore system with m identical cores. The thermal constraint of the system is referred to as T_{crit}, and it is one of the input parameters of the system. Per-core DVFS is available with a finite set of available voltage and frequency (vf) levels, i.e., $vf \in \{vf_{\min}, \ldots, vf_{max}\}$. The suitable voltage value for each frequency has been selected considering the non-linear relationship between frequency and voltage as explained in [49]. The power model consists of static and dynamic components [11][20]. The total power consumption of each core, P_i, can be written as:

$$P_i = P_{dynamic} + P_{static} = \alpha . C_{eff} . v_i^2 . f_i + v_i . I_{leak} + P_{ind} \quad (1)$$

TABLE 1
DO178B safety requirements [22]

ζ	A	B	C	D	E
PFH	$< 10^{-9}$	$< 10^{-7}$	$< 10^{-5}$	$\geq 10^{-5}$	-

where α, C_{eff}, v_i, and f_i are the activity factor of the task, effective switched capacitance, supply voltage, and the operating frequency of the core during the execution of task τ_i, respectively. Intuitively, down-scaling the vf levels will reduce power consumption. However, that will decrease the reliability of the tasks [50], as will be shown later in the reliability model. For the thermal model, we employ the state-of-the-art thermal model presented in [17].

3.2 Reliability Model

Since mixed-criticality embedded systems often control safety-critical applications, tolerating faults and achieving high reliability levels are of great importance; i.e., faults must be detected, and appropriate recovery tasks must be successfully completed before the deadlines. Faults can be classified as permanent, transient, or intermittent based on their occurrence and duration. Permanent faults result from hardware component failure or manufacturing defects. Recovery from this kind of fault is only possible by replacing or repairing the faulty component. Transient faults occur for a short time period and then disappear without physical damage to the processor. It is often induced by electromagnetic interference and cosmic radiation. Intermittent faults occur frequently, and it is difficult to detect because after its occurrence the system operates correctly. Transient faults are the most common type of faults, and their number is continuously increasing due to high complexity, smaller transistor sizes, higher operational frequency, and lower voltage levels. The rate of transient faults is often much higher compared to that of permanent faults. Transient-to-permanent fault ratios can vary between 2:1 and 100:1 or higher [48]. Therefore, in this paper, we consider transient faults.

Transient faults are usually assumed to follow a Poisson distribution with an average rate of λ [9][52]. Considering the effects of DVFS on transient fault rate, the fault rate at the scaled supply voltage $v_i=\rho_i v_{max}$ is modeled as [54][55]:

$$\lambda(\rho_i) = \lambda_0 10^{\frac{d(1-\rho_i)}{1-\rho_{min}}} \quad (2)$$

where λ_0 is the fault rate at the maximum voltage (v_{max}), ρ_{min} is the ratio of the minimum supply voltage v_{min} to maximum supply voltage v_{max}, and the exponent value d is a technology-dependent constant [36]. Considering $f_i=\sigma_i f_{max}$, the reliability of a task is defined as the probability of executing the task successfully, in the absence of transient faults [9]. Hence, the reliability of task τ_i is [54][55]:

$$R_i^{LO}(\rho_i,\sigma_i) = e^{-\lambda(\rho_i)\frac{W_i^{LO}}{\sigma_i}} \quad (3)$$

$$R_i^{HI}(\rho_i,\sigma_i) = e^{-\lambda(\rho_i)\frac{(W_i^{HI}-W_i^{LO})}{\sigma_i}} \quad (4)$$

where $\lambda(\rho_i)$ is given by Eq. 2 and W_i/σ_i is the execution time of τ_i when executed at $f_i=\sigma_i f_{max}$. Since the reliability of HC tasks depends on their WCET, Eq. 3 and Eq. 4 compute the reliability of each HC task based on low-level WCET and its execution time in overrun mode (W_i^{HI}-W_i^{LO}), respectively.

Since there might be a variation in the execution time of the tasks, we have designed the system based on WCETs similar to many state-of-the-art techniques ([11][12][51][57]). In this paper, we exploit NMR for fault-tolerance where there are N copies for each task. Therefore, the reliability of one copy of each HC task is computed based on Eq. 5.

$$R_i(\rho_i,\sigma_i) = R_i^{LO}(\rho_i,\sigma_i).R_i^{HI}(\rho_i,\sigma_i) \quad (5)$$

The reliability of each task τ_i by considering N copies is computed by the following equation, as explained in [11] [35][56]:

$$R_i^N(\rho_i,\sigma_i) = \sum_{\lceil N/2 \rceil}^{N} \binom{N}{j} R_i^j(\rho_i,\sigma_i)[1-R_i^j(\rho_i,\sigma_i)]^{N-j} \quad (6)$$

In MCSs, each criticality level has an important property, which is known as Probability of Failure per Hour (PFH=1-R) that represents the maximum probability of failure to which each task of that level must adapt. The DO-178B/C (aeronautics domain) [22], IEC61508 (generic electrical and/or electronic and/or programmable electronic (E/E/PE)) [23], and ISO26262 (automotive domain) [24] are three standards being the most commonly used in MCSs. In the DO-178B standard five criticality levels from A with highest, to E with lowest criticality levels are defined. Safety requirements of each criticality level are shown in Table 1. Hence, each task τ_i from HC task set must be guaranteed to be schedulable, even in presence of faults, to achieve a failure rate of at most PFH_i=PFH(ζ^{HI}). It should be noted that the space shuttle [58], X-38 Crew return vehicle [59], Boeing 777 [60], and the MARS system [61] are examples of real-world safety-critical embedded systems that exploiting more than two replications to satisfy the reliability target based on the considered safety standards.

3.3 Task Model

In this paper, we consider task graphs because, in certain applications, computational activities cannot be executed in an arbitrary order, and they have to respect precedence relations that are defined at the design time [34][62]. Such precedence relations are usually described through task graphs [62]. A task graph G(V, E), is a directed acyclic graph where each node (V_i), represents an individual task, and the edges represent the dependencies among these tasks, i.e., E represents only the edges between the nodes. A directed edge between two tasks shows that there is a data transfer between them.

We consider an MCS with two different criticality levels, which are denoted as high-criticality and low-criticality levels. Thus, the tasks can belong to any of these two criticalities out of the five criticality levels in the DO-178B standard [22]. The deadline of the whole graph is equal to D. Each task τ_i in a task graph has {ζ, W^{LO}, W^{HI}, X, s, e, O, vf^{τ}_{min}, vf^{τ}, P, A, Z} parameters:

- $\zeta \in \{LC, HC\}$ denotes the criticality level of τ_i.
- W^{LO}: The designer-specified WCET for τ_i.
- W^{HI}: The CAs-specified WCET for τ_i.

- X: The assigned core to τ_i.
- s: The start time of τ_i.
- e: The completion time of τ_i.
- O ∈ {*NO*, *OV*} denotes the operational mode of τ_i.
- $vf^{\tau}{}_{min}$: The minimum vf level which satisfies R_{target} for τ_i.
- vf^{τ}: Selected vf level for τ_i, i.e., either $vf^{\tau}{}_{min}$ or vf_{max}.
- $P(O, vf^{\tau})$: Peak power consumption of task τ_i throughout its execution time at the operational mode (O), i.e., W^{LO} or W^{HI}- W^{LO}, running at the selected vf level (vf^{τ}).
- A: List of successors of τ_i.
- Z: List of predecessors of τ_i.

We consider a task graph with NMR provision; i.e., there are N copies for each node. Although the copy nodes share the same predecessors and successors as the original node, they have some differences such as: the assigned core, the assigned vf level, and MSSAC factor which will be introduced in detail later. By considering N copies for each node, V length is $n\times N$, while E dimensions are $(n\times N)\times(n\times N)$. Here, n is the number of tasks in the task graph and N is the number of versions of each task (including the original version).

To store all dependencies between the nodes, we define a dependency matrix, $Dep_{(n\times N)\times(n\times N)}$, where $Dep[i,j]$=1, when task i directly depends on task j, i.e., E_{ij} =1, or when any parent of node i depends on task j. Hence, each node inherits the dependencies of its parents. Therefore, the list of successors for each node will be driven from the dependency matrix. Particularly, for a task, the 1 value in column j (corresponding to task τ_j) will indicate the successors of task j (the corresponding row indices represents the indices of successors). Contrarily, the one values in row i indicates the predecessors of task i. In the dual-criticality systems, if ζ_i=*LC*, $\tau_i.W_i^{HI}=\tau_i.W_i^{LO}$, otherwise $\tau_i.W^{LO}\leq \tau_i.W^{HI}$ [5][6]. Moreover, WCETs of the tasks are considered as the input of our proposed scheduling technique; therefore it is orthogonal to the approaches that analyze/estimate WCETs of the tasks on real systems [63]-[65].

4 Problem Definition

Given the task graph $G(V_{n\times N}, E_{(n\times N)\times(n\times N)})$, m homogenous cores, and set of vf levels, where $vf\in\{vf_{min},\ldots,vf_{max}\}$ levels, the goal of our method is to maximize the quality of service (QoS) of LC tasks while keeping the tasks' timing, reliability, and temperature constraints in mixed-criticality multicore systems. As introduced in Section 1, considering the temperature directly within the scheduling process will increase the complexity further, and therefore, we employ the thermal safe power (TSP) [17] constraint. TSP is calculated as a function of the number of the simultaneous active cores, referred to as AC, and it guarantees to satisfy the temperature constraint, T_{crit}. Mathematically, we can formulate the above problem as follows:

Optimization Goal: Maximize the total QoS of LC tasks:

$$Maximize \quad \text{QoS}_{total}(\text{G}) = \frac{\#\text{Executed LC tasks}}{\#\text{Total LC tasks}} \qquad (7)$$

Reliability Constraint: The vf level of each task must be selected so that the reliability of each task meets the reliability constraint as follows:

$$\forall \tau_i, \quad R_i^N(\rho_i,\sigma_i) \geq R_{target} \qquad (8)$$

TSP constraint: The summation of the power consumption of all cores (m) at time t should be less than the TSP for the current number of the active core at time t; i.e., $AC(t)$:

$$\text{At each time } t, \ \sum_{j=1}^{m} P_j(t) \leq TSP(AC(t)) \qquad (9)$$

Timing Constraint: The completion time of each HC task should be less than the deadline.

$$\forall \tau_i, \quad \tau_i.e \leq \tau_i.D, \text{ for all HC tasks} \qquad (10)$$

Task Graph Dependency Constraints: Dependency constraints between tasks should not be violated even after fault or overrun occurrence or applying DVFS. If there is a dependency between two nodes, the completion time of the previous node should be smaller than the start time of its dependent task.

$$\forall i,j, \quad \tau_i.e \leq \tau_j.s, \quad \text{if} \quad Dep[i,j]=1 \qquad (11)$$

This problem is known to be an NP-hard problem in a strong sense [16][17]. Therefore, finding an optimal solution will have exponential-time complexity. Therefore, we employ a heuristic for our thermal-aware scheduling that aims at maximizing the QoS of the LC tasks under the aforementioned constraints.

It is worthy to mention that our scheduling algorithm needs to give guarantees at the offline phase for meeting timing and thermal constraints. Therefore, the worst-case execution scenarios (including worst-case fault and overrun occurrence scenarios) and worst-case estimations for time and power are considered by our scheduling policy.

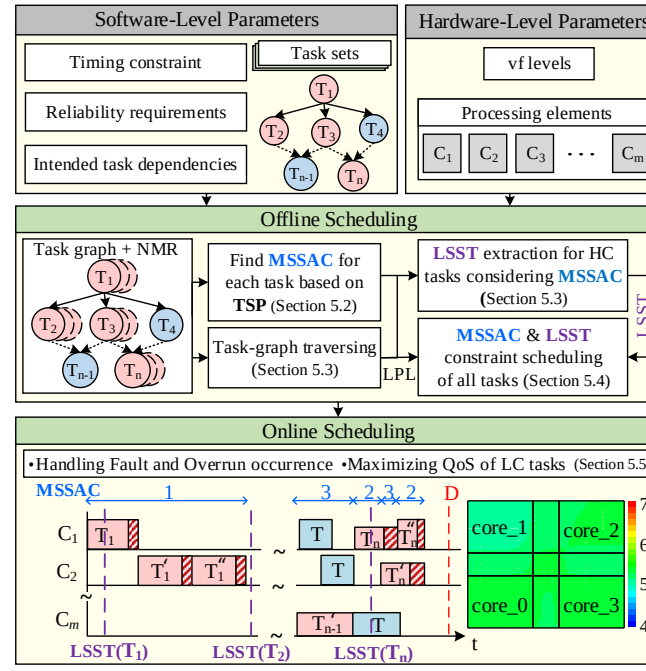


Fig. 3. An overview of our proposed TherMa-MiCs scheme.

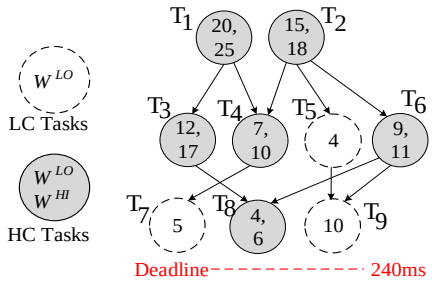


Fig. 4. An example task graph

5 Our Proposed Thermal-Aware Scheduling

We propose a thermal-aware scheduling method, named TherMa-MiCs, which consists of two phases; offline and online, as illustrated in Fig. 3. All necessary software and hardware information about the system will be inputs to our scheduler at the offline phase. This information includes tasks' WCETs, the worst-case power profile, deadline, dependencies between tasks, reliability target, and cores' vf levels. As it is shown in Fig. 3, the given task graph is extended to consider fault-tolerant provisions resulting from employing N Modular Redundancy (NMR) technique. After that, for each task, the minimum vf level that satisfies the reliability target, referred to vf^{τ}_{min}, will be extracted, as explained in Section 3. Then, our proposed Maximum Safe Simultaneous Active Cores (MSSAC) factor is computed for each task considering its vf^{τ}_{min} and vf_{max}, as explained in Section 5.2. This factor is used as a constraint in the scheduling process.

In order to schedule LC tasks besides the HC tasks (without violating the timing constraints of HC tasks), we need to extract first the latest safe start time (namely LSST) factor for each HC task, at which the task can start its execution and complete it before the deadline, while considering faulty and overrun situations, as well as all graph dependencies. To extract LSST factors, HC tasks and their replicas are assigned to cores and scheduled based on the proposed policy (Algorithm 2) from the deadline of the graph to start time while considering the graph dependencies and also MSSAC factor. Finally, the main scheduling policy (Algorithm 3) schedules all tasks from the start to the deadline with considering both LSST and MSSAC factors while attempting to schedule LC tasks on the available slack times without violating LSST and MSSAC constraints. In the online phase, the scheduler will exploit the slack times that might dynamically become available at runtime, as explained in Section 5.5, in order to schedule more LC tasks, thereby improving the QoS as much as possible. In the following subsections, all of the aforementioned steps will be explained in detail.

Illustrative example: Throughout the explanation of the steps of our TherMa-MiCs scheme, we present an illustrative example to demonstrate the functionality of each step and its inputs and outputs. In the illustrative example, we consider an example of a task graph with nine tasks, whose parameters are shown in Fig. 4. The deadline for the graph is equal to 240ms. It is worth noting that for the sake of simplicity to illustrate the steps of our proposed technique, the tasks used in this illustrative example are synthetically generated. Hence, the WCETs at different vf levels in Table 2, the MSSAC factors in Table 3, and the reliability are computed based on this task set. However, in Section 6 real applications from the MiBench benchmark suite [66][67] are considered. The target multicore system consists of four cores, where the available vf levels are vf_1=[1.016v 1.2GHz], vf_2=[1.055v 1.4GHz], vf_3=[1.118v 1.6GHz], and vf_4=[1.322v 2.0GHz]. More details about the experimental setup are explained in Section 6.

TABLE 2
Extracting the minimum acceptable vf level for each task, (i.e., vf^{τ}_{min}), and corresponding WCET at vf^{τ}_{min}.

Task	Low-Level WCET (ms) at different vf levels; vf_1, vf_2, vf_3, vf_4					Overrun WCET (W^{HI}-W^{LO}) (ms) at different vf levels; vf_1, vf_2, vf_3, vf_4				
	vf_1	vf_2	vf_3	vf_4	vf^{τ}_{min}	vf_1	vf_2	vf_3	vf_4	vf^{τ}_{min}
T_1	33	28	25	20	vf_4	8	7	6	5	vf_4
T_2	25	21	18	15	vf_2	5	4	3.7	3	vf_2
T_3	20	17	15	12	vf_3	8	7	6	5	vf_3
T_4	12	10	8	7	vf_2	5	4	3.7	3	vf_2
T_5	7	6	5	4	vf_2	-	-	-	-	-
T_6	15	13	11	9	vf_2	3.3	2.9	2.5	2	vf_2
T_7	8	7	6	5	vf_1	-	-	-	-	-
T_8	6	5.5	5	4	vf_1	3.3	2.9	2.5	2	vf_1
T_9	16	14	12	10	vf_1	-	-	-	-	-

TABLE 3
MSSAC factor for the normal and overrun parts of each task

Task	MSSAC at normal mode		MSSAC at overrun mode	
	vf^{τ}_{min}	vf_{max}	vf^{τ}_{min}	vf_{max}
T_1	1	1	1	1
T_2	3	1	3	1
T_3	2	1	2	1
T_4	3	1	4	1
T_5	3	1	-	-
T_6	3	2	3	2
T_7	4	2	-	-
T_8	4	2	4	2
T_9	4	1	-	-

5.1 Satisfying Reliability Target

To satisfy the reliability target, we employ for the first time NMR fault-tolerant technique [11][35] for MSCs. According to the NMR method, a majority voting is done, after executing the $\lceil N/2 \rceil$ copies of each task. When a fault occurs the remaining ($\lfloor N/2 \rfloor$) copies should be executed to perform the majority voting.

Applying DVFS for reducing the power consumption, increases the execution time of the task which results in lower reliability according to equations 2-6. In our proposed method, we consider the DVFS is applied to the first $\lceil N/2 \rceil$ copies of each task and the remaining replicas will be executed at the maximum vf level.

Algorithm 1. Find Maximum Safe Simultaneous Active Cores (MSSAC)

Inputs: $V_{n\times N}$: list of tasks, *TSPList*: List of TSPs for all # active cores
Output: *MSSAC Table*.

// find four MSSAC values for each task to cover all scenarios;
Function FindMSSAC(*V*, *TSPList*)
1. *start*⊚1;
2. *end*⊚ # cores;
3. **for** each $\tau_i \in V$ **&** each *O* ∈{*NO, OV*} **&** *each vf* ∈{$vf^{\tau_{min}}$, vf_{max}} **do**
4. *MSSAC*(τ_i,*O*,*vf*)⊚GreatestMSSAC (τ_i.*P*(*O*,*vf*),*start*,*end*,*TSPList*);
5. **return** *MSSAC*;

end function
function GreatestMSSAC (τ_i.*P*, *start*, *end*, *TSPList*)
1. **if** start ≥ end **then**
2. **return** infeasible;
3. *middle* = (*end*-*start*+1)/2;
4. **if** *P* ≤ *TSPList[middle]* **and** *P* > *TSPList[middle+1]* **then**
5. **return** *middle*;
6. **else if** *P* < *TSPList[middle]* **then**
7. **return** GreatestMSSAC (τ_i.*P*, *start*, *middle*, *TSPList*);
8. **else**
9. **return** GreatestMSSAC (τ_i.*P*, *middle*, *end*, *TSPList*);

End function

In the illustrative example (Fig. 4), we consider that Three Modular Redundancy (TMR) method is exploited for fault tolerance, where there are three copies for each HC task. Table 2 shows the low-level WCET (W^{LO}) and WCET at overrun mode (W^{HI}-W^{LO}), for the synthetic generated tasks shown in Fig. 4 at the available vf levels listed earlier. As it was mentioned before, applying DVFS will increase the execution time of the task and reduce the reliability of the task (Equations 2-6). In this table, the normal and overrun WCETs of each task at different vf levels are computed, i.e., vf_1 corresponds to the minimum vf level and vf_4 corresponds to the maximum vf level. Therefore, the WCET of the task at vf_4 is smaller than its WCET at vf_1. Then the reliability of each task at all vf levels is computed, and the minimum vf level which still satisfies the reliability target (based on equations 2-6) is determined in the gray column. For example, the normal WCET of T_2 when executing based on vf_2 is 21ms and this vf level is the minimum one that still satisfies the reliability target, and the lower vf levels are not acceptable for this task due to reliability requirements. Therefore, vf_2 is reported in the gray column regarding $vf^{\tau_{min}}$. Moreover, the overrun WCET of T_2 is computed in all vf levels, and as it is reported in the second part of the table, vf_2 is the minimum one which still satisfies the reliability target of T_2, and the overrun WCET of the task in this vf level is equal to 4ms. Therefore, the green colors show the WCET of the task in the normal and overrun modes corresponding to $vf^{\tau_{min}}$, and the corresponding vf level is reported in gray columns.

5.2 Maximum Safe Simultaneous Active Cores (MSSAC)

In order to consider temperature through scheduling, we derive the Maximum Safe Simultaneous Active Cores (MSSAC) factor for each task. To do this, TSP values are computed for all possibilities of the number of active cores using the algorithm proposed in [17], and the output is stored in TSPList (the details of generating TSPList are described in Fig. 8). Then, we employ a binary search to find how many cores can be concurrently active with each task based on its maximum power consumption, as demonstrated in Algorithm 1. In particular, there is a need to find the maximum number of active cores, at which the TSP value is equal or greater than the power consumption of the task. Importantly, there are four power consumption values that need to be considered for each task due to the following reasons: First, each HC task has two peak power values; one during the execution at the normal mode and another during the execution at the overrun mode. Second, some of the task replicas will be executed at $vf^{\tau_{min}}$, while others will be executed at vf_{max}. As a result, there are four MSSAC values for each HC task and two MSSAC values for each LC task, since they do not have overrun mode. Algorithm 1 generates the MSSAC table for the tasks. We consider that applying Algorithm 1 for the tasks in the illustrative example (Fig. 4) results in Table 3. Importantly, the table rows corresponding to the HC tasks will be replicated to represent the MSSAC factors for the HC replicas. It should be noted that the first and second replicas will be scheduled with the MSSAC at $vf^{\tau_{min}}$ while the third replica is scheduled with MSSAC at vf_{max}. The time complexity of Algorithm 1 is O((N^{LO}+ N^{HI})×log*C*), where N^{LO}, N^{HI}, and *C*, are the number of LC tasks, HC tasks, and cores, respectively.

During the scheduling process, at any given time interval the number of active cores should not exceed the MSSAC factors of all tasks that need to be scheduled at that time interval. For example, consider T_3, T_5, and T_6 from Fig. 4 which can be executed concurrently according to the task graph. As it can be seen in Table 3, T_3, T_5, T_6 have the MSSAC values of 2, 3, 3, respectively. To schedule these

TABLE 4
Longest path to leaves for the example task set in Fig. 4

Task	Path to leaves	Path lenght
HC: T_1	$T_1 \rightarrow T_3 \rightarrow T_8$	48
	$T_1 \rightarrow T_4 \rightarrow T_7$	40
HC: T_2	$T_2 \rightarrow T_4 \rightarrow T_7$	33
	$T_2 \rightarrow T_5 \rightarrow T_9$	32
	$T_2 \rightarrow T_6 \rightarrow T_8$	35
	$T_2 \rightarrow T_6 \rightarrow T_9$	39
HC: T_3	$T_3 \rightarrow T_8$	23
HC: T_4	$T_4 \rightarrow T_7$	15
LC: T_5	$T_5 \rightarrow T_9$	14
LC: T_6	$T_6 \rightarrow T_8$	17
	$T_6 \rightarrow T_9$	21
HC: T_7	T_7	5
LC: T_8	T_8	6
HC: T_9	T_9	10

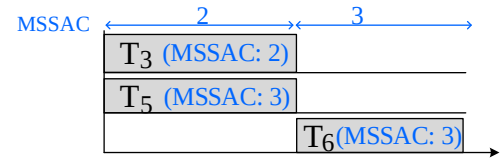


Fig. 5. A simple example of considering MSSAC in the scheduling.

Algorithm 2. Mapping and temporary scheduling from deadline to start to extract LSST

Inputs: *G*: Input task graph, *LPL_Q*: Priority task queue, *D*: Deadline, *MSSAC Table*.
Output: *X*: Task to core mapping, *LSST*: Latest Safe Start Time.

Function LSST (*G, LPL_Q, D, MSSAC*)
1. **while** *LPL_Q*≠ϕ **do**
2. τ_i ⊚ The first HC task of *LPL_Q*; // where *i* is the task index
3. $\tau_i.X$ ⊚ Select core based on WFD policy;
4. $\tau_i.vf^{\tau}$ ⊚ $vf^{\tau}{}_{min}$ for $\lfloor N/2 \rfloor$ replicas, and, vf_{max} for $\lfloor N/2 \rfloor$ replicas;
5. **if** $\tau_i.A=\phi$ **then** //Selected task does not have any successor
6. k ⊚ D;
7. **else**
8. k ⊚ min. start time of $\tau_i.A$;
9. **while** $k \geq \tau_i.W^{HI}$ at $\tau_i.vf^{\tau}$ **do**
10. ts⊚Find first free time slot before k on X_i;
11. Partition ts to ts^{LO} & ts^{HI} based on $\tau_i.W^{LO}$ & $\tau_i.W^{HI}$- $\tau_i.W^{LO}$;
12. $MSSAC^{LO}_{min}$⊚Min(MSSAC(τ_i,*NO*,$\tau_i.vf^{\tau}$), MSSACs of all tasks
-- at ts^{LO});
13. $MSSAC^{HI}_{min}$⊚Min(MSSAC(τ_i,*OV*, $\tau_i.vf^{\tau}$), MSSACs of all tasks
-- at ts^{HI});
14. **if** $MSSAC^{LO}_{min} \geq$ (*AC* at ts^{LO})+1 & $MSSAC^{HI}_{min} \geq$ (*AC* at ts^{HI})+1
-- **then**
15. Schedule τ_i at ts on $\tau_i.X$;
16. *LPL_Q*.Remove(τ_i);
17. **break;**
18. k ⊚ k-ts;
19. **if** τ_i is not scheduled **then**
20. **return** infeasible;
21. **for all** τ_i **do**
22. τ_i.LSST⊚Start time of first instance of each HC task;
23. **if** τ_i.LSST < 0 **then**
24. **return** infeasible;

End function

Algorithm 3. LSST- & MSSAC- Constrained scheduling

Inputs: *G*: Input task graph, *LPL_Q*: Priority task queue, *D*: Deadline, *LSST*, *MSSAC Table*, *X*: Task to core mapping.
Output: The main task scheduling.

Function mainScheduling (*G, LPL_Q, D, LSST, X, MSSAC*)
1. **while** *Q*≠ϕ **do**
2. τ_i⊚ The first task of *LPL_Q*;// where *i* is the task index
3. $\tau_i.vf^{\tau}$ ⊚ $vf^{\tau}{}_{min}$ for $\lfloor N/2 \rfloor$ replicas, and, vf_{max} for $\lfloor N/2 \rfloor$ replicas;
4. **if (**$\tau_i.\zeta$ = LC**) then**
5. $\tau_i.X$ ⊚ Select core based on WFD policy;
6. **if** $\tau_i.Z=\phi$ **then** //Selected task does not have any predecessors
7. k ⊚0;
8. **else**
9. k ⊚ max. end time of $\tau_i.Z$;
10. **while** $k \leq \tau_i.W^{HI}$ at $\tau_i.vf^{\tau}$ **do**
11. ts⊚Find first free time slot after k on $\tau_i.X$;
-- **// check MSSAC factors**
12. **if (**$\tau_i.\zeta$ = HC**) then**
13. Partition ts to ts^{LO} & ts^{HI} based on $\tau_i.W^{LO}$ & $\tau_i.W^{HI}$-$\tau_i.W^{LO}$;
14. $MSSAC^{LO}_{min}$⊚Min (MSSAC(τ_i,*NO*, $\tau_i.vf^{\tau}$), MSSACs of all
-- tasks at ts^{LO});
15. $MSSAC^{HI}_{min}$⊚Min (MSSAC (τ_i,*OV*, $\tau_i.vf^{\tau}$), MSSACs of all
-- tasks at ts^{HI});
16. **if** $MSSAC^{LO}_{min} \geq$(*AC* at ts^{LO})+1 **&** $MSSAC^{HI}_{min} \geq$(*AC* at ts^{HI})+1
-- **then // check LSST constraint**
17. **while (** start time of $ts > \tau_i.LSST$**) then**
18. Unschedule the LC task (LC_t) of the smallest LPL;
19. *LPL_Q*.add (LC_t);
20. Schedule τ_i at ts on $\tau_i.X$;
21. *LPL_Q*.Remove(τ_i);
22. **break;**
23. k ⊚ k+ts
24. **else if (**$\tau_i.\zeta$ = LC**) then**
25. MSSAC⊚Min. of (MSSAC(τ_i,*NO*, $\tau_i.vf^{\tau}$), MSSACs of all
-- tasks at ts);
26. **if** MSSAC ≥ (AC at ts) +1
27. Schedule τ_i at ts on $\tau_i.X$;
28. *LPL_Q*.Remove(τ_i);
29. **break;**
30. k ⊚ k+ts
31. **if (**$\tau_i.\zeta$ = LC**) &** is not scheduled **then**
32. *LPL_Q*.remove(τ_i);

End function

tasks, the scheduler will consider the minimum MSSAC factors among these three tasks (that can be run concurrently), which is equal to 2. That means, only two tasks can be scheduled in parallel to satisfy the MSSAC constraint, while the third one needs to be scheduled at the next time interval, as shown in Fig. 5.

5.3 Extracting LSST for HC tasks

The latest safe start time (LSST) for each HC task needs to be extracted to be able to schedule LC tasks without leading to violating timing constraints of HC tasks when a fault or overrun occurs [18]. By definition, LSST is the time at which the task can start its execution and complete it before the deadline. In order to guarantee meeting all constraints, the worst-case scenario must be considered in the offline scheduling. That implies, all HC tasks and their replicas need to be scheduled, and also the overrun parts of all of them need to be considered as well. Thus, to extract LSST factors, we map all HC tasks and their replicas to the cores and then schedule them in reverse order from the deadline of the graph in the schedule to the start, and the LSST of each task will be the start time of its first version in this reverse schedule.

For graph traversing during the scheduling, we employ the well-known list scheduling algorithm [68]. Specifically, we find the Longest Path to Leaves (LPL) for each node (the task of the graph), i.e., all paths from each node to the leaves of the graph are determined and the summation of the execution time of the nodes in the path is calculated. It is worthy to mention that each task will be represented by one only one node. Then the largest summation of the execution time of the nodes in the path from the mentioned node to the leaves is reported as the LPL of the task. For the HC tasks, their W^{HI} is considered, and their replicas have exactly the same LPL value. Table 4 shows all existing paths from each node to the leaves of the example task graph of Fig. 4, and the computed LPL for each node is determined by red color. The resulting LPL values are stored in a queue, referred to as *LPL_Q*. Importantly, the LPL values of the HC tasks will be replicated to represent the LPL for the HC task replicas. Then, *LPL_Q* is sorted in increasing order; Afterward, the scheduler selects the HC tasks

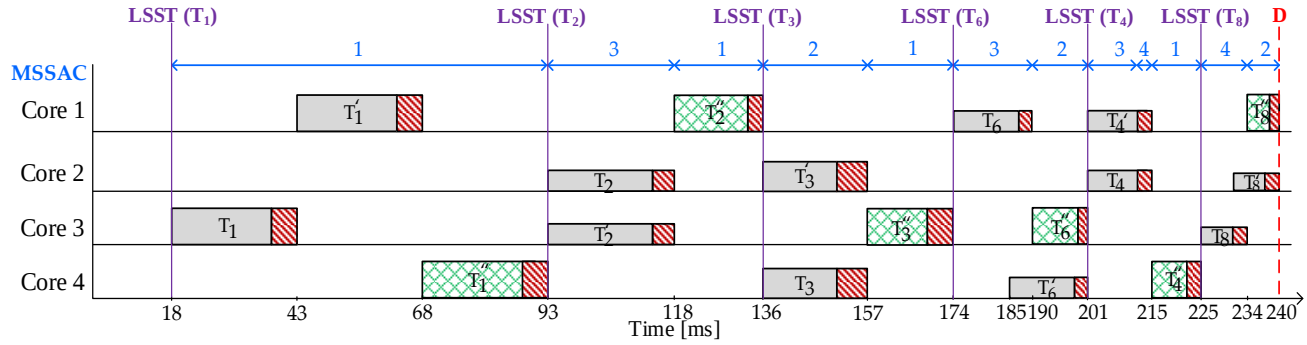


Fig. 6. Scheduling HC tasks **from deadline to start** for obtaining the LSST, constrained by the MSSAC factors of the tasks.

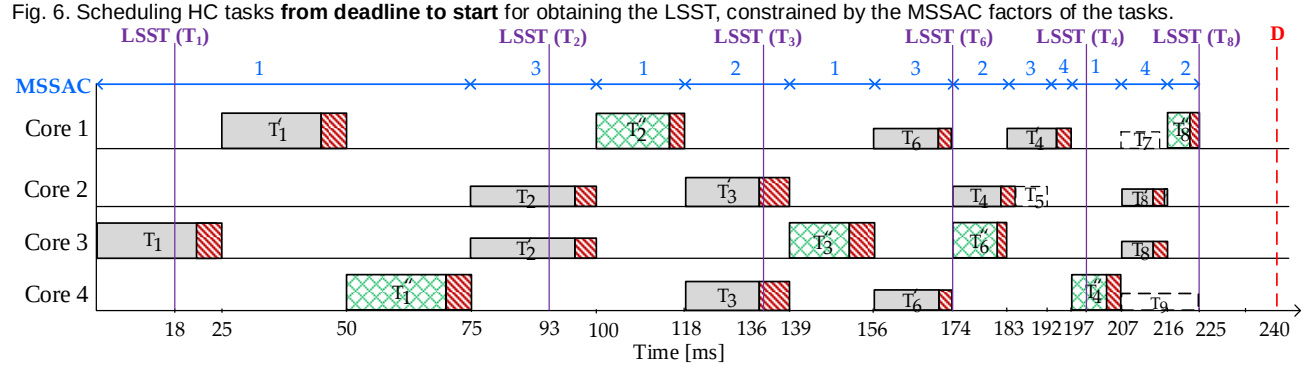


Fig. 7. The final offline scheduling of our proposed TherMa-MiCs **from start to deadline** that satisfies both timing and temperature constraints through employing LSST and MSSAC constraints. Some LC tasks could be scheduled in the available slack time on the cores.

from *LPL_Q*, one by one, and maps them to cores based on Worst Fit Decreasing (WFD) bin packing [6] which helps to exploit the parallelism inherent in multicores, thereby minimizing the application execution time [69]. While other heuristics like Next Fit Decreasing (NFD), First Fit Decreasing (FFD), and Best Fit Decreasing (BFD) aim to minimize the number of processors, and they often result in a highly unbalanced workload distribution [69]. The pseudo-code of this WFD-based mapping is shown in Algorithm 2 (lines 1-3). After mapping, our proposed scheduler selects the first possible start time of the task on the selected core starting from the deadline of the graph. In particular, the scheduler (in lines 4-8) determines the time interval that the selected task can potentially execute in it, considering the dependency constraints. That implies, if the selected task does not have any successors it can be scheduled at the first free time slot on the assigned core. Otherwise, it should be scheduled after all of its successors. The parameter k determines the potential start time for the selected task in reverse order. Line 10 determines the first free time slot (ts) before k on the assigned core (X_i) for the selected task τ_i. At first, the selected time slot ts is partitioned into two parts (ts^{LO} and ts^{HI}) based on the W^{LO} and W^{HI}-W^{LO} of the selected task at the corresponding vf level; i.e., $\lceil N/2 \rceil$ copies of each task will be executed at $\tau_i.vf^{\tau}{}_{min}$, and the $\lfloor N/2 \rfloor$ copies will be executed with $\tau_i.vf_{max}$ (line 11). It should be noted that although the normal and overrun parts of each version share the same vf level, they may have different MSSAC factors. Therefore, when the scheduler finds the free slot time for scheduling the whole task, it should consider the MSSAC value of both normal and overrun parts of the task. At both parts of the found time slot, the MSSAC constraint needs to be checked; i.e., the MSSAC of the selected task and all the tasks scheduled on other cores at each part of ts must be less than or equal to the total number of active cores (AC) at ts (as shown in lines 12-13). The MSSAC of the task will be derived from MSSAC Table 3 (the output of Algorithm 1), based on its operational mode and its selected vf level. If MSSAC constraint is satisfied for both the normal and overrun parts of the task (line 14), the task will be scheduled at ts (line 15) and removed from *LPL_Q* (line 16). Otherwise, the scheduler finds the next free time slot in line 18. If after traversing all free time slots, the task is still not scheduled, the algorithm returns infeasible in lines 19-20.

Table 4 shows the LPL for the tasks in the illustrative example. The rows belong to the HC tasks will be triplicated for replicas, and then *LPL_Q* will be sorted in increasing order. Fig. 6 depicts the mapping and scheduling of HC tasks with their corresponding replicas from the deadline of the graph based on dependency and MSSAC constraints. It should be noted that the first and second replicas will be scheduled considering their MSSAC for $vf^{\tau}{}_{min}$. The third replica of each task (represented by green color quadrants) should be executed in faulty mode at vf_{max} in both normal and overrun modes, and therefore, its MSSAC for vf_{max} will be considered. Note that the difference in the width of the rectangles (that indicate the tasks) shows the different vf levels.

To demonstrate how our scheduler finds the time slots that a task can be scheduled, we take T_8 in Fig. 6, as an example. T_8 has no successors and has the largest LPL among all tasks and should be scheduled first. To schedule the tasks, we start from the deadline which is 240ms, and traverse the schedule in reverse order from the deadline to the start of the schedule to find a free time slot for scheduling

the tasks. T_8'' and T_8' which have MSSAC equal to 2 are executed concurrently on their designated cores. When the scheduler wants to schedule T_8, finding the first free possible time slot to schedule the task is not enough to start the task at that time, because the MSSAC constraint needs to be checked. If the MSSAC constraint is not satisfied, the scheduler tests the next free time slot until satisfying the MSSAC constraint. In this example, T_8 is scheduled at 225ms. It should be noted that T_8 does not have any successors and therefore we only consider MSSAC. However, if there is a task that has some successors, its start time should be earlier than the earliest start time of all its copies. After scheduling all HC tasks, their corresponding replicas, and their overrun parts, the scheduler reports the start time of the first version of each HC task as the LSST of that task in lines 21-24.

The time complexity of Algorithm 2 is $O(N^{HI} \times t_s)$, where N^{HI}, and t_s are the number of HC tasks, and time slots, respectively.

5.4 LSST- & MSSAC-constrained scheduling

After extracting the LSST of HC tasks, we can schedule now all HC tasks and some LC tasks from the start of the scheduling, such that the latest safe start time (LSST) of HC tasks and the MSSAC constraints are not violated. Algorithm 3 shows the pseudo-code of scheduling all tasks from start time to deadline. In order to schedule tasks, we employ *LPL_Q* queue similar to the previous step, but the *LPL_Q* will be sorted here in decreasing order. Tasks are selected one by one from *LPL_Q* queue for scheduling in line 2. If the selected task is HC, it will be scheduled on its designated core which is driven from Algorithm 2. However, since the LC tasks did not schedule in the previous step, they should be mapped in lines 4-5. In lines 6-9, if the selected task does not have any predecessor, it can start its execution from time 0; otherwise, the maximum end time of all of its predecessors is assigned as the first possible start time of this task. However, the MSSAC constraint will be checked first. Thus, the selected task will be scheduled in the first free time slot after *k*, at which the MSSAC factors of both the normal and overrun parts of the task are satisfied (lines 11-15). Next, If the selected task is an LC, it will be scheduled in the first free time slot after *k*, at which the MSSAC factors of the task are satisfied (lines 11-15). Otherwise, the scheduler finds the next free time slot in line 17. After completing the loop, If the LC task is still not scheduled, it is removed from the *LPL_Q* because it can't schedule in any time slot (lines 18-19).

Similar to Algorithm 2, if the selected task is an HC one, the selected time slot *ts* is partitioned to ts^{LO} and ts^{HI} in lines 10-11. At both parts of *ts*, the MSSAC constraint needs to be checked, because the normal and overrun parts of each task may have different MSSAC factors; i.e., the MSSAC of the selected task and all the tasks scheduled at each part of *ts* must be less or equal the total number of active cores (*AC*) at *ts* (lines 14-15). If the MSSAC constraint is satisfied for both the normal and overrun parts of the task (line 16), the LSST constraint for HC tasks will be checked (lines 17-23). That implies, if scheduling LC tasks lead to LSST violation of an HC task, the scheduler starts to iteratively remove from the schedule the LC tasks with the smallest LPL value from *LPL_Q*, that have been scheduled before that HC task, one by one (line 18), until satisfying the LSST value of the HC task again. The removed LC tasks from the schedule will be added again to the *LPL_Q* to schedule in the next step (lines 19). This scheduling guarantees the execution of all HC tasks even in the overrun or faulty execution modes and attempts to schedule as many LC tasks as possible, considering the worst-case scenario.

Fig. 7 shows the final scheduling of HC tasks with their corresponding replicas, and guaranteed executable LC tasks based on MSSAC and LSST factors from the start to the deadline at design time. It is worthy to mention that employing fault-tolerant techniques has timing overhead. However, it is necessary to pay for this overhead in order to achieve a given reliability target, even if QoS has been reduced. Because ensuring the required reliability for HC is of great importance, otherwise, it will result in catastrophic consequences. Emerging multi-core systems produces great opportunities to exploit the inherent redundancies for executing more tasks concurrently. However, activating all cores at maximum vf levels might violate temperature constraints. Therefore, we propose to calculate the MSSAC factor for the tasks, which specifies the maximum simultaneous tasks that can be run with each other, without violating the temperature constraints. Then, the tasks will be scheduled in parallel if both the MSSAC factors and the dependency constraints between tasks are satisfied. This parallel execution of the tasks on different cores allows scheduling more LC tasks and thereby QoS will be improved without violating the predefined timing, thermal, and reliability constraints.

The time complexity of Algorithm 3 is $\text{Max}\{O(N^{HI} \times t_s \times N^{LO}), O(N^{LO} \times t_s)\}$, where N^{HI}, N^{LO}, and t_s are the number of HC tasks, LC tasks, and time slots, respectively.

5.5 Runtime scheduling

At runtime, the available slack times are used for improving the QoS of LC tasks by considering the dependency and MSSAC constraints. Slack times at runtime might release due to replica or overrun cancelation. In particular, at runtime, the tasks will be scheduled according to the precomputed schedule. However, if overrun or fault occurrence scenarios do not occur, and dynamic slack times are released, then LC tasks can be scheduled on the fly without impacting the already scheduled tasks. For each resulting slack time slot, *ts*, the scheduler finds first the LC tasks, whose predecessors are completely executed, and then selects the task with the largest LPL in *LPL_Q*. If the execution time of the task at vf^{t}_{min} is equal to or less than the *ts*, then the MSSAC is checked for the feasibility of scheduling under temperature constraints. If the MSSAC is not satisfied, this task cannot schedule at this free slack time. In case the execution time of the task is bigger than the slack time (*ts*), the scheduler checks whether the execution time of the task at vf_{max} is equal or less than the *ts*, and again checks the MSSAC constraint. If the selected task is not schedulable, the next candidate task will be checked for scheduling in *ts*. This process will be iterated for all released slack times.

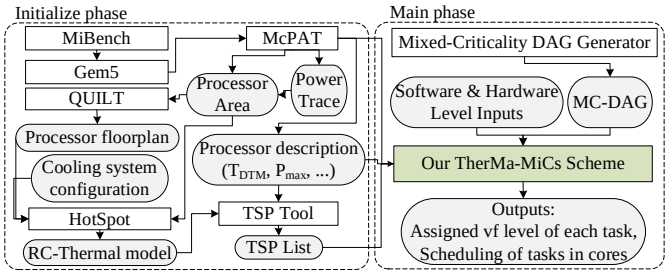


Fig. 8. The tool flow adopted in our experiments.

It should be noted that the time complexity of algorithms 1, 2, and 3 of the offline phase is computed in section 5. Regardless of the actual execution time (which depends on many parameters and the selected execution scenario), the computations are done in the offline phase. Moreover, the lightweight online manager uses the available released slack times only for improving the QoS of LC tasks, if needed, by scheduling them on the fly without impacting the already scheduled tasks. Therefore, according to our experimental results, the time overhead of the runtime scheduler is up to 0.57% for the scenarios that half of the tasks are LC tasks. This overhead indicates the ratio of the time required by the online scheduler to the total execution time of the tasks. This overhead is considered negligible [52]. Nevertheless, to account for this overhead and guarantee satisfying timing constraints, we have considered the overhead of the scheduler as a part of the task's WCET similar to the state-of-the-art scheduling policies in the real-time community [18][52][53].

6 Results and Discussion

In this section, we evaluate the effectiveness of our proposed TherMa-MiCs scheme in terms of feasibility, temperature, QoS, and reliability. The tool flow of our experiments is shown in Fig. 8. We consider an ARM processor, which is widely used in many embedded systems, based on simulations conducted on gem5 [70] and McPAT [71] for 22nm with out-of-order Cortex-A15 cores. These simulators, i.e., gem5 and McPAT, extract the power profiles, the worst-case execution time, and chip area. The simulation configurations are written in Table 5. The tool QUILT [72] generates the chip floorplan based on the results of McPAT. For calculating TSPList, the RC thermal model of the chip is required to model the thermal behavior of the processor. The RC thermal model can be extracted using the HotSpot simulator [73] with providing the floorplan of the processor and the configuration of its cooling system. Based on the algorithm introduced in [17] the TSP for worst-case mappings for the pre-defined thermal threshold is calculated. To generate task graphs, we modified the open-source DAG generator for the mixed-criticality systems' tool which is introduced in [18]. The tasks of each DAG are selected from the MiBench benchmark suite [11][66][67], which is commonly used in the embedded system community It is worthy to mention that this benchmark includes a variety of programs in different embedded system areas such as automotive and industrial control, consumer devices, office automation, networking, security, and telecommunications. MiBench has been used by many recent state-of-the-art techniques in the embedded community [67][76].

TABLE 5
The details of simulation configuration

Name	Configuration
Core Type	ARM Cortex-A15
Core Microarch.	ARMv7-A
Machine Type	Out of Order
Feature Size	22nm
# Cores	4, 9, 16, 36
Core vf level	19 vf levels [0.9v, 0.2GHz] to [1.3v, 2.0GHz]
L1 Cache	32KB, 8KB block-width, 4-way
L2 Cache	2MB, 16-way
Memory	2GB, 32-bit LPDDR3e
Chip Thickness	0.15mm
Heat Sink Thickness	1mm
Spreader Thickness	0.1 mm

We generated 100 random DAGs where each generated DAG has a different topology (configuration) from others in terms of the number of nodes (tasks), the connection between nodes, and parallelism degree (low-, middle-, and high-parallelism) [35]. It is known that the height of a task graph can be used to determine the parallelism degree for task graphs with a specific number of tasks. Consider n is the number of tasks in a task graph and h is the task graph height, h can vary between 1 (the highest parallelism degree) and n (a chained task graph with the lowest parallelism degree). Therefore, we consider the following classes: 1) task graphs with high parallelism degrees whose heights are $1 \leq h \leq n/3$; 2) task graphs with medium parallelism degrees whose heights are $n/3 \leq h \leq 2n/3$; and 3) task graphs with low parallelism degrees whose heights are $2n/3 \leq h \leq n$. We compare TherMa-MiCs with the following state-of-the-art methods:

- LE-NMR: A scheduling technique which is proposed in [11] that considers an NMR and aims at reducing the energy overhead of the fault-tolerant technique (NMR) in hard real-time multicore embedded systems. This technique executes the tasks in two phases: the indispensable phase and the on-demand phase. When a task has no faults during the indispensable phase, the time which is reserved for its copies in the on-demand phase is reclaimed to significantly reduce power and energy. We choose LE-NMR to highlight that reducing power and energy is not enough to avoid thermal violations.
- CNMR: It is conventional or classic NMR, where all N copies of each task are executed in parallel [56]. The CNMR technique does not consider power or temperature constraints through scheduling the tasks.
- Medina [18]: This technique proposed a scheduling policy for graph-based mixed-criticality tasks. However, it does not consider any fault-tolerant

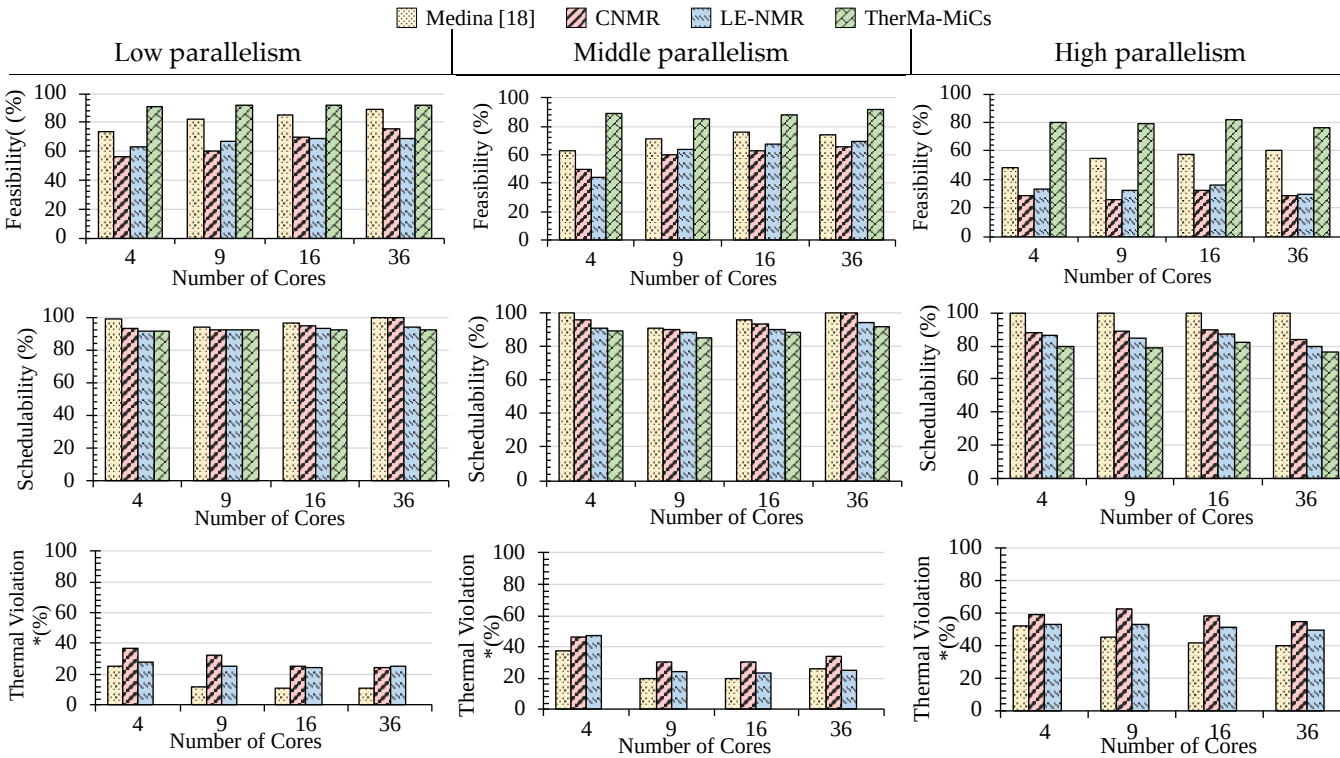


Fig. 9. Comparing the resulting a) Feasibility, b) Schedulability, and c) Thermal violation from the offline scheduling of our proposed method, TherMa-MiCs, with the scheduling policies in the state-of-the-art methods. The feasibility is defined as the percentage of satisfying both timing and temperature constraints in the resulting schedule. Our TherMa-MiCs achieves higher feasibility with 33.47% on average. The schedulability is defined as the percentage of satisfying timing constraints in the resulting schedule.
* these are the thermal violations when timing constraint is satisfied.

provisions and drops all LC tasks in the overrun mode.

As it can be noticed our TherMa-MiCs, LE-NMR, and CNMR employ NMR to satisfy the reliability target. We evaluate those techniques for different N, i.e., N=3, N=5, and N=7. However, due to space limitations, we show the evaluation results of feasibility, temperature, and QoS only for N=3. Nevertheless, in Section 6.4, where the reliability is evaluated, we show the results for N=3, N=5, and N=7, since they are relevant in that evaluation.

6.1 Evaluating the Feasibility

In Fig. 9 we reported our comparison with three state-of-the-art techniques in terms of feasibility with different parallelism degrees of the DAG and different numbers of cores. The feasibility is defined as the percentage of satisfying both timing and temperature constraints in the resulting schedule. Moreover, to have a better understanding of the resulting feasibility numbers, we show the schedulability, which is the percentage of satisfying timing constraints in the resulting schedule, and the percentage of thermal violations when timing constraints are satisfied. The results are reported from the offline scheduling (Section 5.4) that considers the worst-case scenario where all copies of tasks and all overrun parts will be executed completely. The parallelism degree is increased from low parallelism to high parallelism. According to Fig. 9, the proposed method in [18] has the highest schedulability since it is not fault-tolerant. However, our proposed TherMa-MiCs method has lower schedulability, since it considers temperature constraint through scheduling in conjunction with applying DVFS. However, since other methods are not temperature-aware, by increasing the parallelism degree the temperature will increase and more thermal violations occurred, and it can be observed in Fig. 9 how the state-of-the-art methods suffer from high thermal violations, and thereby low feasibility, while our TherMa-MiCs does not lead to any thermal violations, which ultimately helps to increase the resulting feasibility. The reason that Medina [18] has higher feasibility in comparison to others (except TherMa-MiCs) is that it is not fault-tolerant and does not consider replicas.

6.2 Evaluating the Temperature

In addition to the evaluation of the thermal violation percentage in the above section, in this subsection, we evaluate the resulting peak temperature after applying the scheduling of the comparison candidates for different numbers of cores and different parallelism degrees. Here, the realistic execution scenario at runtime has been considered, where all overrun parts will be executed completely and the fault rate is derived from Eq. 2. As shown in Fig. 10, TherMa-MiCs meets the temperature constraint, T_{crit}= 60 ℃ in all scenarios [14]. Note that TherMa-MiCs also meets the temperature constraint when N=5 and N=7, but as aforementioned, these additional results have not been shown due to space limitation. Other methods do not consider temperature constraints. Hence, in these methods tasks will be executed in parallel even in the realistic exe-

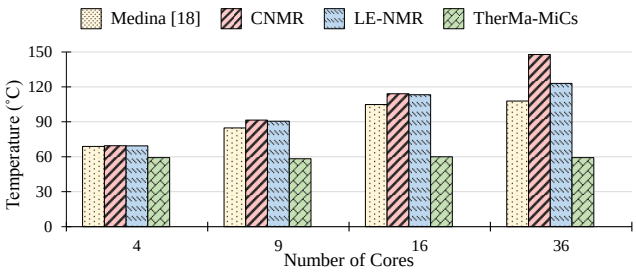


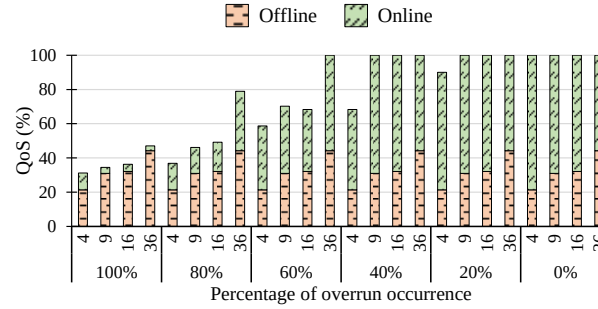
Fig. 10. The resulting peak temperature on the chip. Our TherMa-MiCs satisfies the thermal constraint, while the state-of-the-art techniques violate it.

cution scenario; i.e., there are some situations that even after canceling the overrun or replica parts, the number of the active cores will be beyond the safe limit and temperature constraint is violated. As it is shown in the figure, in higher number of cores, the maximum temperature increases because the number of active cores is increased.

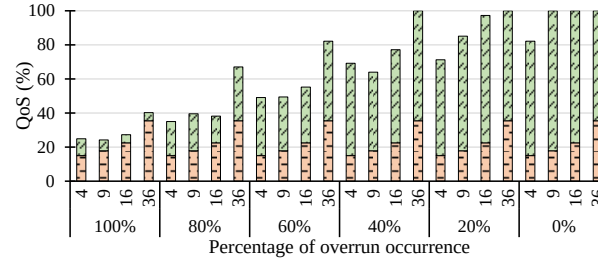
We would like to clarify that the temperature constraint is the input of the system and the scheduler computes the MSSAC factor based on the predefined temperature constraints. Therefore, increasing or decreasing the target temperature does not change the policy of the proposed method and our proposed method is applicable for any temperature target. However, decreasing the value of the temperature target may lead to a decrease in the MSSAC factors of the tasks, and thus the number of tasks that can be executed in parallel will be reduced. That means it decreases the number of LC tasks that can be executed in parallel with the HC tasks, and thereby the QoS is decreased. Moreover, it might also decrease the number of HC tasks that can be executed in parallel with other HC tasks, and in this case, some HC tasks will be delayed to the next intervals, and they ultimately might miss their deadlines, and thereby the feasibility is decreased. Contrarily, increasing the value of the temperature constraint will allow increasing the number of HC tasks and LC tasks that can run in parallel, and thereby the feasibility of HC tasks and the OoS of LC tasks will be improved.

6.3 Evaluating the QoS

The QoS is defined as the ratio of the number of LC tasks that the scheduler guarantees to execute them to the original number of LC tasks [1][30][74][75]. Fig. 11 represents the QoS of TherMa-MiCs in the offline and online phases with different percentages of LC tasks. In this evaluation, task graphs with different parallelism degrees are considered. Fig. 11 consists of two main scenarios: a) Fig. 11a shows the results when less than half of the total number of tasks are LC, and b) Fig. 11b shows the results when more than half of tasks are LC. Moreover, additional scenarios are considered which represent the different percentages of overrun occurrences, and also different numbers of cores are considered. In each figure, the orange bar shows the achieved QoS in the offline phase for different scenarios. The green bars show the amount of QoS improvement in the online phase in addition to the offline phase. The height of the bar represents the total QoS.



a. Percentage of LC tasks to the total number of tasks is less than 50%.



b. Percentage of LC tasks to the total number of tasks is more than 50%.

Fig. 11. QoS of the offline and online phases.

By changing the percentage of HC tasks that overrun in different numbers of cores, the QoS improvement is evaluated. For each number of cores and overrun percentage in Fig. 11, 100 DAGs with different configurations are generated, while in Fig. 11a and Fig. 11b the percentage of LC tasks is less than 50% and more than 50% of total existing tasks in the system, respectively. Then the final result in each bar is the average of achieved 100 results. In the online phase, due to replica and overrun cancellation, dynamic slack times are released which can be used for further scheduling LC tasks to improve the QoS. For example, in Fig. 11a, when the overrun percentage is equal to 60%, in a quad-core system, the QoS of the offline phase is 20%. In the online phase, by exploiting released dynamic slack times the QoS reaches 60%. The 44% is the average QoS of all of these scenarios. In Fig. 11a and Fig. 11b, by increasing the percentage of overrun occurrence, the QoS improvement is decreased because the amount of released dynamic slack is reduced. By increasing the number of cores, the QoS is improved because there are more available resources to schedule LC tasks in advance and an online phase. The QoS improvement of Fig. 11b is lower than Fig. 11a. Due to the MSSAC factor, by increasing the number of LC tasks, the number of LC tasks that can be executed concurrently is reduced. Moreover, when the number of LC tasks is lower than HC tasks (Fig. 11a), in the case that overrun does not occur in the online phase, the slack time will be released and the LC tasks can be executed in the released dynamic slack times. Therefore, in the online phase, the QoS can be reached 100% in most scenarios. However, when the number of LC tasks is more than HC tasks (Fig. 11b), since the number of HC tasks is low, even if overrun does not occur, the amount of released slack time in the

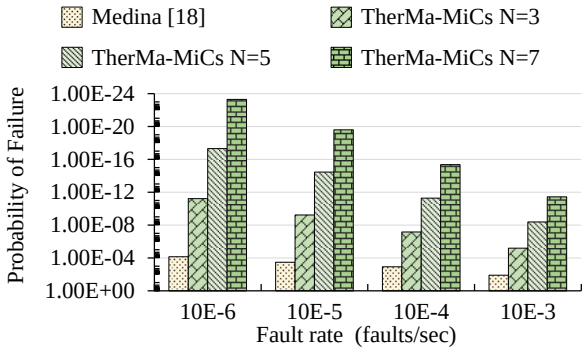


Fig. 12. Comparison of PoF at different fault-rates.

online phase is not efficient to execute all LC tasks. Therefore, in fewer scenarios, the QoS can be reached 100%.

6.4 Evaluating the Reliability

In this paper, the fault rate was modeled using Eq. 2 under parameters $\lambda_0 = 10^{-6}$ and d=3 [11][52]. Therefore, the fault rate varies between 10^{-6} fault/s to 10^{-3} fault/s, corresponding to the maximum and minimum voltage levels [11]. Fig. 12 reports the Probability of Failure (PoF) of applications for Medina [18] and TherMa-MiCs with N=3, N=5, and N=7. We assume that HC tasks are selected from levels A, B, or C in the DO-178B standard [22]. Regarding the selected level, the proper values of N will be selected because the higher value of N satisfies a higher reliability target (i.e., lower PoF). Obviously, Medina [18] fails in tolerating faults and meeting the reliability target at different fault-rates, since it does not employ the fault-tolerant technique.

6.5 Pessimism and Optimality Discussion

Enforcing MSSAC factors that are based on TSP involves some pessimism compared to enforcing temperature constraint directly, due to two reasons. First, TSP is derived based on the worst-case mapping. Secondly, MSSAC factors have been derived based on the peak power consumption of the tasks. On the other hand, solving the defined scheduling problem while considering the temperature directly will further increase the problem complexity, which grows exponentially, as explained earlier. Therefore, we proposed to solve this problem using TSP which enables us to enforce temperature constraint in an abstracted way. This pessimism manifests itself as a reduction in the QoS of the LC tasks.

To examine how far our scheduler is from the optimal solution we have implemented an exhaustive search, which is possible only for small size examples. In the exhaustive search, instead of computing the LPL queue for sorting the tasks (in order to select them for mapping and scheduling) and exploiting any bin packing algorithm for assigning tasks to cores, all combinations of task-to-core mapping options and their scheduling will be checked. Moreover, the optimal solution should directly check the temperature in all time slots during the scheduling, instead of considering the MSSAC factors. We were able to conduct an exhaustive search for only small-size graphs running on a quad-core system. For this small size example, we find the QoS obtained by the optimal solution is the same as the QoS obtained by our algorithm. However, the optimal solution could result in better QoS for bigger task graphs and bigger systems, but the increase in the overhead of obtaining the optimal schedule will grow exponentially along with increasing the input size; i.e., number of HC and LC tasks, number of cores, and number of time slots. Therefore, we could not perform an exhaustive search for a bigger input size.

Additional evaluation of our scheduler could be conducted by comparing it to a solution that considers more combinations than our scheduler, but still not all combinations, because this will not be possible, as explained above. We derive this solution as follows: HC tasks are sorted based on the LPL queue and are mapped based on WFD bin packing. Then HC tasks are scheduled based on computed MSSAC factors. However, for scheduling LC tasks more combinations will be checked, by considering the temperature of the cores at each time slot instead of using the MSSAC factors. By applying this solution, 16% improvement in the QoS has been observed. However, as elaborated in the paper, considering the temperature directly will not be possible in the actual run of the scheduler, due to the high overhead. Nevertheless, this experiment provides us with an estimation about unexploited optimization room (or pessimism) for the QoS by our scheduler compared to a better solution that considers more combinations, but is not feasible in the actual implementation at runtime.

7 Conclusion

In this paper, we proposed the TherMa-MiCs scheme that simultaneously satisfies the timing, temperature, and reliability constraints of the high-criticality tasks, while aiming at maximizing the QoS of low-criticality tasks under the predefined constraints. It exploits the inherent redundancy of multicore platforms for employing the NMR technique to ensure the reliability of MCSs. However, applying fault-tolerant techniques which increase the number of active cores might increase on-chip temperatures beyond safe limits. Therefore, our proposed TherMa-MiCs scheme enforces the thermal safe power (TSP) constraint within the scheduling process to guarantee avoiding thermal violations. Moreover, TherMa-MiCs satisfies the timing constraints for the HC tasks, while at the same time improving the QoS of the LC tasks with an average of 44% without violating timing and temperature constraints.

Acknowledgment

This work is funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Fndn.) – Proj. Nb. 146371743 – TRR 89 Invasive Computing

References

[1] A. Burns, and R. I. Davis, "A survey of research into mixed-criticality systems," *Journal ACM Computing Surveys*, vol. 50, no. 6, 2018.

[2] D. De Niz, K. Lakshmanan, and R. Rajkumar, "On the scheduling of mixed-criticality real-time task sets," *30th IEEE Real-Time Systems Symposium (RTSS)*, Washington, DC, 2009.

[3] P. Huang, P. Kumar, G. Giannopoulou, and L. Thiele, "Energy-efficient DVFS scheduling for mixed-criticality systems," *International Conference on Embedded Software (EMSOFT)*, 2014.

[4] S. Narayana, P. Huang, G. Giannopoulou, L. Thiele, and R. V. Prasad, "Exploring energy saving for mixed-criticality systems on multi-cores," *IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, Vienna, 2016.

[5] H. Su, N. Guan, and D. Zhu, "Service guarantee exploration for mixed-criticality systems," *Int'l Conf. on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, Chongqing, 2014.

[6] H. Su, D. Zhu, and S. Brandt, "An elastic mixed-criticality task model and early-release EDF scheduling algorithms," *ACM ACM Trans. on Design Automation of Electronic Systems (TODAES)*, vol. 22, no. 2, pp. 1-28, 2016.

[7] N. Guan, P. Ekberg, M. Stigge, and W. Yi, "Effective and efficient scheduling of certifiable mixed-criticality sporadic task systems," *IEEE Real-Time Systems Symposium (RTSS)*, 2011.

[8] C. Gu, N. Guan, J. Yu, Y. Wang, and QX. Deng, "Partitioned scheduling policies on multi-processor mixed-criticality systems," *Journal of Software*, vol. 25, pp. 284-297, 2014.

[9] Y. Zhang, and K. Chakrabarty, "Energy-aware adaptive checkpointing in embedded real-time systems," *IEEE Auto. and Test in Europe Conf. and Exhibition (DATE)*, 2003.

[10] H. Khdr, H. Amrouch and J. Henkel, "Aging-constrained performance optimization for multi cores," *55th ACM/ESDA/IEEE Design Automation Conf. (DAC)*, pp. 1-6, 2018.

[11] M. Salehi, A. Ejlali, and B.M. Al-Hashimi, "Two-phase low-energy N-Modular Redundancy for hard real-time multi-core systems," *IEEE Trans. on Parallel and Distributed Systems (TPDS)*, vol. 25, no. 4, pp. 1024-1033, 2015.

[12] R. M. Pathan, "Fault-tolerant and real-time scheduling for mixed-criticality systems," *Real-Time Syst.*, vol. 50, no. 4, pp. 509-547, 2014.

[13] Intel® Xeon Phi™ Coprocessor x100 Product Family, Datasheet, April 2015.

[14] Qualcomm Technologies, Inc., "Thermal debugging guide - Qualcomm developer network," Sep. 2016.

[15] https://ieeexplore.ieee.org/ielx7/12/7847504/7524768/tc-khdr-2595560-mm.zip?tp=&arnumber=7524768

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

[17] S. Pagani, et al., "Thermal Safe Power (TSP): Efficient power budgeting for heterogeneous manycore systems in dark silicon," *IEEE Trans. on Computers (TC)*, vol. 66, no. 1, pp. 147-162, 2017.

[18] R. Medina, E. Borde, and L. Pautet, "Directed acyclic graph scheduling for mixed-criticality systems," *22nd Int'l Conf. on Reliable Software Technologies - Ada-Europe*, 2017.

[19] D. de Niz, and L. T.X. Phan, "Partitioned scheduling of multi-modal mixed-criticality real-time systems on multiprocessor platforms," *19th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, Berlin, 2014.

[20] S. Safari, M. Ansari, G. Ershadi, and S. Hessabi, "On the scheduling of energy-aware fault-tolerant mixed-criticality multicore systems with service guarantee exploration," *IEEE Trans. on Parallel and Distributed Systems (TPDS)*, vol. 30, no. 10, pp. 2338-2354, 2019.

[21] L. Bauer et al., "Analyses and Architectures for Mixed-Critical Systems: Industry Trends and Research Perspective Special Session Extended Abstract," *International Conference on Embedded Software (EMSOFT)*, 2019, pp. 1-2.

[22] DO-178C (2011) Software considerations in airborne systems and equipment certification. RTCA, Inc.

[23] IEC61508 (2010) Functional safety of electrical/electronic/programmable electronic safety-related systems, IEC.

[24] ISO26262 (2011) Road vehicles-functional safety, ISO.

[25] S. Baruah, V. Bonifaci, G. Dangelo, H. Li, A. Marchetti-Spaccamela, S. Van der Ster, L. Stougie "The preemptive uniprocessor scheduling of mixed-criticality implicit-deadline sporadic task systems," *Euromicro Conf. on Real-Time Systems (ECRTS)*, 2012.

[26] D Liu, J Spasic, N Guan, G Chen, S Liu, and T Stefanov, "EDF-VD scheduling of mixed-criticality systems with degraded quality guarantees," *IEEE Real-Time Systems Symposium (RTSS)*, 2016.

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

[28] J. Lin, A. M. K. Cheng, D. Steel, and M. Yu-Chi Wu "Scheduling mixed-criticality real-time tasks in a fault-tolerant system," *RTSS*, 2014.

[29] A. Thekkilakattil, R. Dobrin, and S. Punnekkat, "Mixed criticality scheduling in fault-tolerant distributed real-time systems, " *Int'l Conf. on Embedded Syst. (ICES)*, 2014.

[30] Z. Al-bayati, J. Caplan, B. H. Meyer, and H. Zeng, "A four-mode model for efficient fault-tolerant mixed-criticality systems," *IEEE Design Auto. and Test in Europe Conf. and Exhibition (DATE)*, 2016.

[31] V. Legout, M. Jan, and L. Pautet, "Mixed-criticality multiprocessor real-time systems: Energy consumption vs deadline misses," *Workshop on Real-Time Mixed Criticality Syst. (ReTiMiCS)*, 2013.

[32] M. Völp, M. Hähnel, and A. Lackorzynski, "Has energy surpassed timeliness? Scheduling energy-constrained mixed-criticality systems," *19th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2014.

[33] A. Naghavi, S. Safari, and S. Hessabi, "Tolerating permanent faults with low-energy overhead in multicore mixed-criticality systems," *IEEE Trans. on Emerging Topics in Computing (TETC)*, 2021.

[34] S. Safari, G. Ershadi, and S. Hessabi, "LESS-MICS: A low energy standby-sparing scheme for mixed-criticality systems," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 39, no. 12, pp. 4601-4610, 2020.

[35] 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 Trans. on Parallel and Distributed Systems (TPDS)*, vol. 30, no. 1, pp. 161-173, 2019.

[36] M. Ansari, A. Yeganeh-Khaksar, S. Safari, and A. Ejlali, "Peak-power-aware energy management for periodic real-time applications," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 39, no. 4, pp. 779-788, 2019.

[37] H. Khdr, S. Pagani, E. Sousa, V. Lari, A. Pathania, F. Hannig, M. Shafique, J. Teich, and J. Henkel, "Power density-aware resource management for heterogeneous tiled multicores," *IEEE Trans. on Computers (TC)*, vol. 66, no. 3, pp. 488-501, 2017.

[38] M. Ansari, M. Salehi, S. Safari, A. Ejlali and M. Shafique, "Peak-power-aware primary-backup technique for efficient fault-tolerance in multicore embedded systems," *IEEE Access*, vol. 8, pp. 142843-142857, 2020.

[39] H. Khdr, S. Pagani, M. Shafique and J. Henkel, "Thermal constrained resource management for mixed ILP-TLP workloads in dark silicon chips," *52nd ACM/EDAC/IEEE Design Auto. Conf. (DAC)*, 2015, pp. 1-6.

[40] J. Henkel, H. Khdr, and M. Rapp, "Smart Thermal Management for Heterogeneous Multicores (Special Session)," *IEEE/ACM 22nd Design, Auto. and Test in Europe Conf. (DATE)*, 2019.

[41] B. Pourmohseni, F. Smirnov, H. Khdr, S. Wildermann, J. Teich, and J. Henkel, "Thermally composable hybrid application mapping for real-time applications in heterogeneous many-core systems," *IEEE Real-Time Systems Symp. (RTSS)*, Hong Kong, 2019.

[42] H. Khdr, H. Amrouch, and J. Henkel, "Aging-aware boosting," *IEEE Trans. on Computers (TC)*, vil. 67, no. 9, pp. 1217-1230, 2018.

[43] S. Pagani, H. Khdr, W. Munawar, J-J Chen, M. Shafique, M. Li, and J. Henkel, "TSP: Thermal Safe Power-Efficient power budgeting for many-core systems in dark dilicon," *Int'l Conf. on Hardware-Software Codesign and System Synthesis (CODES+ISSS)*, 2014.

[44] H. Khdr, S. Pagani, E. Sousa, V. Lari, A. Pathania, F. Hannig, M. Shafique, J. Teich, and J. Henkel, "Power density-aware resource management for heterogeneous tiled multicores," *IEEE Trans. on Computers (TC)*, vol. 66, no. 3, pp. 488-501, 2017.

[45] M. Ansari, J. Saberlatibari, S. M. Pasandideh, and A. Ejlali, "Simultaneous Management of Peak-Power and Reliability in Heterogeneous Multicore Embedded Systems," *IEEE Trans. on Parallel and Distributed Systems (TPDS)*, vol. 31, no. 3, pp. 623-633, 2020.

[46] B. Acun et al., "Power, reliability, and performance: One system to rule them all," *in Computer*, vol. 49, no. 10, pp. 30-37, Oct. 2016.

[47] A. Langer, H. Dokania, L. V. Kalé and U. S. Palekar, "Analyzing energy-time tradeoff in power overprovisioned HPC data centers," *IEEE Int'l Parallel and Distributed Processing Symp. Workshop*, 2015.

[48] H. Kopetz, R. Obermaisser, P. Peti, and N. Suri, "From a Federated to an Integrated Architecture for Dependable Embedded Real-Time Systems," Tech. Rep. 22, TU Vienna, 2003.

[49] J. Henkel, H. Khdr, S. Pagani, and M. Shafique, "New trends in dark silicon," *ACM/EDAC/IEEE Design Auto. Conf. (DAC)*, 2015.

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

[51] A. Yeganeh-Khaksar, M. Ansari, S. Safari, S. Yari-Karin, and A. Ejlali, "Ring-DVFS: Reliability-aware reinforcement learning-based DVFS for real-time embedded systems," *IEEE Embedded Systems Letters*, 2020.

[52] A. Ejlali, B. M. Al-Hashimi, and P. Eles, "Low-energy standby-sparing for hard real-time systems," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 31, no. 3, pp. 329–342, 2012.

[53] R. Medina, E. Borde, and L. Pautet, "Scheduling multi-periodic mixed-criticality DAGs on multi-core architectures," *IEEE Real-Time Systems Sympo. (RTSS)*, 2018.

[54] B. W. Johnson, Design and Analysis of Fault Tolerant Digital Systems. Addison-Wesley Longman Publishing Co., Inc., USA. 1988.

[55] K. S. Trivedi, Probability and statistics with reliability, queuing and computer science applications, (2nd edition), John Wiley and Sons Ltd., GBR. 2001.

[56] I. Koren, and C.M. Krishna, "Fault-Tolerant Systems," Morgan Kufman, Elsevier, San Fransisco, 2007.

[57] M. A. Haque, H. Aydin, and D. Zhu, "On reliability management of energy-aware real-time systems through task replication," *IEEE TPDS*, vol. 28, no. 3, pp. 813-825, 2017.

[58] J. R. Sklaroff, "Redundancy management technique for space shuttle computers," *IBM Journal of Research and Development*, vol. 20, no. 1, pp. 20-28, 1976.

[59] L. E. P. Rice, and A. M. K. Cheng, "Timing analysis of the X-38 space station Crew Return Vehicle avionics," *Proc. IEEE-CS Real-Time Technology and Applications Symp.*, 1999.

[60] Y. C. B. Yeh, "Design Considerations in Boeing 777 Fly-By-Wire Computers," *Proc. IEEE Int'l High-Assurance Systems Engineering Symp.*, 1998.

[61] H. Kopetz, H. Kantz, G. Grunsteidl, P. Puschner, and J. Reisinger, "Tolerating transient faults in MARS," *Int'l Symp. Fault-Tolerant Comput.*, 1990.

[62] D. Socci, P. Poplavko, S. Bensalem, and M. Bozga, "Multiprocessor scheduling of precedence-constrained mixed critical jobs," *IEEE Int'l Symp. on Real-Time Distributed Computing*, 2015.

[63] http://www.mrtc.mdh.se/projects/wcet/sweet.html.

[64] "aiT Worst-Case Execution Time Analyzer - Homepage," http://www.absint.com/ait/, 2009.

[65] S. Petersson, et al. "Using a WCET analysis tool in real-time systems education," *WCET*, 2005.

[66] M. R. Guthaus, et. al., "MiBench: A free, commercially representative embedded benchmark suite," *4th IEEE Ann. Workshop Workload Characterization*, pp. 3–14, 2001.

[67] "MiBench homepage." [Online]. Available: http://vhosts.eecs.umich.edu/mibench/.[Accessed: Nov-2020].

[68] T. L. Adam, K. M. Chandy, and J. R. Dickson, "A comparison of list schedules for parallel processing systems," *Communications of the ACM*, vol. 17, no. 12, pp 685-690, 1974.

[69] T. Wei, P. Mishra, K. Wu, and H. Liang, "Fixed-priority allocation and scheduling for energy-efficient fault tolerance in hard real-time multiprocessor systems," *IEEE Trans. on Parallel and Distributed Systems (TPDS)*, vol. 19, no. 11, pp. 1511-1526, Nov. 2008.

[70] N. Binkert, et. al., "The gem5 simulator," *ACM SIGARCH Comput. Archit. News*, vol. 39, no. 2, pp. 1-7, May 2011.

[71] S. Li, et. al., "McPAT: An integrated power, area, and timing modeling framework for multicore and manycore architectures," *Annual Int'l Symp. on Microarchitecture,* 2009.

[72] G. J. Briggs, E. J. Tan, N. A. Nelson, and D. H. Albonesi, "QUILT: a GUI-based integrated circuit floorplanning environment for computer architecture research and education," *Workshop on Computer Architecture Education*, 2005.

[73] W. Huang, et. al., "HotSpot: A compact thermal modeling methodology for early-stage VLSI design," in *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, vol. 14, no. 5, pp. 501-513, 2006.

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

[75] E. Yip, M. M. Kuo, P. S. Roop, and D. Broman, "Relaxing the synchronous approach for mixed-criticality systems," *IEEE Real-Time and Embedded Technology and Applications Symp.*, 2014.

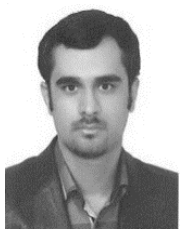
[76] S. K. Mandal, G. Bhat, J. R. Doppa, P. P. Pande, and U. Y. Ogras, "An energy-aware online learning framework for resource management in heterogeneous platforms," *ACM Trans. on Design Auto. of Electronic Systems*, no. 25, vol. 3, 2020.

Sepideh Safari received the M.Sc. degree in computer engineering from Sharif University of Technology, Tehran, Iran, in 2016 with an excellent grade and the first rank. She received the Ph.D. degree in computer engineering from Sharif University of Technology, Tehran, Iran, in 2021. She was a visiting researcher in the Chair

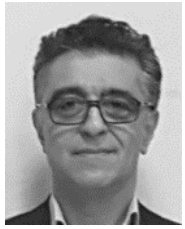
for Embedded Systems (CES), Karlsruhe Institute of Technology (KIT), Germany, from 2019 to 2021. Her research interests include low-power design of cyber-physical systems, energy management in fault-tolerant embedded systems, and multi-/many-core systems with a focus on dependability/reliability.

Heba Khdr is a postdoctoral researcher and a group leader at the Chair for embedded Systems (CES) in Karlsruhe Institute of Technology (KIT) in Germany. She received her Ph.D. (Dr.-Ing.) in Computer Science from Karlsruhe Institute of Technology (KIT) in 2018.
In 2005, she received her Diploma in Informatics Engineering from Aleppo University in Syria with an excellent grade and the first rank. From 2005 until 2007 she worked as a software engineer in in the industry sector in Syria. She worked as an assistant in Aleppo University from 2008 until 2010. In 2011 she did an equivalent master thesis at KIT.
Her research interests are thermal management and resource management in multi- and many-core systems. In 2012 she received Research Student Award from KIT. She received Best Paper Award from IEEE/ACM International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS) in 2014, and four HiPEAC paper awards.

Pourya Gohari-Nazari received the B.Sc. degree in computer engineering from the University of Isfahan. He is currently working toward the M.Sc. degree in the Department of Computer Engineering at Sharif University of Technology, Tehran, Iran. His research interests are thermal management in multi-/many-core systems and design of embedded systems with a focus on low-power and reliability.



Mohsen Ansari received his M.Sc. and Ph.D. degrees in computer engineering from Sharif University of Technology, Tehran, Iran, in 2016 and 2021, respectively. He was a visiting researcher in the Chair for Embedded Systems (CES), Karlsruhe Institute of Technology (KIT), Germany, from 2019 to 2021. Now, he is a postdoctoral researcher and a research group leader of Embedded Systems Research Laboratory (ESR-LAB), and a lecturer at the department of computer science and engineering, Sharif University of Technology. His research interests include low-power design of embedded systems and multi-/many-core systems with a focus on dependability/reliability.



Shaahin Hessabi received the BS and MS degrees in electrical engineering from Sharif University of Technology, Tehran, Iran, in 1986 and 1990, respectively, and the Ph.D. degree in electrical and computer engineering from the University of Waterloo, Ontario, Canada. He joined Sharif University of Technology, in 1996. Since 2007, he has been an associate professor in the Department of Computer Engineering, Sharif University of Technology, Tehran, Iran. He has published more than 100 refereed papers in the related areas. His research interests include cyber-physical systems, reconfigurable and heterogeneous architectures, network-on-chip, and system-on-chip. He has served as the program chair, general chair, and program committee member of various conferences, like DATE, NOCS, NoCArch, and CADS.

Jörg Henkel (M'95-SM'01-F'15) is currently with the Karlsruhe Institute of Technology (KIT), Germany, where he is directing the Chair for Embedded Systems (CES). Prof. Henkel received the masters and the Ph.D. (Summa cum laude) degrees, both from the Technical University of Braunschweig, Germany. He then joined the NEC Laboratories, Princeton, NJ, USA. His current research interests include design and architectures for embedded systems with focus on low power and reliability. Prof. Henkel has received the 2008 DATE Best Paper Award, the 2009 IEEE/ACM William J. Mc Calla ICCAD Best Paper Award, the CODES+ISSS 2011, 2014 and 2015 Best Paper Awards. He was the General Chair of major CAD events incl. ICCAD and ESWeek. He is the Chairman of the IEEE Computer Society, Germany Section, and was the Editor-in-Chief of the ACM Transactions on Embedded Computing Systems for two terms. He is currently the Editor-in-Chief of the IEEE Design&Test Magazine. He is also an Initiator and Spokesperson of the national priority program on Dependable Embedded Systems of the German Science Foundation and the site coordinator (Karlsruhe site) of the three-university collaborative research center on invasive computing. He is a Fellow of the IEEE and holds ten US patents.