Received 7 February 2025, accepted 13 February 2025, date of publication 17 February 2025, date of current version 24 February 2025.

Digital Object Identifier 10.1109/ACCESS.2025.3542472

RESEARCH ARTICLE

ReTMiC: Reliability-Aware Thermal Management in Multicore Mixed-Criticality Embedded Systems

SEPIDEH SAFARI[1], MOHSEN ANSARI[1,2], SHAAHIN HESSABI[2], (Member, IEEE), AND JÖRG HENKEL[3], (Fellow, IEEE)
[1]School of Computer Science, Institute for Research in Fundamental Sciences (IPM), Tehran 19395-5746, Iran
[2]Department of Computer Science and Engineering, Sharif University of Technology, Tehran 11365-11155, Iran
[3]Department of Computer Science, Karlsruhe Institute of Technology (KIT), 76131 Karlsruhe, Germany

Corresponding author: Mohsen Ansari (ansari@sharif.edu)

This work was supported in part by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Project 146371743-TRR 89 Invasive Computing; and in part by the Sharif University of Technology under Grant G4030811.

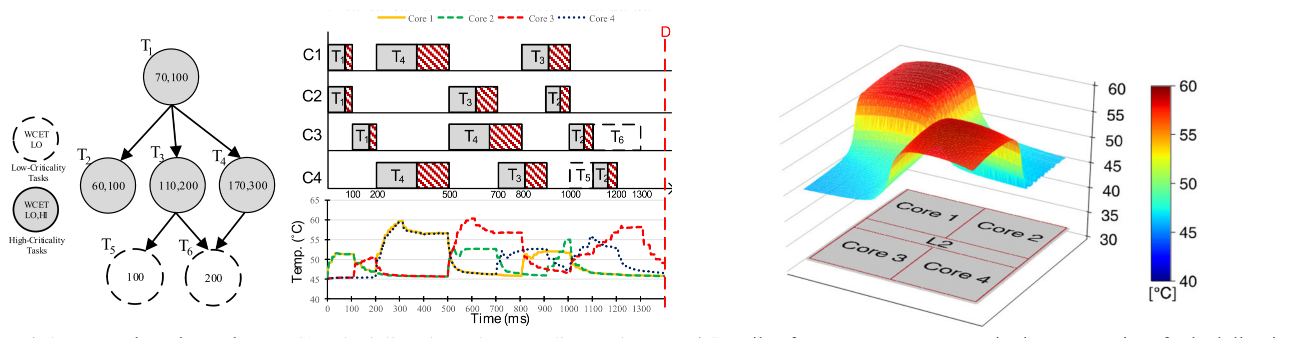
ABSTRACT As the number of cores in multicore platforms increases, temperature constraints may prevent powering all cores simultaneously at maximum voltage and frequency level. Thermal hot spots and unbalanced temperatures between the processing cores may degrade the reliability. This paper introduces a reliability-aware thermal management scheduling (ReTMiC) method for mixed-criticality embedded systems. In this regard, ReTMiC meets Thermal Design Power as the chip-level power constraint at design time. In order to balance the temperature of the processing cores, our proposed method determines balancing points on each frame of the scheduling, and at run time, our proposed lightweight online re-mapping technique is activated at each determined balancing point for balancing the temperature of the processing cores. The online mechanism exploits the proposed temperature-aware factor to reduce the system's temperature based on the current temperature of processing cores and the behavior of their corresponding running tasks. Our experimental results show that the ReTMiC method achieves up to 12.8°C reduction in the chip temperature and 3.5°C reduction in spatial thermal variation in comparison to the state-of-the-art techniques while keeping the system reliability at a required level.

INDEX TERMS Mixed-criticality systems, multicore platforms, task replication, embedded systems, thermal balancing.

I. INTRODUCTION

Mixed-Criticality Systems (MCSs) integrate a large number of tasks with different levels of criticality on the same computing platform, meeting stringent non-functional requirements relating to cost, space, heat generation, and power consumption [1], [2]. By aggressively scaling the feature size, multicore platforms are becoming the dominant trend for developing MCSs [3]. However, by increasing the number of cores, the power consumption of the chip increases, hence the on-chip temperature is elevated [4], [5], [6]. Violating the temperature constraint of a chip or unbalanced temperature will ineluctably accelerate processor wear-out which finally leads to earlier permanent faults occurrences, and the system lifetime drastically decayed. Therefore, if the wear-out failures are not taken into consideration during the task assignment to cores and scheduling process, or urgent countermeasures are not exploited to mitigate the temperature effects, some processors might age faster than others and become the reliability bottleneck for the whole embedded system, thus significantly reducing the lifetime reliability. Moreover, when the temperature constraint of a chip is violated, Dynamic Thermal Management (DTM) techniques such as Dynamic Voltage and Frequency Scaling (DVFS) are automatically activated to prevent hardware failure. However, DTM techniques such as throttling down the voltage and frequency levels will degrade the application reliability in terms of increasing the transient fault rate or

The associate editor coordinating the review of this manuscript and approving it for publication was Liang-Bi Chen.



a) An example task graph.

b) Scheduling the tasks according to the TP3M policy [13] for the worst-case scenario.

c) Details of processor temperature in the worst point of scheduling in terms of thermal balance.

FIGURE 1. **Motivational example of unbalanced thermal in scheduling policy with considering peak power consumption.**

leading to missing timing constraints [5], [7]. In order to prevent such consequences and due to the Thermal Design Power (TDP) constraint (as the chip-level power constraint), it may not be possible to simultaneously power on all processing cores at the full performance [8]. Since TDP is the maximum sustainable power that a multicore chip can safely consume, violating TDP triggers a performance throttling mechanism through lowering the voltage and frequency levels or power-gating to prevent possible overheating problems [7]. However, these techniques may lead to violating the timing constraints of MCSs [57]. Additionally, due to mobility, power, reliability, and size limitations, the exploitation of cooling mechanisms such as fans and conventional heat sinks is not a common procedure in embedded systems [58]. Moreover, overheating problems such as thermal hot spots and unbalanced temperatures may lead to performance losses and jeopardize the processing cores' reliability through unbalanced aging of different cores in multicore platforms [5].

There are two main approaches to temperature balancing techniques among processing cores. The first approach is based on task migration at run time to distribute the power consumption between the processing cores, which will achieve temperature flattening [9], [16]. The second approach focuses on reducing the power consumption of the processing cores to perform thermal management by employing the DVFS and Dynamic Power Management (DPM) techniques [7]. However, task migration and applying frequent DVFS may have a significant time overheads [10], [37].

Apart from thermal and timing issues, similar to other electronic systems, MCSs are susceptible to transient faults, which are considered as one of the severe reliability concerns that are induced by the technology scaling [59]. To ensure the reliable operation of MCSs, many fault-tolerant techniques have been employed to satisfy a given reliability target. However, fault-tolerant techniques (due to exploiting time, hardware, software, or information redundancies) incur significant time and power/energy overheads to the multicore MCSs [59]. The study in [59] provides a comprehensive survey about applying different fault-tolerant techniques in real-time systems with considered goals and constraints of the mapping and scheduling policies. The fault-tolerant techniques, due to simultaneously executing multiple versions of a task in parallel, may have temperature overhead which may lead to thermal violation by activating all cores concurrently. Fault-tolerant techniques can be classified into three broad categories based on exploited redundancy types including: i) hardware-based, ii) software-based, and iii) hybrid techniques [60]. Hardware-based techniques add extra hardware modules which changes the original architecture of the system or its components. Therefore, such techniques must be implemented during the design of the system. Hardware-based techniques have a high cost, verification and testing time, and area overhead which leads to higher power consumption as well. Software-based techniques exploit the concepts of software, time, and information redundancies to detect faults during the execution of the program [59]. The exploitation of the re-execution fault-tolerant technique in mixed-criticality systems is presented in [11], [61], [62], and [64]. The study in [65] supports on-demand redundancy which exploits dual modular redundancy (DMR), TMR, and passive replication in MCSs. However, none of these works have considered power/energy and temperature management. The study in [37] tolerates permanent faults by applying the standby-sparing technique with low energy overhead in independent periodic mixed-criticality systems. The study in [12] has introduced a parallelism and reduction policy in every primary-backup pair of the multi-core platform to increase the quality of service (QoS) of low-criticality tasks in a mixed-criticality system. The study in [3] proposed the LETR-MC scheme that employs the task replication fault-tolerant technique to satisfy reliability constraints while meeting timing and energy constraints in independent periodic mixed-criticality multi-core systems. Task replication is a fault-tolerant technique that, if intelligently used, can result in lower power and time overheads [13], [14]. The task replication technique considers different copies for each task based on its reliability target, and as soon as successful completion of the first copy of each task, the remaining copies will be dropped to reduce the time and power overheads [3].

In addition to timing, thermal, and reliability issues, considering the Quality of Service (QoS) of low-criticality (LC) tasks is another challenge in designing MCSs [3], [12]. Quality of Service of LC tasks has different definitions in the literature. Some research work selectively executes some LC tasks in the overrun mode. Some other research works, reduce the execution time of the LC tasks whenever it is necessary, in which each task has a lower execution time in comparison to its original worst-case execution time which leads to lower QoS. In the independent periodic/sporadic task model by increasing the periods of the LC tasks in the overrun mode the QoS (the ratio of the number of executed jobs to the original number of jobs for each LC task) will be decreased [22], [23]. Moreover, there exist some research works that drop all LC tasks in the overrun mode [2], [18], [19]. Employing thermal management and fault-tolerant techniques increases the probability of dropping more LC tasks in the overrun or fault occurrence scenario. Dropping all LC tasks in MCSs is not acceptable [3], [12]. Therefore, in this paper, we guarantee a minimum required QoS for LC tasks in all operation modes of the system.

In the following motivational example, the problem of the unbalanced temperature of the task replication mechanism in multicore mixed-criticality embedded systems is discussed.

A. MOTIVATIONAL EXAMPLE

Let us consider a quad-core processor with 3W of TDP constraint [13] that executes a task graph with six tasks $\{T_1, \ldots, T_6\}$. Fig. 1a shows dependencies between the tasks where the two numbers at each node show the low-level and high-level worst-case execution times (WCET) of the corresponding task, i.e., WC^{LO} and WC^{HI}, respectively. The tasks share a common deadline of $D = 1400$ms. Fig. 1b shows the scheduling of the given task graph with the TP3M method presented in [13] for the worst-case fault scenario. The TP3M method considers N-Modular Redundancy (NMR) as the fault-tolerant technique, where each task has N copies, and their results are compared to perform a complete majority voting. In this example, the TP3M method uses TMR (i.e., NMR with $N = 3$) and each task has three copies. The TP3M method tries to schedule the tasks that meet the chip TDP. Although this method meets the system's power constraint, as shown in Fig. 1c, it cannot balance the temperature of the cores efficiently. In Fig. 1c, the differences between cores' temperatures in TP3M method are shown; the chip temperature is unbalanced. Therefore, TP3M cannot be an efficient method in terms of thermal balancing. Since it employs peak-power-aware real-time scheduling, it cannot balance the temperature of the processing cores. Hence, it is necessary to present a method for balancing the system's temperature to avoid the negative effects of thermal unbalancing, such as reliability degradation and unbalanced aging.

1) CONTRIBUTIONS

In this paper, we define the problem of mixed-criticality task mapping and scheduling on a multicore embedded system when exploiting the dynamic task replication technique while meeting timing, reliability, and temperature constraints at design time. While in the runtime the temperature is balanced and the QoS is improved.

The main contributions of this paper are:

- Proposing a new dynamic task replication technique for graph-based mixed-criticality tasks to tolerate transient fault occurrence.
- Proposing peak-power-aware scheduling to meet the TDP constraint through extracting Balancing Points (BP) and Balancing Factors (BF) at the offline phase to be exploited at run time for flattening temperature in graph-based mixed-criticality embedded systems.
- Proposing a lightweight online manager to further reduce the system temperature and evenly distribute the heat on the surface of the chip through a run-time task re-mapping technique to increase the lifetime reliability of the whole system.
- Guaranteeing a minimum acceptable QoS for LC tasks at the offline phase and improving it at the online phase while meeting timing, reliability, and temperature constraints.

2) PAPER ORGANIZATION

The remaining paper is arranged as follows. Section II provides a literature review on the subject, followed by the required models and assumptions in Section III. Our proposed method and algorithms are presented in Section IV. Experimental results are presented in Section V, and Section VI concludes the paper.

II. RELATED WORK

The related work can be classified to three different categories.

A. SCHEDULING ALGORITHMS IN MIXED-CRITICALITY SYSTEMS WITHOUT CONSIDERING FAULT-TOLERANCE AND PEAK-POWER/TEMPERATURE MANAGEMENT

Mixed-criticality systems were first introduced by Vestal [17]. Different scheduling algorithms are proposed for periodic/sporadic MCSs, such as Earliest Deadline First with Virtual Deadline (EDF-VD) [18], [19], [20], and [21] which mostly discard all LC tasks after entering the overrun mode, and Early Release-Earliest Deadline First (ER-EDF) [22] and [23], which increases the period of LC tasks in the overrun mode for guaranteeing QoS. The previous mentioned papers consider single core platforms, however, there are some studies such as [24], [25], and [26] which consider multicore systems. Moreover, there are some research study which consider scheduling of DAG-based tasks set on multicore mixed-criticality systems [27], [28], [38], and [39]. Apart from proposed scheduling algorithms for DAG-based

TABLE 1. Overview of the related work.

Reference	Application Model	Real-time Model	Architecture Model	Fault-tolerance Technique	Temperature / Peak-Power Management
[17][18][19][20][21]	Sporadic	Mixed-Criticality	Single-core	×	×
[22][23]	Periodic	Mixed-Criticality	Single-core	×	×
[24]	Sporadic	Mixed-Criticality	Multi-core	×	×
[25][26]	Periodic	Mixed-Criticality	Multi-core	×	×
[27][28][38][39]	Sporadic DAG	Mixed-Criticality	Multi-core	×	×
[72]	DAG	Hard Real-time	Heterogeneous Multi-core	×	×
[11]	Sporadic	Mixed-Criticality	Single-core/multi-core	Re-execution	×
[29][62][63]	Periodic/Sporadic	Mixed-Criticality	Single-core	Re-execution	×
[61]	Periodic	Mixed-Criticality	Multi-core	Re-execution	×
[64]	Frame-based	Mixed-Criticality	Single-core	Re-execution	×
[65]	Sporadic	Mixed-Criticality	Homogenous Multi-core	DMR, TMR, Replication	×
[35]	Periodic	Mixed-Criticality	Homogenous Multi-core	Reliability-aware	×
[36]	Periodic	Mixed-Criticality	Single-core	Re-execution	×
[3]	Sporadic	Mixed-Criticality	Homogenous Multi-core	Replication	×
[12]	DAG	Mixed-Criticality	Homogenous Multi-core	Standby-Sparing	×
[37]	Periodic, Sporadic	Mixed-Criticality	Homogenous Multi-core	Standby-Sparing	×
[71]	Periodic	Mixed-Criticality	Homogenous Multi-core	Checkpointing	×
[5]	Periodic	Hard real-time	Heterogeneous Multi-core	Primary/backup	✓
[13]	Periodic DAG	Hard real-time	Homogenous Multi-core	NMR	✓
[15]	Periodic	Hard real-time	Homogenous Multi-core	Primary/backup	✓
[73]	DAG	Hard real-time	Homogenous Multi-core	Re-execution	✓
[66][67][68][69][70]	Periodic	Hard real-time	Heterogeneous Multi-core	×	✓
[74]	Periodic	Mixed-criticality	Homogenous Multi-core	×	✓
ReTMiC	Periodic DAG	Mixed-Criticality	Homogenous Multi-core	Task Replication	✓

MCSs, there are some scheduling algorithms for DAG-based real-time systems. Kumar et al. in [72] have proposed two low-overhead heuristic algorithms called Global Slack Aware Quality-level Allocator (G-SLAQA) and Total Slack Aware Quality-level Allocator (T-SLAQA), which can produce satisfactorily efficient as well as fast solutions within a reasonable time. G-SLAQA, the baseline heuristic, is greedier and quicker than its counterpart T-SLAQA, whose performance is at least as efficient as G-SLAQA. The proposed schemes' efficiency has been extensively evaluated through simulation-based experiments using benchmark and randomly generated DTGs on heterogeneous multiprocessor platforms. The study in [75] focused toward the design of optimal as well as heuristic static scheduling techniques for periodic DAGs on heterogeneous multiprocessor platforms to minimize overall makespan. As it is clear, none of the mentioned previous work considers thermal management or fault-tolerant along with guaranteed QoS of LC tasks in the dependent task model in mixed-criticality systems.

B. FAULT-TOLERANCE SCHEDULING WITHOUT CONSIDERING PEAK-POWER OR TEMPERATURE

The study in [59] provides a comprehensive survey about applying different fault-tolerant techniques in real-time systems (including soft, hard, mixed-criticality systems) with considered goals and constraints of the mapping and scheduling policies. Some recent works that explore the fault-tolerant scheduling for MCSs without considering power/thermal management such as works presented in [11], [29], [30], [61], [62], [63], [64], and [65].

Few studies have considered power/energy, or temperature management in fault-tolerant mixed-criticality systems. The study in [35] has considered energy management in the mixed-criticality system with three criticality levels while the reliability of the system is kept at its original value. The study in [36] has reduced the energy consumption of a single-core MCS, which exploits re-execution as a fault-tolerant technique. The work in [3] has proposed a scheme to satisfy timeliness, energy management, fault tolerance, and guaranteed service level in multicore MCSs with sporadic task models. The approaches in [12], and [37] have focused on real-time constraints, energy management, reliability requirements, and QoS in dual-criticality MCSs in addition to exploiting standby-sparing to tolerate permanent faults. The energy consumption of a mixed-criticality system with an independent periodic task model is reduced in [71] when checkpointing is exploited as the fault-tolerant technique. As it is clear, none of the previous work considers thermal management in fault-tolerant MCSs in the dependent task model.

C. PEAK-POWER/TEMPERATURE-AWARE MAPPING AND SCHEDULING

Peak power reduction and thermal management have been widely studied from the performance aspect in [7]. Pagani et al. [7] introduced a new power budgeting technique for multicore systems with no timing constraint. This technique focuses on increasing performance while keeping the processor's temperature under a safe limit. The study in [5] proposed a power management method for heterogeneous multicore systems that reduces power consumption and tolerates both transient and permanent faults through primary/backup fault-tolerant techniques while considering core-level power constraint, real-time constraint, and aging effect. Some works try to reduce thermal and peak power while meeting the timing constraints [13], [15]. The reference [13] has proposed a new scheduling algorithm for hard real-time systems to reduce the chip-level peak power consumption and meet TDP. Chen et al. [16] proposed an approximation algorithm for reducing the peak temperature of multiprocessor systems with no heat transfer between the processors of real-time tasks without precedence constraints. The mentioned thermal-aware scheduling algorithms focus on partitioned scheduling of periodic real-time tasks or a set of job instances without periodicity.

None of these works have considered peak power and thermal management in mixed-criticality systems with dependent tasks. Few works, e.g. [31], [32], [33], and [34], cope with the power/energy management problem in MCSs, without considering reliability requirements. The study in [31] has minimized the dynamic average power consumption of single-cores through optimal DVFS in EDF-VD scheduling. The work in [32] has extended the work in [31] to multicores. The study in [33] has reduced static power consumption by applying the DPM technique in single-core MCSs. Volp et al. [34] have considered an energy budget for multicore MCSs, which considers the energy utilization of HC tasks and drops all LC tasks in overrun mode.

The research in [66], [67], [68], [69], and [70] present the heuristic strategies for energy and temperature-efficient scheduling of a set of hard real-time periodic tasks on a DVFS-enabled heterogeneous platform. Devaraj et al. in [73] have proposed a supervisory control-based fault-tolerant scheduling synthesis scheme for hard real-time tasks modeled as precedence-constrained task graphs, executing on homogeneous multicores. Further, the authors devise search strategies to obtain schedules that maximize reliability and minimize peak power consumption.

None of the mentioned previous work in this category considers both thermal management and fault-tolerant along with guaranteed QoS of LC tasks in the dependent task model in mixed-criticality systems.

Table 1 shows the summary of related works from the perspective of application, real-time and architecture models, following with applied fault tolerance technique and temperature management.

III. MODELS AND ASSUMPTIONS

This section presents the task, system, reliability, power consumption, and thermal models to formulate our proposed method.

A. SYSTEM AND APPLICATION MODEL

We consider a homogeneous multicore embedded system with M cores $\{C_1, C_2, \ldots, C_M\}$ that supports DVFS with a finite set of available voltage and frequency levels, i.e. $VF=\{vf_{min}, \ldots, vf_{max}\}$. In this paper, we consider directed acyclic graphs (DAGs) where n dependent tasks $\{\mathrm{T}_1, \mathrm{T}_2, \ldots, \mathrm{T}_n\}$ are executed and should be completed before the deadline D of the graph. A task graph is composed of nodes and edges where each node represents a task while the edges represent data dependencies between the tasks. Fig. 1a shows an example task graph with LC and HC nodes. Each task T_i has $\{\zeta_i, WC_i^{LO}, WC_i^{HI}, X_i, s_i, e_i, O_i, SU_i, PR_i, P_i, r_i\}$ parameters that will be determined in the offline phase:

- $\zeta_i \in \{\mathrm{LC}, \mathrm{HC}\}$: The criticality level of T_i.
- WC_i^{LO}: The designer-specified WCET for T_i.
- WC_i^{HI}: The CAs-specified WCET for T_i.
- X_i: The assigned core to T_i.
- s_i, e_i: The start time and completion time of T_i after mapping and scheduling.
- $O_i \in \{\mathrm{NO}, \mathrm{OV}\}$: The operational mode of T_i.
- SU_i, PR_i: List of successors and predecessors of T_i.
- P_i: Peak power consumption of T_i.
- r_i : The number of required replicas of T_i.

In the dual-criticality systems, HC tasks have two WCETs, one that is estimated by the system designer, which is called low-level WCET (WC^{LO}), and one that is more pessimistic and estimated by the certification authority (CA), which is called high-level WCET (WC^{HI}). LC tasks have only low-level WCETs. Therefore, if ζ_i =LC, WC_i^{HI} =WC_i^{LO}, otherwise $WC_i^{LO} < WC_i^{HI}$ [11], [38]. The system starts its operation in the normal mode (NO). Whenever an HC task exceeds its WC^{LO} without signaling completion, the system switches to the overrun mode (OV) where HC tasks will be executed based on WC^{HI}. Fig. 3 and Table 4 in Section IV-C show an example of a mixed-criticality DAG with its parameters. We will exploit this example to introduce our illustrative example in Section IV-C. Indeed, the illustrative example (see Section IV-C) will also reflect all mentioned parameters in detail.

B. POWER CONSUMPTION MODEL

The power model consists of static and dynamic components [3], [40], and [41]. The total power consumption of a core is:

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

where, C_{eff}, v_i, and f_i are the effective switching capacitance, operating voltage, and frequency of the core during the execution of task T_i, respectively. Intuitively, down-scaling the v-f levels will reduce power consumption. When DVFS is exploited, the task T_i is executed with a lower v-f level than $vf_{\max}$ which is selected from available v-f levels of the core. However, it will decrease the reliability of the tasks [42]. Note that in this paper, we apply the DVFS technique such that the reliability of each task with its corresponding replicas is not degraded and is kept at an acceptable level.

C. FAULT AND RELIABILITY MODEL

Since safety-critical applications are often controlled by mixed-criticality embedded systems, tolerating fault occurrence and reaching high reliability are important goals; i.e.,

TABLE 2. DO178B safety requierement.

ζ	A	B	C	D	E
PFH	$< 10^{-9}$	$< 10^{-7}$	$< 10^{-5}$	$< 10^{-3}$	-

faults must be detected, and appropriate recovery tasks must be successfully completed before the deadlines. In MCSs, each criticality level has an important property, which is known as the Probability of Failure per Hour (PFH). PFH represents the maximum probability of failure to which each task of that level must adapt. The avionics DO-178B standard defines five criticality levels from A with the highest, to E with the lowest criticality levels. Safety requirements of each criticality level are shown in Table 2 [59].

Faults can be categorized into transient and permanent faults. In this paper, we focus on tolerating transient faults because they occur more frequently than permanent faults with a ratio of 100:1 or higher [42]. Exponential distribution with an arrival rate λ is exploited to model transient faults [8]. Note that the average fault rate at the maximum voltage and frequency is denoted by λ_0. The fault rate at a scaled voltage $v_i = \rho_i v_{max} (\rho_{min} < \rho_i < \rho_{max} = 1)$ can be expressed as [8] and [42]:

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

where, d is a constant value in the range of 2 to 6 [43]. We assume that λ_0 equals 10^{-6} at the maximum available frequency [41]. Reliability is defined as the probability of the correct operation of the system according to its specifications in the time interval [0, t], with the assumption that it was functioning correctly at time 0 [43]. Assuming $f_i = \sigma_i f_{max} (\sigma_i$ is the scaling factor of frequency $(\sigma_{min} < \sigma_i < \sigma_{max} = 1))$ and $v_i = \rho_i v_{max}$, the reliability of the task T$_i$ executing at voltage v_i and frequency f_i in the worst-case scenario, when all HC tasks and corresponding replicas are executed with WC^{HI}, can be expressed as:

$$R_i^{HI}(\rho_i, \sigma_i) = e^{-\lambda(\rho_i)\frac{WC_i^{HI}}{\sigma_i}} \tag{3}$$

Therefore, the Probability of Failure (PoF) of the task T$_i$ is:

$$PoF_i^{HI}(\rho_i, \sigma_i) = 1 - R_i^{HI}(\rho_i, \sigma_i) \tag{4}$$

In the task replication fault-tolerant technique, multiple copies of each task are executed on multiple cores, which significantly increases the likelihood of completing at least one of them successfully (i.e., without encountering transient faults). Note that each replica task has the same successors and predecessors as the original task. The minimum number of required copies for each HC task is achieved as follows [3]:

$$PoF_i^{HI}(\rho_i, \sigma_i)^r \leq PoF_{target} \tag{5}$$

$$\left\lfloor \frac{log(PoF_{target})}{log(PoF_i^{HI}(\rho_i, \sigma_i))} \right\rfloor \leq r_i \tag{6}$$

where, PoF is the probability of failure of a task. The PoF_{target} is the target PoF that should be met based on safety standards. In the task replication technique, an acceptance test is applied at the end of each task's execution time. We consider that each core employs a low-cost, low-power, and high-accuracy fault detection hardware checker like Argus [44]. Therefore, as soon as the original task is completed successfully, the remaining parts of its corresponding replica(s) will be dropped to avoid further power consumption and heat generation. It should be noted that the space shuttle [45], X-38 Crew return vehicle [46], and Boeing 777 [47] are examples of real applications that use more than two replicas to satisfy the given reliability target based on safety standards.

D. THERMAL MODEL

We exploit the RC thermal network [48] to model the thermal behavior of the system with electrical circuit components. Our thermal model consists of N nodes, where $N \leq Z$. We assume our system consists of M cores, and Z blocks in the floorplan, such that Z-M is the number of blocks that correspond to other types of components, e.g., L2 caches and memory controllers. Transient temperature is modeled with a thermal capacitance for each node in the RC thermal network, and thermal nodes are interconnected by using thermal conductance. We denote the ambient temperature by θ^{Amb} and consider it to be constant. Therefore, no capacitance will be associated with θ^{Amb}. In this model, heat sources indicate the power consumption of the cores and other blocks on the chip. We model each thermal node's temperature as a function of its power consumption, the temperature of the neighboring nodes, and the ambient temperature. We build a system of N differential equations for an RC thermal network with N thermal nodes, which can be described as [7]:

$$\boldsymbol{A\theta}' + \boldsymbol{B\theta} = \boldsymbol{P} + \theta^{Amb}\boldsymbol{G} \tag{7}$$

The matrices $\mathbf{A}=[a_{i,j}]_{N\times N}$ and $\mathbf{B}=[b_{i,j}]_{N\times N}$ represent thermal capacitance and conductance values, respectively. In matrix $\mathbf{B}$, values are in [*watt*/*kelvin*] unit and include the thermal conductance values between vertical and lateral neighboring nodes. The column vector $\theta = [\theta_i]_{N\times 1}$ contains the temperature value of each node, and the column vector $\theta\prime = [\theta\prime_i]_{N\times 1}$ accounts for the first-order derivative of the temperature on each node with recording to time. The power consumption measurement of each node is contained in the column vector $\mathbf{P} = [p_i]_{N\times 1}$. We represent the thermal conductance between each node and the ambient by using the column vector $\mathbf{G} = [g_i]_{N\times 1}$. For each node i, we set the corresponding value in the column vector $\mathbf{G}$ to zero if the node is not related to the ambient temperature, e.g., the temperature of a core or an internal node. For calculating steady-state temperature, we set $\mathbf{A}$ to zero then we have [7]:

TABLE 3. Notations.

Symbol	Description
M	Set of homogenous cores in the system
VF	Set of voltage and frequency levels in the system, where $VF=\{vf_{min}, \ldots, vf_{max}\}$
n	Set of dependent tasks $\{T_1, T_2, \ldots, T_n\}$
D	Deadline of the whole graph
ζ_i	The criticality level of T_i, where $\zeta_i \in$ {LC, HC}
WC_i^{LO}	The designer-specified WCET for T_i
WC_i^{HI}	The CAs-specified WCET for T_i
X_i	The assigned core to T_i
s_i	The start time of T_i after mapping and scheduling
e_i	The completion time of T_i after mapping and scheduling
O_i	The operational mode of T_i, where $O_i \in$ {NO, OV}
SU_i	List of successors of T_i
PR_i	List of predecessors of T_i
P_i	Peak power consumption of T_i
r_i	The number of required replicas of T_i
$\lambda(\rho_i)$	The fault rate at a scaled voltage
ρ_i	The scaling factor of voltage
σ_i	The scaling factor of frequency
$R_i^{HI}(\rho_i, \sigma_i)$	The reliability of the task T_i executing at voltage v_i and frequency f_i, when all HC tasks and corresponding replicas are executed with WC^{HI}
$PoF_i^{HI}(\rho_i, \sigma_i)$	The probability of Failure (PoF) of the task T_i
θ^{Amb}	Ambient temperature
θ	steady-state temperature
$\theta^C_{Max}, \theta^C_{Min}$	The maximum and minimum chip temperatures at a specific time

$$\boldsymbol{\theta} = \boldsymbol{B}^{-1}(\boldsymbol{P} + \theta^{Amb}\boldsymbol{G}) \qquad (8)$$

Table 3 shows all exploited notations throughout the paper.

IV. PROPOSED METHOD

Fig. 2 shows the overall operational flow of the proposed ReTMiC (Reliability-Aware Thermal Management in Multicore Mixed- Criticality Embedded Systems) method. First, the scheduler receives software and hardware level parameters, as explained in Section III. As shown in IV-A, the proposed method consists of two offline and online phases. In the offline phase, first, the minimum number of the required replica(s) for each task in the maximum v-f level to satisfy the reliability target is computed, and the task graph is extended to include replica(s). Then, Algorithm 1 is called to determine the minimum v-f levels for each task such that the reliability target is satisfied. In the next step, all the normal and overrun parts of each HC task and corresponding replicas are mapped to the cores and scheduled in a way that meets timing and TDP constraints. Next, LC tasks are mapped and scheduled in a way that satisfies the QoS and TPD constraints. In the final extracted schedule at the offline phase, all LC, HC, and corresponding replica tasks are scheduled in a way that

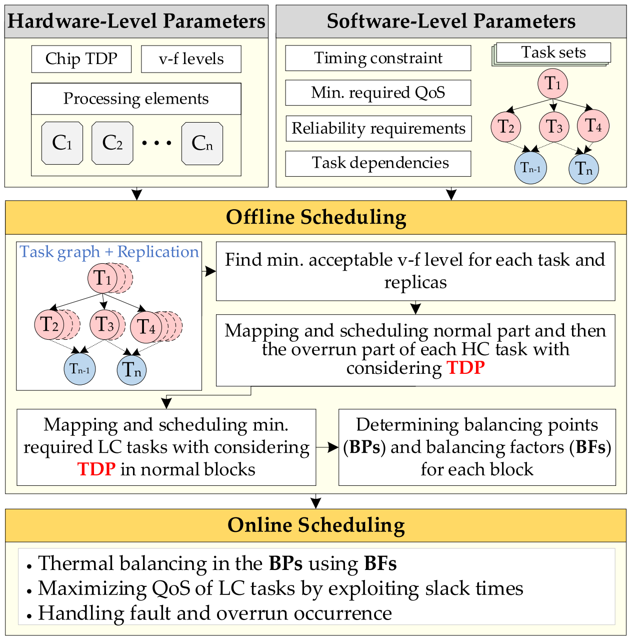

Hardware-Level Parameters
Chip TDP
v-f levels
Processing elements
C1
C2
Cn
Software-Level Parameters
Timing constraint
Min. required QoS
Reliability requirements
Task dependencies
Task sets
T1
T2
T3
T4
Tn-1
Tn
Offline Scheduling
Task graph + Replication
Find min. acceptable v-f level for each task and replicas
Mapping and scheduling normal part and then the overrun part of each HC task with considering TDP
Mapping and scheduling min. required LC tasks with considering TDP in normal blocks
Determining balancing points (BPs) and balancing factors (BFs) for each block
Online Scheduling
• Thermal balancing in the BPs using BFs
• Maximizing QoS of LC tasks by exploiting slack times
• Handling fault and overrun occurrence

FIGURE 2. The operational flow of our proposed ReTMiC scheme.

meets the deadline, reliability, TPD, and QoS constraints even in the worst-case fault or overrun occurrence scenarios. However, since the actual temperatures of tasks are determined at run-time, the temperature should be balanced at runtime. Hence, at design time, the scheduler should determine the temperature balancing points and balancing factors that will be exploited at run-time. To this end, according to the scheduling, the proposed method specifies Balancing Points (BPs) and Balancing Factors (BFs) (explained in detail in subsection IV-B1) to balance the overall temperature of the chip at run-time.

BPs are time points before starting each execution block, and at each BP the online algorithm checks the temperature of tasks inside each execution block for re-mapping. It should be noted that at each execution block, only one task will be executed on each core. Therefore, in this paper, not only the peak temperature is satisfied thorough considering the TDP threshold, but also the overall temperature will be balanced to avoid aging effects and increasing permanent fault rate. Indeed, by re-mapping the tasks at the beginning of each execution block (at each balancing point) before starting the execution of the task, the overall temperature will be balanced without violating the peak temperature constraint. The balancing points, the eligible tasks for execution in each block, and balancing factors are determined through our proposed methods in Section IV-B.

In the online phase, the scheduler balances the system temperature in different BPs by exploiting balance factors and task migration. Moreover, dynamic released slack times at runtime (due to replica cancellation, cancellation of overrun sections, or early completion of tasks) will be exploited to schedule more LC tasks to further improve the overall QoS (explained in subsection IV-B2).

A. PROBLEM DEFINITION

In the following, we define the problem of mixed-criticality task mapping and scheduling on a multicore embedded system when exploiting the task replication technique while meeting temperature constraints at design time and balancing the temperature at runtime.

1) OPTIMIZATION GOAL

In this paper, we have two optimization goals as follows:

- Minimize the $\Delta\theta$ (*spatial thermal variation*) of running tasks in the system chip:

$$Minimize : \Delta\theta = \theta^C_{Max} - \theta^C_{Min} \tag{9}$$

where, θ^C_{Max} and θ^C_{Min} are the maximum and minimum chip temperatures at a specific time, respectively.

- Maximize the total QoS of LC tasks:

$$Maximize\ QoS_{total} = \frac{\#\ Scheduled\ LC\ tasks}{total\ \#\ LC\ tasks} \tag{10}$$

2) RELIABILITY CONSTRAINT

The number of replicas (j) and v-f levels of each task (T$_i$) should be chosen to satisfy the task-level reliability target as follows:

$$R_{target} \leq \forall T_i, 1 - \prod_{j=1}^{r_i}(1 - R_{replica_j}) \tag{11}$$

3) TIMING CONSTRAINT

The completion time of each preemptive task should not exceed the task graph timing constraint:

$$\forall T_i, \quad e_i \leq D \tag{12}$$

4) CHIP POWER CONSTRAINT

The total power consumption of all underlying cores should be less than the chip TDP constraint at each time slot:

$$\forall T_i intimeslotk : \sum P_{i,k} < TDP \tag{13}$$

5) QoS CONSTRAINT

The QoS of the system in the offline phase should be bigger than or equal to QoS_{target}, which is the input of the system:

$$QoS_{target} \leq \frac{\#\ Scheduled\ LC\ tasks}{total\ \#\ LC\ tasks} \tag{14}$$

6) TASK GRAPH DEPENDENCY CONSTRAINTS

The completion time of the predecessor task (PR_j) should be smaller than the start time of its dependent task.

$$\forall T_i, T_j, e_i \leq s_j, \quad ifT_i \in PR_j \tag{15}$$

B. ALGORITHM DISCUSSION

Our proposed method consists of offline and online phases, which are explained in detail as follows.

1) OFFLINE PHASE

The offline phase consists of the following steps. First, the minimum number of required replicas and corresponding v-f level for each task is determined in Algorithm 1. Then HC tasks and their corresponding replicas and then LC tasks are assigned to cores and scheduled through the proposed method (Algorithm 2 describes the steps of scheduling in detail). Finally, the proposed method for selecting the set of eligible tasks that can be executed in each execution block is described in Algorithm 3. Finally, the balancing points (BP) and balancing factors (BF) that will be exploited at run-time will be determined at design time.

It is worth mentioning that the mixed-criticality systems should guarantee that all the constraints are met in the worst-case scenario in the design time because they are used in critical applications. Therefore, in mixed-criticality applications, all timing requirements should be guaranteed offline, before putting the system in operation [56]. If even one HC task cannot be completed within its timing constraints, the system must notify this fact in advance. Therefore, the task set should be known a priori which is one of the most important characteristics of real-time systems (that is known as predictability) [58]. Hence, in the offline phase, we have considered the worst-case scenario of running tasks on the cores. Therefore, we have pushed most of the computation overheads to the offline phase. Hence, in our proposed method, in the final extracted schedule at the offline phase, all LC, HC, and corresponding replica tasks are scheduled in a way that meets the deadline, reliability, TPD, and QoS constraints even in the worst-case fault or overrun occurrence scenarios. However, since the actual temperatures of tasks are determined at run-time, the temperature can be balanced at runtime. Hence, at design time, the scheduler should determine the temperature balancing points (BP) and balancing factors (BF) which will be exploited at run-time. To this end, according to the offline scheduling, the proposed method specifies Balancing Points (BPs) and Balancing Factors (BFs) (explained in detail in subsection IV-B1) to balance the overall temperature of the chip at run-time.

2) DETERMINING THE MINIMUM NUMBER OF REPLICAS

Algorithm 1 demonstrates the pseudo-code for finding the minimum number of the replica(s) and v-f levels for each task and its corresponding replicas. At first, the algorithm uses Eq. 5 to find the minimum number of replicas for each task at the maximum v-f level (line 1). In lines 2-4, the algorithm initializes the maximum available v-f levels for all generated replicas by the VF.getMax function. In lines 5-14, the algorithm finds the minimum acceptable v-f level for each task with its corresponding replicas (each task has j replicas) that still meet the reliability target with the same number of replicas. Indeed, each task and its corresponding replicas are selected consecutively through the **For** loop, and in each iteration, only the v-f level of one of the versions of the task is reduced (line 7). Then, again through Eq. 11, the algorithm calculates the reliability of the selected task (through calculateReliability function). If the calculated reliability is greater than or equal to the reliability target of the task (T_Q.reliability), and the temporary variable j is smaller than the number of task replicas (T_Q.R()), the algorithm increases j and continues the loop to check another replica for v-f level reduction (lines 8-10). Otherwise, if the temporary variable j is not smaller than the number of task replicas, the algorithm sets j to one, to start the process of decreasing the v-f level of the replica(s) by starting the first version (lines 11-12). In opposite to line 8, if the calculated reliability with a new v-f level does not satisfy the task reliability target, it shows that the algorithm cannot reduce the v-f level anymore. In this case, the previous computed v-f level, which satisfies the reliability target, is selected in line 15 through the increaseVF() function. In line 16, the set of tasks and their corresponding replicas with their computed v-f levels are returned. In another scenario, when the algorithm reaches the lowest v-f level of the system for all replicas, the loop is finished and the algorithm returns the computed v-f levels in line 19. The time complexity of Algorithm 1 is O(V×T), where V and T are the numbers of v-f levels and tasks, respectively. $T^i_Q T^i_Q T^i_Q$

Algorithm 1 Voltage and Frequency Calculation for Tasks and Replicas

Inputs: T_Q**:** Task and corresponding replicas with v-f level, *VF*:Available processor v-f levels.
Output: The number of task's replicas, and v-f level for each replica.

```
Function TaskReplica($T_Q$, $VF$)
1.  $T_Q$.R ← Calculate #replicas for the task in max. v-f level using
      Eq. 5;
2.  for $i = 0$ to $T_Q$.R() do
3.     $T^i_Q$.setVF←$VF$.getMax();   //Set Max. v-f level for each
       replica
4. end for
5. $j$ ←1;   // Number of replicas for each task
6.  for $i = 0$ to $T_Q$.R()×($VF$.Size()-1) do
7.    $T^i_Q$.decreaseVF();
8.    if $T_Q$.calculateReliability() ≥ $T_Q$.reliability then   //w.r.t.
       Eq. 11
9.       if $j$ < $T_Q$.R() then
10.          $j$ ←$j$+1;
11.        else
12.          $j$ ←1;
13.        end if
14.     else
15.        $T^i_Q$.increaseVF();
16.        return $T_Q$;
17.     end if
18.  end for
19.  return $T_Q$;
End function
```

3) TASKS' MAPPING AND SCHEDULING

Algorithm 2 shows the pseudo-code of the task mapping and scheduling mechanisms for HC and LC tasks in the offline phase. It is worth mentioning that our proposed method can work based on any mapping policies in the offline phase such as Worst-Fit Decreasing (WFD), First-Fit Decreasing (FFD), and Best-Fit Decreasing (BFD) bin packing. Moreover, in the online phase, tasks will be re-mapped based on tasks' thermal behavior. In this algorithm, we use the first-fit (FF) mapping policy. In the First-Fit Decreasing bin packing, cores are sorted in decreasing order by utilization, then the selected task is allocated into the core with the lowest capacity available (largest utilization), in which it can be feasibly allocated. In Algorithm 2, at first, all HC tasks and their corresponding replicas in the graph are assigned to τ in line 1. Then, in line 2, all tasks in τ are sorted based on their WC^{LO} in decreasing order. In line 3, the balancing point (*BP*) array is initialized to keep the balancing points time in the scheduling. In lines 4-17, all HC tasks and their corresponding replicas are scheduled such that timing, reliability, and power constraints are met. In the **While** loop, first, Algorithm 3 is called every time to return the tasks that can be scheduled in the present block through the SelectTask(G, τ,C,TDP) function. The maximum number of eligible tasks in each block (ST.Size in Algorithm 3) is equal to the number of cores (C.Size), and each task is assigned to a separate core. In our proposed method, there are two kinds of blocks, known as *normalBlocks* and *overrunBlocks*, where *normalBlocks* include all tasks based on their WC^{LO}, while *overrunBlocks* consider the overrun parts of HC tasks (WC^{HI}-WC^{LO}). In line 6, the completion time of the last task (including its overrun part) that is scheduled until the current time, among all cores, is determined and assigned to the *SchedulingFinishTime* parameter. The start time of each new block (*newBlockStartTime* parameter) will be equal to *schedulingFinishTime.* Next, in lines 7-10, the algorithm schedules normal parts of all HC tasks one by one in the current block such that all tasks are shifted to the end of the block, i.e., *newBlockStartTime*+*blockLength*-*blockTasks*[i].WC^{LO} (line 8). In line 9, after scheduling each replica of each task, the counter of replicas is increased (ScheduledReplica()). After this step, in lines 11-13, the overrun parts of the selected tasks are scheduled at the end of their normal parts on the same core.

It should be noted that tasks in the overrun blocks will be scheduled from the beginning of the block to the end time while tasks in the normal blocks are shifted to the end of the block. In lines 14-16, the deadline violation is checked. In line 14, the number of *BPs* is multiplied by the worst-case time overhead of the online mechanism (OH^{TB}) and its overhead is subtracted from the deadline (D). If the deadline (considering the online overhead) is violated, the algorithm returns infeasible. Otherwise, it continues the procedure by adding the *schedulingFinishTime* as a new balancing point to the *BP* array. After scheduling all HC tasks and corresponding replicas, in lines 18-49, the algorithm tries to add as many LC tasks as possible into the scheduling. For this purpose,

Algorithm 2 The Task Scheduling Mechanism in the Offline Phase

Inputs: *G*: Task graph, *D*: Deadline, *C*: Set of cores, *TDP*: Chip TDP, OH^{TB}: Time overhead of online thermal balancing mechanism.
Output: *S* : The task scheduling, *BPs:* Balancing points, *BFs*: balancing factors for each core of blocks.

```
Function OfflineScheduling($G$, $D$,$C$,$OH^{TB}$)
1.  $\tau \leftarrow G$.getAllHCTasks; //HC tasks and their replicas
2.  $\tau$.sort();          // sort tasks in decreasing order w.r.t $WC^{LO}$
3.  $BP \leftarrow$ null; //initialize an array for saving the time of $BPs$
4.  while all tasks in $\tau$ are not scheduled do
5.      $blockTasks \leftarrow$ SelectTask($G$, $\tau$,$C$,$TDP$); //Call Algorithm 3
6.      $newBlockStartTime \leftarrow S$.schedulingFinishTime();
7.      for $i = 1$ to $blockTasks$. Size() do
8.          $S$. Schedule ($blockTasks[i]$, ($newBlockStartTime+blockLength$- $blockTasks[i].WC^{LO}$), $C[i]$);
9.          $blockTasks$ [$i$].ScheduledReplica()++;
10.     end for
11.     foreach $task$ in $blockTasks$ do
12.         $S$. ScheduleOverrunPart($task$);
13.     end for
14.     if $S$.SchedulingFinishTime() > $D$- $\#BPs \times OH^{TB}$ then
15.         return infeasible;
16.     $BP$. Add($S$.schedulingFinishTime());
17. end while
18. foreach $normalBlock$ in $S$ do
19.     if $normalBlock$ does not have a free core then
20.         continue;
21.     end if
22.     foreach unscheduled $LC$ task in $G$ do
23.         $k \leftarrow$ FinishTimeOfPredecessors($LC$);
24.         if $block$.startTime $\geq k$ and $blockLength \geq LC.WC^{LO}$ then
25.             if blockTasksPeakPower+$LC$.peakPower()<TDP then
26.                 $S$.Schedule($LC$, ($blockStartTime+blockLength$- $LC.WC^{LO}$), firstFreeCore);
27.                 $LC$.Scheduled()++;
28.                 if the block does not have a free core then
29.                     break;
30.                 end if
31.             end if
32.     end if
33.     end for
34. end for
35.  $\tau \leftarrow G$.getAllUnscheduledLCTasks;
36   $\tau$.sort();          // sort tasks in decreasing order w.r.t $WC^{LO}$
37.  while all tasks in $\tau$ are not scheduled do
38.     $blockTasks \leftarrow$ SelectTask($G$, $\tau$,$TDP$); //Call Algorithm 3
39.     $newBlockStartTime \leftarrow S$.schedulingFinishTime();
40.     for $i = 1$ to $blockTasks$. Size() do
41.         $S$. Schedule($blockTasks[i]$, ($newBlockStartTime+blockLength$- $blockTasks[i].WC^{LO}$), $C[i]$;
42.         $blockTasks$ [$i$].Scheduled()++;
43.     end for
44      if $S$.schedulingFinishTime() > $D$- $\#BPs \times OH^{TB}$ then
45.         $S$.removeAllLCTasks($D$-($\#BPs \times OH^{TB}$));
46.         Break;
47.     end if
48.     $BP$. Add($S$.schedulingFinishTime());
49. end while
50.  if $S$.QoS < $QoS_{target}$ then
51.     return infeasible;
52. end if
53. foreach $task$of each $blocks$ in $S$ do
54.     $BFs.Add$(compute $BFs$ for $task$ from Eq. 16);
55. end for
56.  return ($S$, $BPs$, $BFs$);
End function
```

first, it tries to add LC tasks to existing normal blocks. In line 19, it checks whether the selected block has a free core. If there is a free core, it checks the unscheduled LC tasks for adding to the selected block (lines 22-33). If the conditions of adding a task to a block (dependency (k is equal to the finishing time of all predecessors of LC tasks), task length (WC^{LO}), and TDP constraints) are met in lines 24-25, the algorithm schedules the selected LC task in the block through S.Schedule(LC, ($blockStartTime+blockLength$- $LC.WC^{LO}$), firstFreeCore) function (line 26). Otherwise, the next LC task will be checked for scheduling. If, after adding an LC task to a block, there is a remaining free core in the selected block, the algorithm tries to add another LC task. Otherwise, it breaks the loop and continues the procedure with the next normal block (lines 28-29). After checking all present normal blocks, the algorithm should use the remaining time to the deadline to make new normal blocks and schedule more LC tasks. For this purpose, the algorithm uses the same mechanism that has been presented for scheduling HC tasks (lines 35-49). In line 44, it checks the deadline violation for LC tasks. If any task violates the deadline, the algorithm in line 45 removes all LC tasks with a completion time bigger than the deadline considering the online overhead through the S.removeAllLCTasks(D-($\#BPs \times OH^{TB}$)) function. Then, the QoS of the scheduling is compared to the target QoS. If the QoS_{target} is not satisfied, the algorithm returns infeasible, because the QoS is the constraint of the system and at design time a minimum acceptable level of QoS should be guaranteed. Finally, the algorithm starts to calculate the balancing factors (BFs). At runtime, the online manager will exploit *BFs* for balancing the temperature of the processing cores (lines 53-55). In the end, it returns the scheduling (S), *BPs,* and *BFs*. The purpose of determining *BPs* is to exploit the re-mapping technique instead of task migration during the execution of any task. Because in task migration a task will be preempted in the middle of its execution on a core and continue its execution on another core, which has unpredictable time overhead. However, in the re-mapping technique, the mapping of tasks may be changed at the beginning of each block before the task is started to execute. The time complexity of Algorithm 2 is O($T \times C$), where T and C are the number of tasks and cores.

4) DETERMINING ELIGIBLE TASKS FOR EXECUTION IN A BLOCK

Algorithm 3 shows the pseudo-code of selecting tasks that can be executed in one block. In the beginning, the *ST* array (The set of selected tasks that can run in one block) is initialized to *null* (line 1). This array is used to save the selected tasks on each block. Then, the variable *TPP* is set to *zero*. *TPP* keeps the sum of the peak power consumption of selected tasks at each block. In line 3, the maximum number of tasks in each block (*ST*.Size) is equal to the number of cores (*C*.Size). In line 4, the temporary variable (i.e., flag) is initialized to *false*, which is used to identify the situation when there are no tasks for selection. In lines 5-13, the algorithm iterates among tasks to find a proper task that can be added to *ST*. In this loop there are two conditions: 1) the longer tasks are selected as soon as possible, and 2) the replicas of each task are executed

Algorithm 3 Selecting Tasks That Can be Run in Each Block

Inputs: *G*: Task graph (including all information related to each task), τ: Selected tasks from Algorithm 2, *C*: Set of cores, *TDP*: Chip TDP.
Output: *ST*: Selected tasks that can run in one block.

```
Function SelectTask(G,τ,C,TDP)
1.   ST ← null;
2.   TPP ← 0; // Sum of selected tasks' peak power
3.   while ST.Size() ≠ C.Size() do
4.     flag ← false;
5.     for i = 1 to τ.Size() do
6.       if τ[i] predecessors are not scheduled then continue;
7.       if τ[i].ScheduledReplica()+ST.NumberOf(τ[i])+1
           > τ[i].R()
–            then continue;
8.       if τ[i].peakPower()+TPP ≥TDP then continue;
9.       ST.Add(τ[i]);
10.      TPP ← TPP+τ[i].peakPower();
11.      flag ← true;
12.      if ST.Size() = C.Size() Break;
13.    end for
14.    if flag = false then Break;
15.  end while
16.  return ST;
End function
```

in a different block as much as possible. By exploiting these conditions, replicas can be immediately dropped after the successful completion of the original task. Next, the algorithm checks whether predecessors of the current task are scheduled in the past blocks. In line 7, if the number of replicas for the selected task is equal to the required number of replicas for the corresponding task, the algorithm goes on with the next task (τ[i].R() shows the number of replicas for each task). In line 8, the new peak power consumption of tasks with adding a new task to this block is checked. If the summation of this task's peak power with variable *TPP* violates the chip TDP, it means that this task cannot be scheduled in the current block. Therefore, the algorithm goes on with the next task. If the algorithm passes lines 6-8, it shows that the selected task can be executed in the block. In line 9, the selected task is added to the *ST* array, and in line 10, the algorithm updates the *TPP* array. In line 11, the temporary variable becomes *true* to show that the algorithm selects at least one task in one complete iteration of the **For** loop. After ending one complete iteration of the **For** loop, the temporary variable in line 14 is checked. If it was *false,* it means that there is no other task that can be executed in the current block, and the algorithm breaks the **While** loop, and in line 16, it returns the variable *ST* that keeps the selected tasks. The time complexity of Algorithms 3 is O($T \times C$), where T and C are the number of tasks and cores. $\geq$

5) ONLINE PHASE

At runtime, we exploit a re-mapping technique based on the value of *BFs* that are calculated in the offline phase (Algorithm 2) and the temperature of the *BPs* to balance the temperature of the processing cores. Indeed, *BF* is an auxiliary criterion for approximating temperature behavior on each core. The following proposed equation computes the *BF* for each core (C_i):

Algorithm 4 Online Thermal Balancing

Inputs: *S* : The task scheduling, *C*: Set of processor cores, *BP:* Balancing points, *BFs*: Balancing factors of the blocks, $\theta(c_i)$: Current core's temperature
Output: Tasks re-mapping.

```
Function ThermalBalancing(S,C,BP,BFs,θ)
1.  if S. Time = BP then    // checking for reaching the balancing
      point
2.    for i = 1 to C. Size do
3.      Selected ← Select core in the block with biggest BF in
        BFs;
4.      c^Cool ← Select core with smallest θ(c_i) in C;
5.      S.Map(Selected in c^Cool);
6.      C.Remove(c^Cool);
7.      BFs.Remove(Selected);
8.  end for
End function
```

$$BF_i = \left(1 - \frac{slack\ time}{block\ length}\right)\theta_i^{\infty} + \theta^{Amb}$$

In the above equation, *slack time* is the available slack time in the normal block, θ_i^{∞} is the steady-state temperature of the corresponding task, and θ^{Amb} indicates the ambient temperature. In this equation, the coefficient $\left(1 - (slack\ time \big/ block\ length)\right)$ is used to indicate the effect of slack times (that may exist in scheduling before executing the tasks of each block on the cores) on the final temperature of each task of the block. It is worth mentioning that to extract the θ_i^{∞} for each task, similar to [49], all tasks are executed, one at a time at the selected frequency on a single core of the processor in the offline phase.

Algorithm 4 shows the pseudo-code of the online thermal balancing and task re-mapping mechanism. First, in line 1, the current time of scheduling is checked. If the scheduling reaches a balancing point, the online manager in each iteration selects the core in the block with the biggest *BF* (line 3) and maps its corresponding task to the core with the lowest current temperature (C^{cool}). In lines 4-5, the mapping is updated based on the mentioned changes. Our proposed algorithm decides about re-mapping at the beginning of each block; i.e., before the start of execution of the tasks in the block. Therefore, there are no currently running tasks and no task will be terminated in the middle of its execution. Moreover, the overhead of re-mapping is considered a part of the worst-case execution time (WCET) of the task with the longest execution time in each partition.

The time complexity of Algorithm 4 is O($C \times \log C$), where C is the number of cores. It is worth mentioning that based on our observation in the worst-case scenario on a 36-core system the time overhead of the online thermal balancing mechanism is 43μs for each BP.

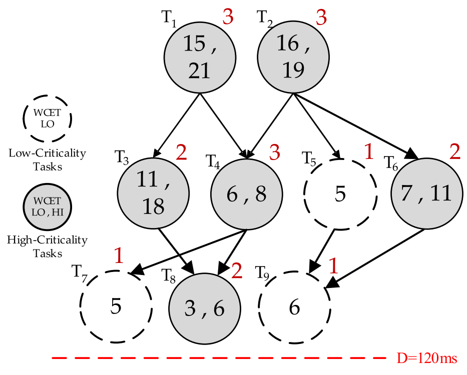


FIGURE 3. An Illustrative Example.

TABLE 4. The illustrative example task set details.

Task	ζ	v-f level	WC^{LO}	WC^{HI}
$T_{1,1}$	HC	[1,153v, 1.7GHz]	18	25
$T_{1,2}$	HC	[1,153v, 1.7GHz]	18	25
$T_{1,3}$	HC	[1,318v, 2.0GHz]	15	21
$T_{2,1}$	HC	[1,153v, 1.7GHz]	19	23
$T_{2,2}$	HC	[1,153v, 1.7GHz]	19	23
$T_{2,3}$	HC	[1,318v, 2.0GHz]	16	19
$T_{3,1}$	HC	[1,318v, 2.0GHz]	11	18
$T_{3,2}$	HC	[1,318v, 2.0GHz]	11	18
$T_{4,1}$	HC	[1,153v, 1.7GHz]	7	9
$T_{4,2}$	HC	[1,318v, 2.0GHz]	6	8
$T_{4,3}$	HC	[1,318v, 2.0GHz]	6	8
T_5	LC	[0,973v, 1.0GHz]	10	-
$T_{6,1}$	HC	[1,048v, 1.3GHz]	10	16
$T_{6,2}$	HC	[1,048v, 1.3GHz]	10	16
T_7	LC	[0,973v, 1.0GHz]	10	-
$T_{8,1}$	HC	[1,318v, 2.0GHz]	3	6
$T_{8,2}$	HC	[1,318v, 2.0GHz]	3	6
T_9	LC	[0,973v, 1.0GHz]	12	-

In the online phase, dynamic slack times that are released due to replica or overrun cancelation, or early completion of tasks are used to improve the QoS. In runtime, the scheduler finds the longest LC task (whose predecessors have been completely executed) that can fit in the released slack times. Then, TDP constraint of the system including the execution of the selected LC task is checked. If all system constraints are met, the selected LC task will be scheduled. Otherwise, the online scheduler tries to find another LC task that can fit in the released slack time.

It is worth mentioning that since the mapping of tasks to cores and extracting balancing points and balancing factors are done in the offline phase, the proposed method can be scalable to a higher number of DAGs or heavy DAGs with the higher number of tasks, and higher number of cores.

C. AN ILLUSTRATIVE EXAMPLE

In this subsection, we use an example to illustrate how our ReTMiC method works. In this example, we consider a task graph with six HC and three LC tasks, as presented in Fig. 3, whose detailed parameters are listed in Table 4. These tasks share a common deadline, $D = 120$ms. We consider a quad-core processor with a TDP of 3W. The red number above each node of the task graph shows the required number of replicas for each task (For example, task T_1 has three replicas which are represented with $T_{1,1}$, $T_{1,2}$, and $T_{1,3}$, respectively). According to Algorithm 1, the v-f levels for the original task and all replicas are determined. The difference in the height of boxes, that show the tasks, reflects their different v-f levels. After selecting the appropriate v-f level, based on Algorithm 2 and Algorithm 3, the scheduler starts selecting eligible tasks that can be executed in one block. For simplicity of presentation of task selection for executing in one block in Algorithm 3, we have extracted the tasks queue in each iteration of Algorithm 3, as shown in Fig. 4. The tasks in each iteration are sorted based on WC^{LO} and the scheduler tries not to schedule more than one replica of each task in one block. Selected tasks in each iteration of Algorithm 3 are shown with a green box in Fig. 4. In this example, T_1 and T_2 can be selected first because they do not have any predecessors and are not dependent on each other. Based on the *Longest Task First* (LTF) policy and tasks peak power, two replicas of T_2 and one replica of T_1 are selected (Fig. 4a) and the other tasks in this queue cannot be selected for scheduling because of TDP violation. It should be noted that the second number in the subscript of each task's name shows the replica number. In each iteration of making a normal block, ReTMiC makes a block w.r.t the length of the longest task in that block and all remaining tasks in the block are shifted to the end of the block. In the first block, $T_{2,1}$ is selected since it is the largest task, and the length of the block is chosen to be equal to its WC^{LO}(19ms). $T_{2,1}$ is scheduled between 0 to 19ms on C1, and $T_{1,1}$ is scheduled between 1ms to 19ms on C2 (because it must be shifted to the end of the block) and the second replica of T_2 ($T_{2,2}$) is scheduled between 0 to 19ms on C3 (Fig. 5a). *It should be noted that our proposed method can work based on any mapping algorithm, i.e., it is regardless of a specific mapping algorithm. Therefore, in this example, tasks are mapped with the First Fit bin-packing policy in each block.* Next, the overrun parts of the tasks which are scheduled in the first normal block are scheduled in the time slots between 19ms and 26ms. The overrun part of each task starts from the beginning of each overrun block on the same core of the normal part in order to avoid context switching between the normal and overrun parts of each task. We should mention that the length of each overrun block is equal to the biggest overrun part ($WC^{HI} - WC^{LO}$) of tasks in that block. Now, the end time of the current block (including the normal and overrun parts of the tasks), which is equal to 26ms, is added to the Balancing Points (BPs) of the system (Fig. 5b). In the next step, between the remaining replicas of T_1 and T_2(because of dependency constraint, other tasks in the graph cannot be a candidate to select), $T_{1,2}$ and $T_{2,3}$ are selected and because of TDP violation, $T_{1,3}$ cannot be selected (Fig. 4b). Same as before, first, the normal parts of each replica must be scheduled. Since $T_{1,2}$ has the longer WC^{LO}, the length of the block equals to its WC^{LO} and it is scheduled between 26ms to 44ms on C1. In the same way, $T_{2,3}$ is scheduled between 28ms to 44ms on C2 at the end of the block. Next, the overrun parts of the tasks in this block are scheduled at the end of

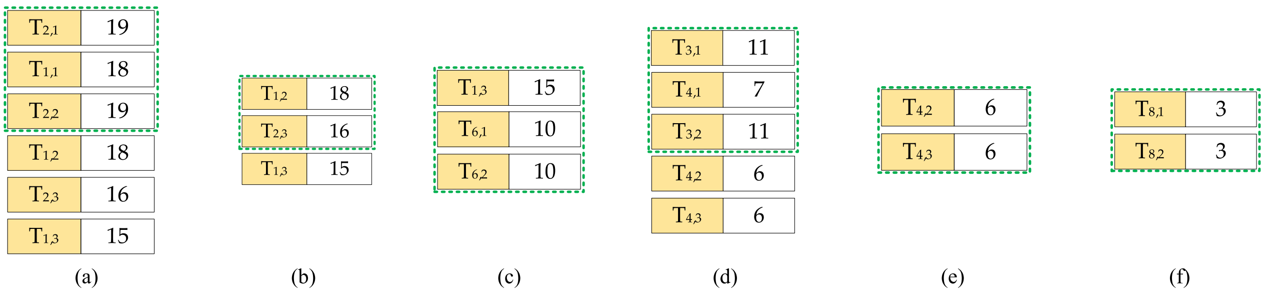


FIGURE 4. Detail of selecting tasks in Algorithm 3 and its tasks queue in each iteration.

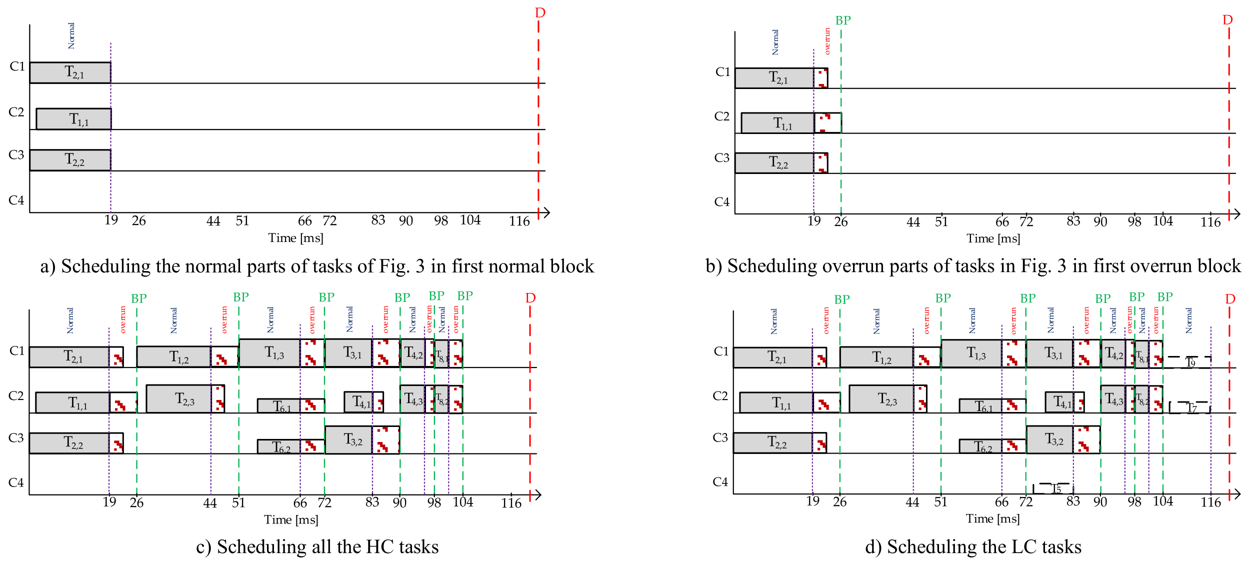

a) Scheduling the normal parts of tasks of Fig. 3 in first normal block
b) Scheduling overrun parts of tasks in Fig. 3 in first overrun block
c) Scheduling all the HC tasks
d) Scheduling the LC tasks
Time [ms]

FIGURE 5. An Illustrative example of scheduling Fig. 3 based on proposed ReTMiC method in a quad-core platform at design time.

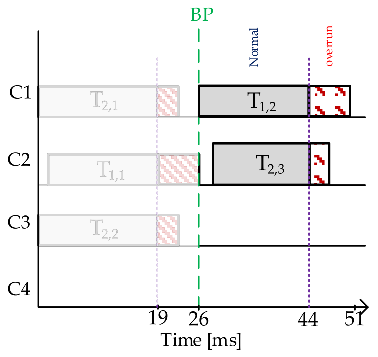


FIGURE 6. An illustrative example of calculating BF for a block.

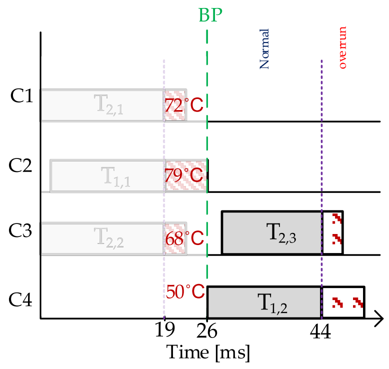


FIGURE 7. An illustrative example of the runtime function of ReTMiCmethod.

their normal parts between 44ms to 51ms. Now, the BP of this block is determined (which is equal to 51ms). In the next block, since all replicas of T_2 in the previous blocks have ended, the replicas of T_6 can be the candidates to select (Fig. 4c). The third copy of T_1 ($T_{1,3}$) and two copies of T_6 are ($T_{6,1}$, $T_{6,2}$) selected to schedule in the next block. $T_{1,3}$ is scheduled in the time slot 51ms to 72ms on C1 and two copies of T_6 are scheduled in the time slot 56ms to 72ms on C2 and C3, respectively. In the next step, the overrun block is created and the overrun parts of the tasks are scheduled between 66ms to 72ms immediately after their normal parts on the same core. Now, the new BP is added to BPs (72ms). Next, with the completion of T_1 in the previous blocks, T_3 can be scheduled in the current block. With building the queue for the tasks that can execute in the current block, two copies of T_3 and one copy of T_4 are selected for the next block (Fig. 4d). Of course, we can schedule the second copy of T_4 on C4 in this block, but scheduling this task in this block can result in the chip TDP violation. Based on Algorithm 2, $T_{3,1}$ is scheduled from 72ms to 83ms on C1. Then, $T_{4,1}$ is scheduled between 76ms to 83ms on C2 and $T_{3,2}$ is scheduled in the time slot 72ms to 83ms on C3. Next, the overrun parts of each task are scheduled at the end of their normal parts. Now, the end time of the current scheduling (90ms) is added as a new BP to BPs array. Therefore, the two remaining replicas of T_4 are selected for the new block (Fig. 4e). $T_{4,2}$ and $T_{4,3}$ are scheduled in the time slot 90ms to 96ms on C1 and C2, respectively. Next, the overrun parts of these tasks are scheduled between 96 to 98ms on C1 and C2. After that, the BPs are updated, and a new BP (98ms) is added to BPs. Next, the two replicas of T_8 ($T_{8,1}$, $T_{8,2}$) are selected for the new block (Fig. 4f). These tasks are scheduled in the time slot 98ms to 101ms on C1 and C2, respectively. Then, the overrun parts of these tasks are added to scheduling between 101 to 104ms. Next, the BP for this block (104ms) is added to BPs. Now, the scheduling procedure of HC tasks is ended (Fig. 5c).

In the following, the scheduler tries to add LC tasks to the scheduling as much as possible. First, it tries to add LC tasks to current normal blocks using Algorithm 2, lines 18-34. It should be noted that in our proposed method, tasks are not preempted. Hence, each LC task must schedule in a normal block that has enough time length to fit the entire task. Because of the dependency constraint of the task-graph, there is no LC task that can be scheduled in blocks of the system in the time slot 0 to 19ms and 26ms to 44ms. By adding LC task (T_5) to the next normal block (51 to 66ms), the chip TDP will be violated. Therefore, the scheduler checks the next normal block (72ms to 83ms) for adding T_5. In this block, T_5 can be scheduled because the block has enough time length, and no TDP violation is happening. After that, the scheduler checks the next block (90ms to 96ms), and since the WC^{LO} of remaining tasks (T_7, T_9) are longer than the block length, none of them can be scheduled in this block. In the next block, 98ms to 101ms, there is the same situation as the previous block, and we cannot add any task to this block. Now, for scheduling the remaining tasks, the scheduler makes a new block to schedule these tasks using Algorithm 2, lines 35-49. It schedules T_9 in the time slot 104ms to 116ms on C1, and T_7 in the time slot 106ms to 116ms on C2 (Fig. 5d). Indeed, Fig. 5d shows the final schedule where all HC tasks and their corresponding replicas and LC tasks are scheduled while, timing, reliability, and temperature constraints are met.

1) AN ILLUSTRATIVE EXAMPLE OF CALCULATING BALANCING FACTORS (BFS)

Next, BFs for each block must be calculated. In the following, the procedures for calculating BF for the second block of the illustrative example are shown. We assume that the steady temperatures for $T_{1,2}$ and $T_{2,3}$ are equal to 82°C and 89°C, respectively, and the ambient temperature for all cores equals 45°C. According to offline scheduling, the second block is between 26ms and 51ms (Fig. 6). Now, BFs are calculated for each core using Eq. 18. In core C1, there is no slack time. Therefore, Eq. 18 is simplified to $\theta_i^{\infty} + \theta^{\text{Amb}}$ and equals to 127. For core C2, the slack time is 2ms, and block length equals to 25ms. Then, BF is calculated as $BF_2 = \left(1 - (2/25)\right) \times 89 + 45$ and it equals 126.8. In core C3 and C4, the slack time equals block length; thus, their BFs are equal to 45.

2) AN ILLUSTRATIVE EXAMPLE OF THE RUNTIME FUNCTION OF RETMIC METHOD

For more clarification on the online phase of the proposed ReTMiC method, we show how our online re-mapping technique applies to the second block of the offline scheduling with reaching the first BP. As shown in Fig. 7, we assume that the temperatures of the cores in the BP are equal to 72°C, 79°C, 68°C, and 50°C, respectively. According to Algorithm 4, the task with the highest BF is assigned to the core with the lowest temperature. As a result, the designated task of C1 with the highest BF is re-mapped to the core with the lowest temperature (C4), and the assigned task of C2 with the second-highest temperature is re-mapped to the core with the second-lowest current temperature (C3).

V. EVALUATIONS

In this section, we present our system setup and the results of our evaluations.

A. EXPERIMENTAL SETUP

We have considered a homogeneous multicore platform that consists of out-of-order ARM Cortex-A15 cores. We have employed the gem5 simulator [41] and McPAT [42] to obtain the power profile and area details. We used the open-source tools introduced in [30] for generating real mixed-criticality DAGs (MC-DAG), i.e., the topology of DAGs (including the nodes and communication between nodes) is generated with the tool in [38], then each node of the graph (task) has characteristics associated with the selected task of real-world workload PARSEC benchmark suite [43]. Real DAG-based benchmarks like Stream-IT [54] can also be exploited in evaluations. Results are exported based on 100 generated MC-DAGs with different random configurations where each MC-DAG consists of up to 100 tasks. Moreover, we used the HotSpot simulator [48] and MatEx [44] to model steady temperature and transient temperature, respectively. Fig. 8 shows the overview of our experimental framework flow, and the simulation configurations are summarized in Table 5. Our ReTMiC method is compared with state-of-the-art techniques of hard real-time and mixed-criticality systems. In our analyses, the comparative state-of-the-art techniques are:

- Medina [38]: This technique schedules DAG-based mixed-criticality tasks and guarantees the schedulability of HC tasks in both normal and overrun modes. However, it drops all the LC tasks in the overrun scenario and does not consider any fault-tolerant technique.
- TP3M [13]: This work has proposed a peak-power-aware mapping and scheduling method to meet TDP

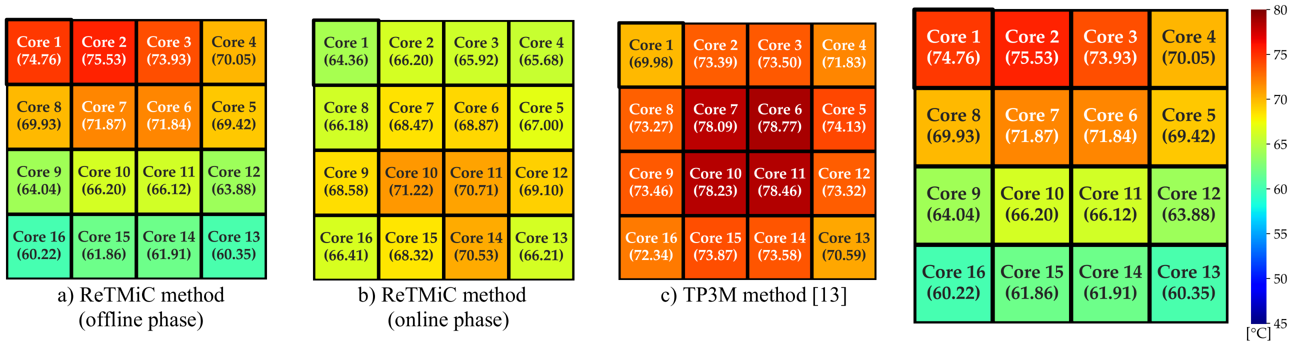


FIGURE 8. An experimental example of the effectiveness of our ReTMiC method.

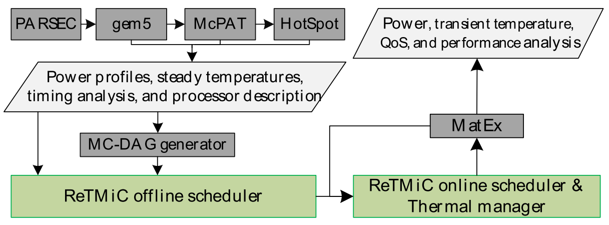

PARSEC
gem5
McPAT
HotSpot
Power, transient temperature, QoS, and performance analysis
Power profiles, steady temperatures, timing analysis, and processor description
MatEx
MC-DAG generator
ReTMiC offline scheduler
ReTMiC online scheduler & Thermal manager

FIGURE 9. Overview of our experimental framework.

in DAG-based hard real-time systems. They used the N-modular redundancy (NMR) technique for tolerating transient faults such that the chip-level power constraint is met.

- LE-NMR [41]: This method presents a low-energy NMR-based technique for multicore DAG-based hard real-time systems which reduces energy consumption while improving reliability. This paper is a suitable option for comparison because the system, reliability, and fault models are similar to our proposed method.

It is worth mentioning that in the hard real-time methods, which were considered in our comparison (TP3M and LE-NMR), we assumed that all tasks were executed with their WC^{HI} because these methods did not contain any strategy to deal with the overrun phase in mixed-criticality tasks with two kinds of WCETs.

B. EVALUATING TRANSIENT TEMPERATURE

This section evaluates the temperature of different methods. We show the effectiveness of ReTMiC in terms of thermal balancing and reducing the temperature. We consider a system with a 16-core processor and a task graph with 100 tasks. After scheduling tasks with different methods, we extract the transient temperature for each method. Fig. 9 shows the average transient temperature of the platform for different methods. In Fig. 9a, the maximum temperature of ReTMiC in the offline phase is shown. Note that the maximum spatial $\Delta\theta$ of ReTMiC in the offline phase is about 15°C. However, in Fig. 9b, after applying the run-time part of the ReTMiC scheme, the maximum transient temperature among all the cores is about 71.22°C, and the maximum $\Delta\theta$ is about 6°C. From Fig. 9c and Fig. 9d, it is clear that other methods

TABLE 5. The details of simulation configuration.

Name	Configuration
Core Type	ARM Cortex A15
Feature Size	22nm
# Cores	4, 9, 16, 36
Core v-f 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
Chip Thickness	0.5mm
Heat Sink Thickness	0.7mm

are worse than our ReTMiC method in terms of $\Delta\theta$ and maximum transient temperature combination because they did not consider the temperature threshold. To arrange a more general analysis of the thermal behavior, we measure the average and peak temperature of each method 100 times with different configurations. The average temperature can give us a better sense of the long-term thermal behavior of each method. As shown in Fig. 10, ReTMiC provides on average 10.3°C (up to 12.8°C) peak temperature reduction, and 8.2°C (up to 9.7°C) average temperature reduction. As an example, in the TP3M method which provides the best results among the state-of-the-art, the setup with 36 cores reached the peak temperature equal to 88.5°C and an average temperature equal to 79.1°C, while our method reached the peak temperature equal to 81.8°C and the average temperature equal to 73.4°C.

C. EVALUATING THE SPATIAL THERMAL VARIATION ($\Delta\theta$)

To evaluate the thermal balancing of different methods, we have extracted two thermal factors (average $\Delta\theta$ and maximum $\Delta\theta$). Fig. 11a shows the maximum spatial $\Delta\theta$ (i.e., $\theta^C_{Max} - \theta^C_{Min}$ in a specific time) during the execution of tasks for each method. Fig. 11b shows the average spatial $\Delta\theta$ from the start to the end of executing tasks for each method. The experiments show that ReTMiC provides on average 1.9°C (up to 3.5°C) in terms of maximum $\Delta\theta$ reduction, and 1.3°C (up to 2.4°C) in terms of average $\Delta\theta$ reduction in comparison to other methods. For example, in 36 cores configuration, the maximum and average $\Delta\theta$ in the TP3M method are 18.9°C

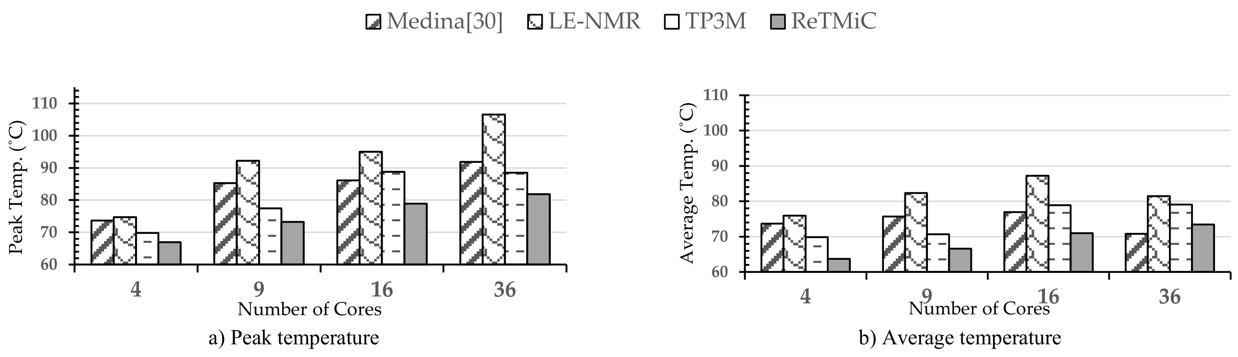


FIGURE 10. The resulting transient temperature on the chip.

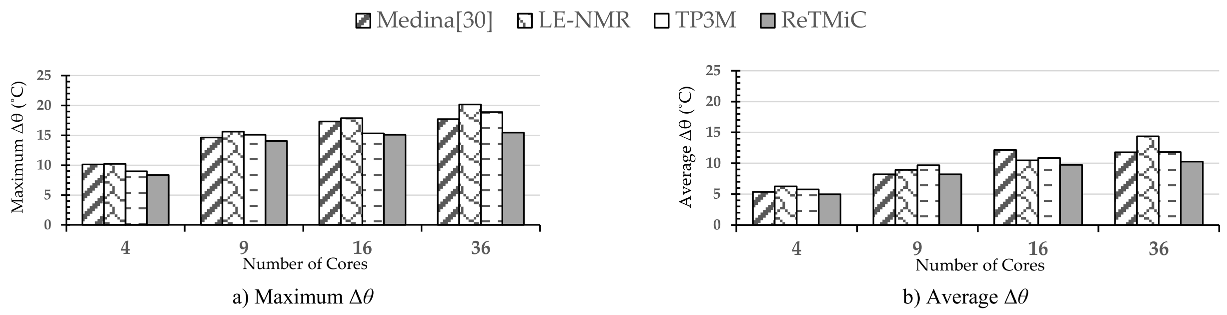


FIGURE 11. The resulting spatial $\Delta\theta$ temperature on the chip.

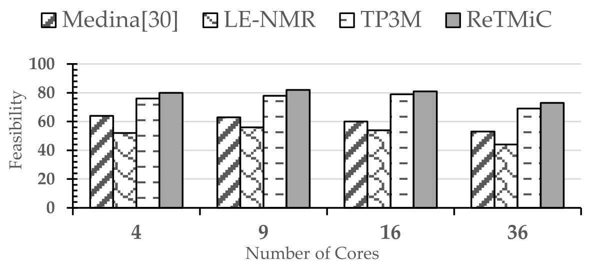


FIGURE 12. Comparing the feasibility based on worst-case scenario in theoffline phase.

and 10.9°C respectively, while the values for our method are 15.4°C and 9.6 °C, respectively.

D. EVALUATING THE FEASIBILITY

In this subsection, we discuss the feasibility of different methods in the worst-case scenario on different numbers of cores. We define feasibility as the percentage of satisfying both timing and TDP constraints. The results are reported from the offline phase with the worst-case scenario (see Fig. 12). As it is shown in Fig. 12, ReTMiC provides, on average, 15.5% improvement in terms of feasibility compared to the other state-of-the-art methods.

E. EVALUATING THE PEAK POWER CONSUMPTION

Fig. 13 shows the normalized peak power consumption in the worst-case scenario to the TDP constraint for Medina, Medina with replication, TP3M, LE-NMR, and ReTMiC for different numbers of cores in different parallelism degrees. The height of a task graph can be used to determine the parallelism degree for task graphs with a specific number of tasks [13]. If 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, the heights of the task graphs with high parallelism degrees, medium parallelism degrees, and low parallelism degrees are $1\leq h \leq n/3$, $n/3\leq h \leq 2n/3$, and, $2n/3\leq h \leq n$, respectively.

It should be noted that in Medina with replication to have a fair comparison we have adopted Medina [38] method with task replication fault-tolerant technique. As can be seen, only TP3M and ReTMiC meet TDP constraints in all scenarios, because other methods (LE-NMR and Medina) are very dependent on the parallelism degree of the task graph, and with increasing the parallelism degree, they violate the TDP constraint. In summary, ReTMiC provides on average 30.11% peak power reduction as compared to the other methods. Although both TP3M and our scheme respect the TDP constraint, as demonstrated in Sections V-B and V-C, our method achieves a smaller transient temperature and spatial thermal variation due to the online re-mapping technique.

F. EVALUATING THE QOS

The QoS of ReTMiC is reported in the offline phase and it is evaluated based on the percentages of HC tasks' overrun in the online phases. In this result, we assume a minimum guaranteed QoS. We set this minimum QoS to 30 and 60 percentages and report the average results. To show the effect

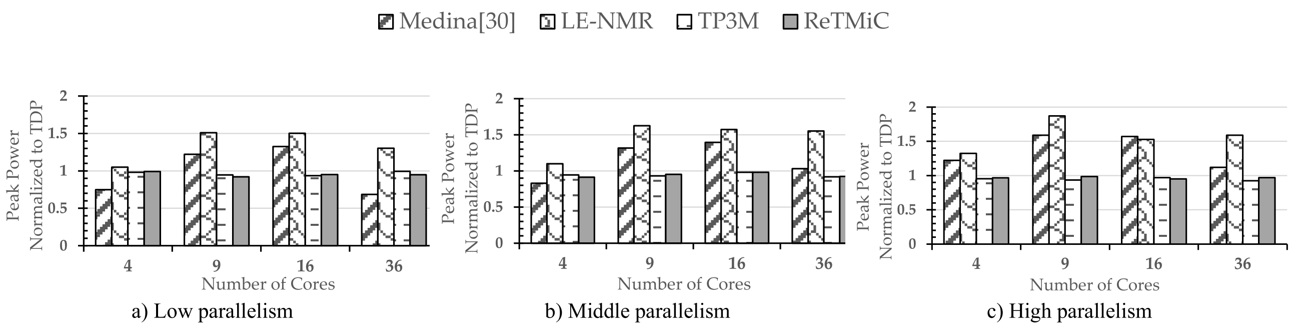


FIGURE 13. Normalized peak-power to the chip TDP in the worst-case scenario.

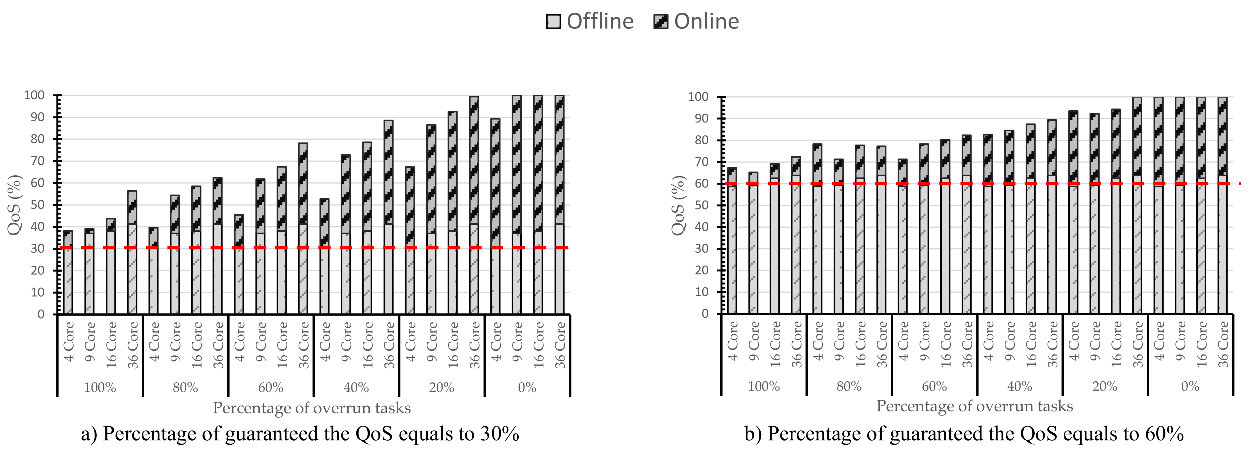


FIGURE 14. Comparing QoS of the offline and online phases.

of runtime scheduling, we assume that 50 percent of tasks in DAG are LC tasks. By increasing the percentage of HC tasks that overrun, the released dynamic slack time in runtime is reduced. As a result, the number of LC tasks that are scheduled at run time is reduced, and hence, the QoS is reduced. As shown in Fig. 14, the online improvement of QoS with increasing overrun percentage is decreased.

VI. CONCLUSION

This paper presents a thermal-aware task replication to deal with the processors' hot spots in fault-tolerant MCSs. In this regard, our proposed method employs TDP as the chip-level power constraint. We also determine balancing points and balancing factors at the offline phase to balance the temperature of the system in the runtime. At run time, we propose a lightweight online re-mapping technique that activates at the balancing points and uses a balancing factor to reduce the system's temperature based on the current temperature of each core and their corresponding task behavior. In the future, we will propose a method for mapping and scheduling mixed-criticality multi-DAGs on the heterogenous multicores while timing, TDP, and reliability constraints are met.

REFERENCES

[1] Y.-W. Zhang, R.-K. Chen, and Z. Gu, "Energy-aware partitioned scheduling of imprecise mixed-criticality systems," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 42, no. 11, pp. 3733–3742, Nov. 2023.

[2] A. Burns and R. I. Davis, "A survey of research into mixed criticality systems," *ACM Comput. Surveys*, vol. 50, no. 6, pp. 1–37, Nov. 2018.

[3] 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. Parallel Distrib. Syst.*, vol. 30, no. 10, pp. 2338–2354, Oct. 2019.

[4] A. Hossein Ansari, M. Ansari, and A. Ejlali, "TAFT: Thermal-aware hybrid fault-tolerant technique for multicore embedded systems," *IEEE Embedded Syst. Lett.*, vol. 16, no. 4, pp. 477–480, Dec. 2024.

[5] M. Ansari, S. Safari, A. Yeganeh-Khaksar, R. Siyadatzadeh, P. Gohari-Nazari, H. Khdr, M. Shafique, J. Henkel, and A. Ejlali, "ATLAS: Aging-aware task replication for multicore safety-critical systems," in *Proc. IEEE 29th Real-Time Embedded Technol. Appl. Symp. (RTAS)*, San Antonio, TX, USA, May 2023, pp. 223–234.

[6] 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.

[7] S. Pagani, H. Khdr, J.-J. Chen, M. Shafique, M. Li, and J. Henkel, "Thermal safe power (TSP): Efficient power budgeting for heterogeneous manycore systems in dark silicon," *IEEE Trans. Comput.*, vol. 66, no. 1, pp. 147–162, Jan. 2017.

[8] M. Ansari, S. Safari, S. Yari-Karin, P. Gohari-Nazari, H. Khdr, M. Shafique, J. Henkel, and A. Ejlali, "Thermal-aware standby-sparing technique on heterogeneous real-time embedded systems," *IEEE Trans. Emerg. Topics Comput.*, vol. 10, no. 4, pp. 1883–1897, Oct. 2022.

[9] F. S. Ghahfarokhi and A. Ejlali, "Schedule swapping: A technique for temperature management of distributed embedded systems," in *Proc. IEEE/IFIP Int. Conf. Embedded Ubiquitous Comput.*, Dec. 2010, pp. 1–6.

[10] S. Bertozzi, A. Acquaviva, D. Bertozzi, and A. Poggiali, "Supporting task migration in multi-processor systems-on-chip: A feasibility study," in *Proc. Design Autom. Test Eur. Conf.*, 2006, pp. 1–6.

[11] Z. Al-bayati, J. Caplan, B. H. Meyer, and H. Zeng, "A four-mode model for efficient fault-tolerant mixed-criticality systems," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Mar. 2016, pp. 97–102.

[12] S. Safari, S. Hessabi, and G. Ershadi, "LESS-MICS: A low energy standby-sparing scheme for mixed-criticality systems," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 39, no. 12, pp. 4601–4610, Dec. 2020.

[13] 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. Parallel Distrib. Syst.*, vol. 30, no. 1, pp. 161–173, Jan. 2019.

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

[15] M. Ansari, S. Safari, N. Rohbani, A. Ejlali, and B. M. Al-Hashimi, "Power-efficient and aging-aware primary/backup technique for heterogeneous embedded systems," *IEEE Trans. Sustain. Comput.*, vol. 8, no. 4, pp. 715–726, Oct./Dec. 2023.

[16] J. J. Chen, C. M. Hung, and T. W. Kuo, "On the minimization of the instantaneous temperature for periodic real-time tasks," in *Proc. 13th IEEE Real-Time Embedded Technol. App. Symp. (RTAS)*, Apr. 2007, pp. 236–245.

[17] S. Vestal, "Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance," in *Proc. 28th IEEE Int. Real-Time Syst. Symp. (RTSS)*, Dec. 2007, pp. 239–243.

[18] S. Baruah, H. Li, and L. Stougie, "Towards the design of certifiable mixed-criticality systems," in *Proc. 16th IEEE Real-Time Embedded Technol. Appl. Symp.*, Apr. 2010, pp. 13–22.

[19] S. Baruah and S. Vestal, "Schedulability analysis of sporadic tasks with multiple criticality specifications," in *Proc. Euromicro Conf. Real-Time Syst.*, Jul. 2008, pp. 147–155.

[20] S. Baruah, V. Bonifaci, G. D'angelo, H. Li, A. Marchetti-Spaccamela, S. Van Der Ster, and L. Stougie, "Preemptive uniprocessor scheduling of mixed-criticality sporadic task systems," *J. ACM*, vol. 62, no. 2, pp. 1–33, May 2015.

[21] P. Ekberg and W. Yi, "Bounding and shaping the demand of generalized mixed-criticality sporadic task systems," *Real-Time Syst.*, vol. 50, no. 1, pp. 48–86, Jan. 2014.

[22] H. Su, D. Zhu, and D. Mossé, "Scheduling algorithms for elastic mixed-criticality tasks in multicore systems," in *Proc. IEEE 19th Int. Conf. Embedded Real-Time Comput. Syst. Appl.*, Aug. 2013, pp. 352–357.

[23] H. Su, D. Zhu, and S. Brandt, "An elastic mixed-criticality task model and early-release EDF scheduling algorithms," *ACM Trans. Design Autom. Electron. Syst.*, vol. 22, no. 2, pp. 1–25, Apr. 2017.

[24] H. Li and S. Baruah, "Outstanding paper award: Global mixed-criticality scheduling on multiprocessors," in *Proc. 24th Euromicro Conf. Real-Time Syst.*, Jul. 2012, pp. 166–175.

[25] Z. Al-bayati, Q. Zhao, A. Youssef, H. Zeng, and Z. Gu, "Enhanced partitioned scheduling of mixed-criticality systems on multicore platforms," in *Proc. 20th Asia South Pacific Design Autom. Conf.*, Jan. 2015, pp. 630–635.

[26] H. Xu and A. Burns, "Semi-partitioned model for dual-core mixed criticality system," in *Proc. 23rd Int. Conf. Real Time Netw. Syst.*, Nov. 2015, pp. 257–266.

[27] T. Yang, Y. Tang, X. Jiang, Q. Deng, and N. Guan, "Semi-federated scheduling of mixed-criticality system for sporadic DAG tasks," in *Proc. IEEE 22nd Int. Symp. Real-Time Distrib. Comput. (ISORC)*, May 2019, pp. 163–170.

[28] J. Li, D. Ferry, S. Ahuja, K. Agrawal, C. Gill, and C. Lu, "Mixed-criticality federated scheduling for parallel real-time tasks," in *Proc. IEEE Real-Time Embedded Technol. Appl. Symp. (RTAS)*, Sep. 2016, pp. 1–12.

[29] P. Huang, H. Yang, and L. Thiele, "On the scheduling of fault-tolerant mixed-criticality systems," in *Proc. 51st ACM/EDAC/IEEE Design Autom. Conf. (DAC)*, Jun. 2014, pp. 1–6.

[30] A. Thekkilakattil, R. Dobrin, and S. Punnekkat, "Mixed criticality scheduling in fault-tolerant distributed real-time systems," in *Proc. Int. Conf. Embedded Syst. (ICES)*, Jul. 2014, pp. 92–97.

[31] P. Huang, P. Kumar, G. Giannopoulou, and L. Thiele, "Energy efficient DVFS scheduling for mixed-criticality systems," in *Proc. Int. Conf. Embedded Softw. (EMSOFT)*, Oct. 2014, pp. 1–10.

[32] S. Narayana, P. Huang, G. Giannopoulou, L. Thiele, and R. V. Prasad, "Exploring energy saving for mixed-criticality systems on multi-cores," in *Proc. IEEE Real-Time Embedded Technol. Appl. Symp. (RTAS)*, Apr. 2016, pp. 1–12.

[33] V. Legout, M. Jan, and L. Pautet, "Mixed-criticality multiprocessor real-time systems: Energy consumption vs deadline misses," in *Proc. Workshop Real-Time Mixed-Criticality Syst. (ReTiMiCS)*, 2013, pp. 1–6.

[34] M. Völp, M. Hahnel, and A. Lackorzynski, "Has energy surpassed timeliness? Scheduling energy-constrained mixed-criticality systems," in *Proc. IEEE 19th Real-Time Embedded Technol. Appl. Symp. (RTAS)*, Apr. 2014, pp. 275–284.

[35] H. Sobhani, S. Safari, J. Saber-Latibari, and S. Hessabi, "REALISM: Reliability-aware energy management in multi-level mixed-criticality systems with service level degradation," *J. Syst. Archit.*, vol. 117, Aug. 2021, Art. no. 102090.

[36] Z. Li, X. Hua, C. Guo, and S. Ren, "Empirical study of energy minimization issues for mixed-criticality systems with reliability constraints," in *Proc. LPDC*, 2014, pp. 1–22.

[37] A. Naghavi, S. Safari, and S. Hessabi, "Tolerating permanent faults with low-energy overhead in multicore mixed-criticality systems," *IEEE Trans. Emerg. Topics Comput.*, vol. 10, no. 2, pp. 985–996, Apr. 2022.

[38] R. Medina, E. Borde, and L. Pautet, "Directed acyclic graph scheduling for mixed-criticality systems," in *Reliable Software Technologies—Ada-Europe 2017: 22nd Ada-Europe International Conference on Reliable Software Technologies, Vienna, Austria, June 12-16, 2017, Proceedings 22*. Springer, 2017, pp. 217–232.

[39] R. Medina, E. Borde, and L. Pautet, "Availability enhancement and analysis for mixed-criticality systems on multi-core," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Mar. 2018, pp. 1271–1276.

[40] M. Ansari, A. Yeganeh-Khaksar, S. Safari, and A. Ejlali, "Peak-power-aware energy management for periodic real-time applications," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 39, no. 4, pp. 779–788, Apr. 2020.

[41] 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. Parallel Distrib. Syst.*, vol. 27, no. 5, pp. 1497–1510, May 2016.

[42] Z. Li, C. Guo, X. Hua, and S. Ren, "Reliability guaranteed energy minimization on mixed-criticality systems," *J. Syst. Softw.*, vol. 112, pp. 1–10, Feb. 2016.

[43] R. Sridharan and R. Mahapatra, "Reliability aware power management for dual-processor real-time embedded systems," in *Proc. 47th Design Autom. Conf.*, Jun. 2010, pp. 819–824.

[44] A. Meixner, M. E. Bauer, and D. Sorin, "Argus: Low-cost, comprehensive error detection in simple cores," in *Proc. 40th Annu. IEEE/ACM Int. Symp. Microarchitecture (MICRO)*, Dec. 2007, pp. 210–222.

[45] J. R. Sklaroff, "Redundancy management technique for space shuttle computers," *IBM J. Res. Develop.*, vol. 20, no. 1, pp. 20–28, Jan. 1976.

[46] L. E. P. Rice and A. M. K. Cheng, "Timing analysis of the X-38 space station crew return vehicle avionics," in *Proc. Real-Time Technol. App.*, Jun. 1999, pp. 255–265.

[47] Y. C. Bob Yeh, "Design considerations in Boeing 777 fly-by-wire computers," in *Proc. 3rd IEEE Int. High-Assurance Syst. Eng. Symp. (HASE)*, Nov. 1998, pp. 64–72.

[48] W. Huang, S. Ghosh, S. Velusamy, K. Sankaranarayanan, K. Skadron, and M. R. Stan, "HotSpot: A compact thermal modeling methodology for early-stage VLSI design," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 14, no. 5, pp. 501–513, May 2006.

[49] Y. Lee, H. S. Chwa, K. G. Shin, and S. Wang, "Thermal-aware resource management for embedded real-time systems," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 37, no. 11, pp. 2857–2868, Nov. 2018.

[50] N. Binkert, B. M. Beckmann, G. Black, S. K. Reinhardt, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, and D. A. Wood, "The gem5 simulator," *ACM SIGARCH Comput. Archit. News*, vol. 39, no. 2, pp. 1–7, May 2011.

[51] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi, "McPAT: An integrated power, area, and timing modeling framework for multicore and manycore architectures," in *Proc. 42nd Annu. IEEE/ACM Int. Symp. Microarchitecture (MICRO)*, Dec. 2009, pp. 469–480.

[52] C. Bienia, S. Kumar, J. P. Singh, and K. Li, "The PARSEC benchmark suite: Characterization and architectural implications," in *Proc. Int. Conf. Parallel Architectures Compilation Techn. (PACT)*, Oct. 2008, pp. 72–81.

[53] S. Pagani, J. J. Chen, M. Shafique, and J. Henkel, "MatEx: Efficient transient and peak temperature computation for compact thermal models," in *Proc. Design, Autom. Test Europe Conf. Exhib. (DATE)*, Apr. 2015, pp. 1515–1520.

[54] Y.-H. Gong, J. J. Yoo, and S. W. Chung, "Thermal modeling and validation of a real-world mobile AP," *IEEE Des. Test*, vol. 35, no. 1, pp. 55–62, Feb. 2018.

[55] B. Rouxel and I. Puaut, "STR2RTS: Refactored StreamIT benchmarks into statically analyzable parallel benchmarks for WCET estimation & real-time scheduling," in *Proc. 17th Int. Workshop Worst-Case Execution Time Analysis (WCET)*, 2017.

[56] G. C. Buttazzo, *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications*, vol. 24. Berlin, Germany: Springer, 2011.

[57] S. Safari, H. Khdr, P. Gohari-Nazari, M. Ansari, S. Hessabi, and J. Henkel, "TherMa-MiCs: Thermal-aware scheduling for fault-tolerant mixed-criticality systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 33, no. 7, pp. 1678–1694, Jul. 2022.

[58] M. Ansari, S. Safari, H. Khdr, P. Gohari-Nazari, J. Henkel, A. Ejlali, and S. Hessabi, "Power-aware checkpointing for multicore embedded systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 33, no. 12, pp. 4410–4424, Dec. 2022.

[59] S. Safari, M. Ansari, H. Khdr, P. Gohari-Nazari, S. Yari-Karin, A. Yeganeh-Khaksar, S. Hessabi, A. Ejlali, and J. Henkel, "A survey of fault-tolerance techniques for embedded systems from the perspective of power, energy, and thermal issues," *IEEE Access*, vol. 10, pp. 12229–12251, 2022.

[60] J. R. Azambuja, F. Kastensmidt, and J. Becker, *Hybrid Fault Tolerance Techniques to Detect Transient Faults in Embedded Processors*. Cham, Switzerland: Springer, 2014.

[61] A. Thekilakkattil, R. Dobrin, S. Punnekkat, and H. Aysan, "Optimizing the fault tolerance capabilities of distributed real-time systems," in *Proc. IEEE Conf. Emerg. Technol. Factory Autom.*, Palma de Mallorca, Spain, Sep. 2009, pp. 1–4.

[62] R. M. Pathan and J. Jonsson, "Exact fault-tolerant feasibility analysis of fixed-priority real-time tasks," in *Proc. IEEE 16th Int. Conf. Embedded Real-Time Comput. Syst. Appl.*, Macau, China, Aug. 2010, pp. 265–274.

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

[64] K. Cao, G. Xu, J. Zhou, M. Chen, T. Wei, and K. Li, "Lifetime-aware real-time task scheduling on fault-tolerant mixed-criticality embedded systems," *Future Gener. Comput. Syst.*, vol. 100, pp. 165–175, Nov. 2019.

[65] J. Caplan, Z. Al-bayati, H. Zeng, and B. H. Meyer, "Mapping and scheduling mixed-criticality systems with on-demand redundancy," *IEEE Trans. Comput.*, vol. 67, no. 4, pp. 582–588, Apr. 2018.

[66] S. Chakraborty, Y. Sharma, and S. Moulik, "TREAFET: Temperature-aware real-time task scheduling for FinFET based multicores," *ACM Trans. Embedded Comput. Syst.*, vol. 23, no. 4, pp. 1–31, Jul. 2024.

[67] Y. Sharma, S. Moulik, and S. Chakraborty, "RESTORE: Real-time task scheduling on a temperature aware FinFET based multicore," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Mar. 2022, pp. 608–611.

[68] Y. Sharma and S. Moulik, "CETAS: A cluster based energy and temperature efficient real-time scheduler for heterogeneous platforms," in *Proc. 37th ACM/SIGAPP Symp. Appl. Comput.* New York, NY, USA: Association for Computing Machinery, Apr. 2022, pp. 501–509.

[69] Y. Sharma, S. Gupta, and S. Moulik, "An interplay of energy and temperature minimization techniques for heterogeneous multiprocessor systems," in *Proc. IEEE Region 10 Conf. (TENCON)*, Chiang Mai, Thailand, Oct. 2023, pp. 629–634.

[70] Y. Sharma, S. Chakraborty, and S. Moulik, "ETA-HP: An energy and temperature-aware real-time scheduler for heterogeneous platforms," *J. Supercomput.*, vol. 78, no. 8, pp. 1–25, May 2022.

[71] S. Safari, S. Shokri, S. Hessabi, and P. Lotfi-Kamran, "LEC-MiCs: Low-energy checkpointing in mixed-criticality multicore systems," *ACM Trans. Cyber-Phys. Syst.*, vol. 9, no. 1, pp. 1–29, Jan. 2024.

[72] S. K. Roy, R. Devaraj, A. Sarkar, and D. Senapati, "SLAQA: Quality-level aware scheduling of task graphs on heterogeneous distributed systems," *ACM Trans. Embedded Comput. Syst.*, vol. 20, no. 5, pp. 1–31, Sep. 2021.

[73] R. Devaraj and A. Sarkar, "Resource-optimal fault-tolerant scheduler design for task graphs using supervisory control," *IEEE Trans. Ind. Informat.*, vol. 17, no. 11, pp. 7325–7337, Nov. 2021.

[74] T. Li, T. Zhang, G. Yu, Y. Zhang, and J. Song, "TA-MCF: Thermal-aware fluid scheduling for mixed-criticality system," *J. Circuits, Syst. Comput.*, vol. 28, no. 2, Feb. 2018, Art. no. 1950029.

[75] S. K. Roy, R. Devaraj, and A. Sarkar, "Contention cognizant scheduling of task graphs on shared bus-based heterogeneous platforms," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 41, no. 2, pp. 281–293, Feb. 2022.

SEPIDEH SAFARI received the Ph.D. degree in computer engineering from the Sharif University of Technology, Tehran, Iran, in 2021. She was a Visiting Researcher with the Chair for Embedded Systems (CES), Karlsruhe Institute of Technology (KIT), Germany, from 2019 to 2021. She is currently a Senior Postdoctoral Researcher with the Institute for Research in Fundamental Sciences (IPM), Tehran. Her research interests include the scheduling of real-time systems, low-power/energy design of embedded and cyber-physical systems, fault-tolerant mixed-criticality systems, scheduling algorithms, and distributed multicore systems, with a focus on dependability/reliability. She was a Technical Program Committee (TPC) Member of several conferences, such as RTSS-2024, DSD-2022-2023-2024, and CPSAT-2024. She is an Associate Editor of *Microprocessors and Microsystems* (MICPRO) (Elsevier).

MOHSEN ANSARI received the Ph.D. degree in computer engineering from the Sharif University of Technology, Tehran, Iran, in 2021. He was a Visiting Researcher with the Chair for Embedded Systems (CES), Karlsruhe Institute of Technology (KIT), Germany, from 2019 to 2021. He is currently an Assistant Professor of computer engineering with the Sharif University of Technology. He is also the Director of the Cyber-Physical Systems Laboratory (CPSLab), Sharif University of Technology. His research interests include embedded machine learning, low-power design, real-time systems, cyber-physical systems, and hybrid systems design. He was a Technical Program Committee (TPC) Member of ASP-DAC (2022, 2023, and 2024). He is serving as an Associate Editor for IEEE Embedded Systems Letters (ESL).

SHAAHIN HESSABI (Member, IEEE) received the B.S. and M.S. degrees in electrical engineering from the 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, Waterloo, ON, Canada. He joined the Department of Computer Engineering, Sharif University of Technology, as a Faculty Member, in 1996. He has published more than 130 refereed articles in related areas. His research interests include cyber-physical systems, reconfigurable and heterogeneous architectures, and efficient processing and communication architectures.

JÖRG HENKEL (Fellow, IEEE) received the Diploma and Ph.D. (summa cum laude) degrees from the Technical University of Braunschweig, Germany. He was a Research Staff Member with NEC Laboratories, Princeton, NJ, USA. He coordinates the DFG Program SPP 1500 "Dependable Embedded Systems." He is currently a Site Coordinator of the DFG-TR89 Collaborative Research Center on "Invasive Computing." His research work is focused on co-designing embedded hardware/software systems with respect to power, thermal, and reliability aspects. He has received six best paper awards from, among others, ICCAD, ESWeek, and DATE. He serves as a steering committee chair/member for leading conferences and journals. He is the Chairman of the IEEE Computer Society, Germany Chapter. He served as the Editor-in-Chief for ACM TECS and *IEEE Design & Test*. He is/has been an associate editor of major ACM and IEEE journals. He was the General Chair of ICCAD and ESWeek.