Power-Aware Checkpointing for Multicore Embedded Systems

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

Abstract—Increasing the number of cores integrated on a single chip offers a great potential for the implementation of fault-tolerant techniques to achieve high reliability in real-time embedded systems. Checkpointing with rollback-recovery is a well-established technique to tolerate transient faults in multicore platforms. To consider the worst-case fault occurrence scenario, checkpointing technique requires to re-execute some parts of the tasks, and that might lead to simultaneous execution of task parts with high power consumptions, which eventually might result in a peak power increase beyond the thermal design power (TDP). Exceeding TDP can elevate on-chip temperatures beyond safe limits, and thereby triggering countermeasures that throttle down the voltage and frequency levels or power gate the cores. Such countermeasures might lead to violating task deadlines and degrading the system's reliability. To avoid such severe scenarios, it is inevitable to consider the impact of applying fault-tolerant techniques on the power consumption and prevent violating the power constraint of the chip, i.e., TDP. This paper presents for the first time, a peak-power-aware checkpointing (PPAC) technique that tolerates a given number of faults, *k*, while at the same time meets the power constraint in hard real-time embedded systems. To do this, our proposed technique (PPAC) adjusts the timing of the checkpoints, which have lower power consumption than the tasks to the execution time points that have power spikes beyond TDP. Moreover, PPAC exploits the available slack times on the cores to delay the execution of some tasks to avoid the remaining power spikes beyond TDP, which could not be mitigated by solely adjusting checkpoints. To evaluate our technique, we extend the state-of-the-art system-level simulator, gem5, with the state-of-the-art checkpointing module in Linux. Our experimental results show that our proposed technique is able to tolerate a given number of faults without exceeding the timing and power constraints in hard real-time embedded systems. The resulting peak power reduction achieved by our technique compared to state-of-the-art techniques is an average of 23%. Moreover, our technique employs the Dynamic Power Management (DPM) during the slack times resulting at runtime in the case of fault-free scenarios, which provides energy savings with an average of 17.28% and up to 61.1%.

Index Terms— Peak Power Consumption, Checkpointing, Multicore Platforms, Embedded Systems

1 Introduction

DUE to the continued technology scaling, power densities on the chip are increasing and thereby, on-chip temperature is elevating [1][2]. High temperature jeopardizes chip reliability through aging mechanisms, e.g., electromigration, Negative Bias Temperature Instability (NBTI) [3][4], and it might even lead to permanent damage on the chip. In order to avoid temperature increases beyond safe limits, Dynamic Thermal Management (DTM) techniques are implemented on the chip [27]. In particular, DTM monitors the on-chip temperature during operational mode and takes some countermeasures, e.g., Dynamic Voltage and Frequency Scaling (DVFS) and Dynamic Power Management (DPM), to throttle down the chip, once the temperature approaches the thermal threshold [2]. Obviously, these countermeasures might lead to missing task deadlines, and this is not acceptable in hard real-time systems [5][8]. Moreover, downscaling the voltage and frequency levels degrades the system reliability, because it increases the fault rate, as demonstrated in [8]. As a remedy, system-level techniques [43] enforce power constraints on the chip aiming at suppressing power consumption increases and the potential thermal violations and thereby avoiding the corresponding countermeasures. TDP, thermal design power, is the most commonly used power constraint since it is the highest sustainable power that the cooling system in a computer is designed to dissipate under any workload [1][2].

Apart from power and thermal issues, transient faults, which are typically resulting due to high-energy particle strikes in hardware [12], are considered as severe reliability concerns. To ensure a reliable operation for the system, many system-level techniques, referred to as *fault-tolerant techniques*, have been typically proposed to tolerate a given number of faults. These techniques, however, incur significant time and power overheads, and therefore, most of

- *M. Ansari, P. Gohari-Nazari, A. Ejlali and S. Hessabi are with the Department of Computer Engineering, Sharif University of Technology, Tehran 14588, Iran (e-mails: {ansari, ejlali, hessabi}@sharif.edu, gohary@ce.sharif.edu).*
- *S. Safari is with the School of Computer Science, Institute for Research in Fundamental Sciences (IPM), Tehran, Iran (e-mail: sepideh.safari@ipm.ir).*
- *H. Khdr and J. Henkel are with the Karlsruhe Institute of Technology, Karlsruhe 76131, Germany (e-mails: {heba.khdr; henkel}@kit.edu).*

Manuscript received 16 Oct. 2021; revised 26 June 2022; accepted x m y. Date of publication X Y Z; date of current version X Y Z.
(Corresponding author: Mohsen Ansari and Alireza Ejlali.)
Recommended for acceptance by X. X.
Object Identifier no. x/TPDS.y.z

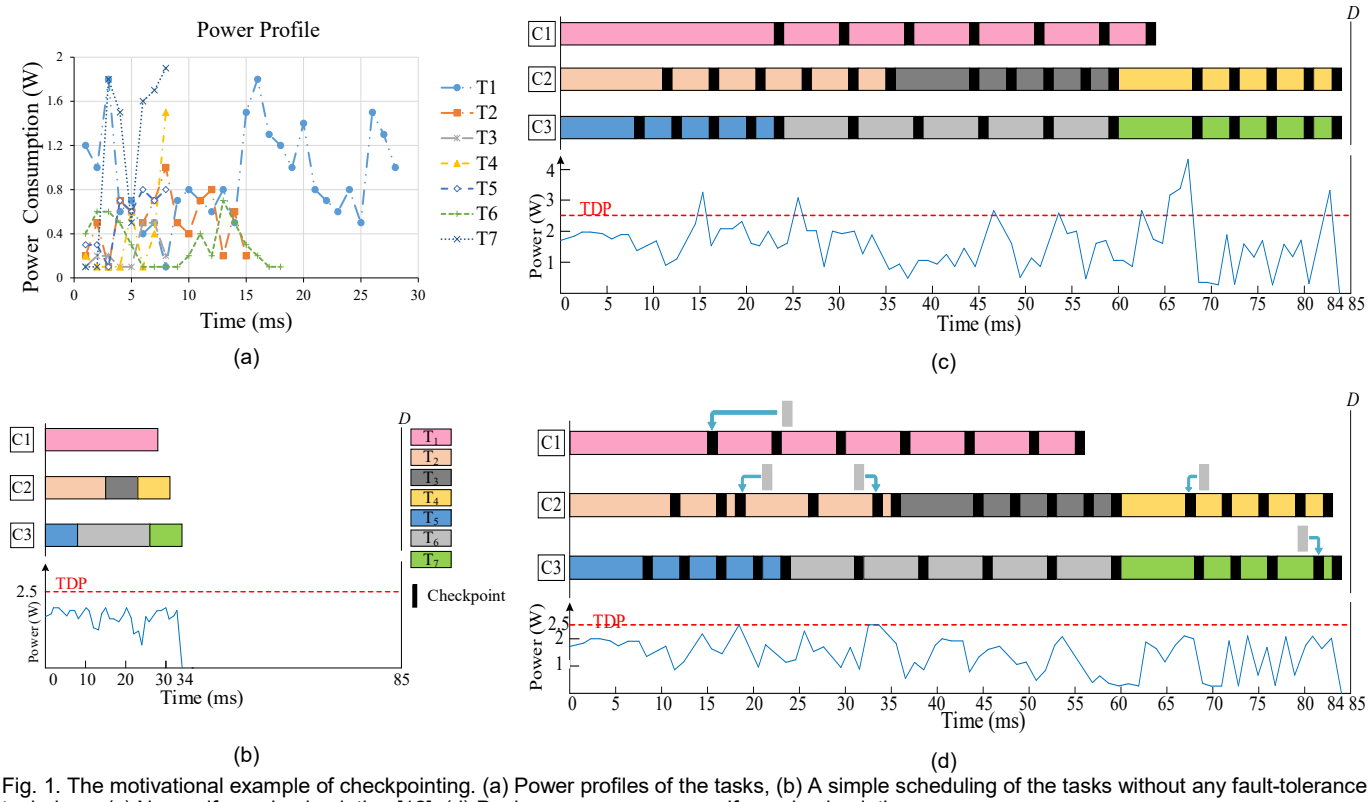

Fig. 1. The motivational example of checkpointing. (a) Power profiles of the tasks, (b) A simple scheduling of the tasks without any fault-tolerance technique, (c) Non-uniform checkpointing [12], (d) Peak-power-aware non-uniform checkpointing.

these mechanisms may not be feasible and applicable in hard real-time embedded systems [5][12]. Checkpointing with rollback recovery is a fault-tolerant technique that can result in lower power and time overheads, if intelligently used, compared to re-execution, replication, triple modular redundancy (TMR), standby-sparing, and etc. [12][25][28][29]. In particular, when a fault occurs, there is no need to replicate or re-execute the whole task, and only the part of the task, which begins from the last safe checkpoint, is required to be executed again [12]. Obviously, this strategy will reduce time and power overheads, compared to re-executing or duplicating the whole task. Although checkpointing has less power overhead but it requires to re-execute some parts of the tasks. That might lead to simultaneous execution of task parts with high power consumption, which eventually might result in a peak power increase beyond thermal design power (TDP).

In order to solve this problem, this paper proposes for the first time a peak-power-aware checkpointing technique that tolerates k faults in hard real-time embedded systems, while at the same time meets the TDP constraint. There are two types of checkpointing techniques proposed in the literature; uniform checkpointing and non-uniform checkpointing. The former inserts checkpoints regularly at a specific time period, while the latter allows selecting uneven timing periods between the checkpoints. Our proposed technique employs non-uniform checkpointing to be able to adjust the timing of the checkpoints so that the TDP constraint is satisfied. In particular, inserting a checkpoint for a task will suspend the task and execute a set of instructions to save the state of the task into the memory. Normally, the power consumption during the checkpoint is much less than the power consumption of the task [12], as will be demonstrated later in Section 5. Therefore, our proposed technique inserts the checkpoints at time points, at which TDP violation is expected. The following motivational example shows how adjusting the checkpoint timing helps to satisfy the TDP constraint.

1.1 Motivational Example

In this example, a triple-core chip with 2.5W of TDP (this assumption is similar to the TDP values in real embedded systems, e.g., ARM processor [20]) is considered that executes seven frame-based tasks T_1-T_7. All the tasks arrive at time t=0ms and have the same deadline D=85ms. The execution time of T_1, T_2, T_6, and {T_3, T_4, T_5, T_7} are 28ms, 15ms, 18ms, and 8ms, respectively. The worst-case power profile of each task is shown in Fig. 1a. To be reliable, it is assumed that this system needs to tolerate 2 faults. The tasks have been scheduled using the scheduling policy presented in [12]. As shown in Fig. 1b, the tasks T_1, {T_2, T_3, T_4}, and {T_5, T_6, T_7} are mapped on C1, C2, and C3, respectively. The resulting total power consumption through the execution time of the frame is depicted in the curve shown below the scheduling. As it can be observed from this curve, the adopted scheduling policy [11] meets the power constraint of the system but it cannot tolerate even a single fault occurrence (i.e., this policy does not support any fault-tolerant technique). Therefore, the system will fail if a fault occurs.

In order to tolerate two faults, the state-of-the-art non-uniform checkpointing technique [12] is employed as

shown in Fig. 1c, where the checkpoints are represented by black rectangles. This technique uses non-uniform intervals that are determined by the equations detailed in [12], where the time and power overheads of the checkpoint insertion are equal to 1ms and 0.1W, respectively. As it can be observed from the power curve, applying the checkpointing technique has led to violating TDP. The reason is that the execution phases of the tasks with high power consumption have been executed at the same time, after inserting the checkpoints. For instance, as shown in Fig. 1c in the time intervals [10ms to 20ms] and [60ms to 70ms] the total power consumption of the chip is higher than TDP. Note that prolonging the execution time (for example for task T_1 from 28ms to 63ms) is due to tolerating 2 faults during the execution time of tasks.

In Fig. 1d the timing of the non-uniform checkpoints is rearranged such that the TDP constraint is met. In particular, the time intervals, at which TDP is violated, are selected and then some checkpoints are shifted to those time points in order to meet the TDP constraint. This is possible, because inserting a checkpoint will suspend the task execution, and therefore it will significantly reduce the power consumption at that point of time (see Section 5), which helps to satisfy the TDP constraint. In Fig. 1d, the gray rectangles at the beginning of the blue arrows indicate the old time points where checkpoints need to be shifted, and the black rectangles at the end of the blue arrows show the new timing of the shifted checkpoints such that TDP is met. For example, at t=15ms, the tasks on the first, second and third cores consume 1.8W, 0.7W, and 0.8W respectively (overall 3.3W). To satisfy TDP, the closest checkpoint on C1 has been shifted to the timing point of t=15ms, and hence, the overall power consumption becomes 1.6W (i.e., less than TDP) at t=15ms. However, because of shifting this checkpoint on C1, a new TDP violation is introduced at t=17ms. To satisfy TDP again, the third checkpoint of T_2 on C2 has been shifted to the violation point. In a similar way, all other TDP violation points have been handled. Hence, it can be observed, how *adjusting checkpoint timing can help satisfying TDP constraint, without any additional overhead for power management.*

1.2 Our Novel Contribution

As observed from the motivational example, adjusting checkpoint timing can help satisfying TDP constraint, without any additional overhead. However, the challenges are, how to select the tasks whose checkpoints need to be shifted, how many checkpoints need to be selected, and what if shifting the checkpoints is not enough to satisfy TDP constraint. All of these challenges have been tackled in our proposed checkpointing technique, as will be explained later on. Our novel contributions are (1) Proposing a peak-power-aware non-uniform checkpointing such that hard real-time constraints are met; (2) Presenting a shifting method for the tasks to satisfy TDP constraint; (3) Delaying some parts of the tasks to meet TDP by considering the hard real-time constraints.

This paper presents, for the first time, a peak-power-aware checkpointing (PPAC) technique that tolerates *a given number of faults* in hard real-time embedded systems, while at the same time meets the power constraint. First of all, an initial scheduling and checkpointing are performed to tolerate the given number of faults and considering the timing constraint. Afterwards, our proposed technique adjusts the checkpoint timing to mitigate the power spikes that exceed TDP. In case the number of checkpoints was not enough to meet TDP constraint throughout the whole execution time, the available slack times on the cores will be exploited to delay the execution of some tasks in order to avoid the remaining power spikes beyond TDP without any additional overhead.

Due to the rare nature of fault occurrence, the fault-free scenario will occur much more than the faulty one. In the fault-free scenario, the repeated parts of the tasks can be dropped out, thereby many slack times result in throughout the execution time. To provide power and energy savings, our technique drops out the unnecessary repeated parts of the tasks and applies DPM in the resulting slack times.

Moreover, in order to evaluate our technique, we have extended the state-of-the-art system-level simulator, gem5, with the state-of-the-art checkpointing module in Linux, which has not been done before, for the best of our knowledge. Our experimental results show that our technique is able to tolerate a given number of faults (between 1 to 5) faults without exceeding the timing and power constraints in hard real-time embedded systems. The details of the proposed technique will be discussed later in Section 4.

2 Models and Assumptions

In this section, we present our system, application, power, and fault models. We also provide reliability modeling of our system in this section.

2.1 System and Application Model

This paper considers a multicore system with m homogeneous cores $C=\{C_1, C_2, \ldots, C_m\}$. The system executes a set of n frame-based hard real-time tasks $\Psi=\{T_1, T_2, \ldots, T_n\}$ where the tasks are released at the same time and share a common deadline D. This application model is adopted in many related works like [6][7]. Each task T_i has a worst-case execution time wc_i. Note that this assumption is valid for frame-based tasks and cases where multiple tasks belong to one complex multi-tasked application [11]. Examples of applications that require hard real-time computing include nuclear power plants, railway switching systems, automotive and avionics systems, air traffic control, telecommunications, some robotics, and military systems. In the last several years, hard real-time computing has been required in new applications areas, such as medical equipment, consumer electronics, flight control systems, and cyber-physical systems [5][11][44][45].

2.2 Power Model

The power consumption of each core consists of static and dynamic power components [2][8][12][24]. The static

power (P_{static}) is dominated by the leakage current. Dynamic power ($P_{dynamic}$) is mainly consumed due to activity resulting by executing the task [22][26].

$$P(V_i, f_i) = P_{static} + P_{dynamic} = I_{sub}V_i + \alpha C_L V_i^2 f_i \quad (1)$$

where C_L is the average switched capacitance, V_i and f_i are supply voltage and operational frequency, I_{sub} is the sub-threshold leakage current and (a) is the activity factor. The dynamic power will change through the execution time of the task, that has been considered by our technique. Moreover, different tasks result in different power profiles. Intuitively, the total power consumption of the chip at a specific time point is the summation of the power consumption of all cores at that point of time.

$$P_{total} = \sum_{i=1}^{m} P(V_i, f_i) \quad (2)$$

In order to show the total power consumption in different time points, we define a vector $\boldsymbol{P}$. Each entry of $\boldsymbol{P}$ shows the total power consumption of the chip at each time point.

2.3 Reliability and Fault Model

In hard real-time embedded systems, we should have both correct executions of tasks (i.e., functional reliability) and also meeting the deadlines (i.e., timing reliability) [42][47]. This is because any faulty result or any missed deadline may result in a system failure. In order to model the reliability, like the works in [8][25][24][35][42][46][47], we exploit the "probability of timely completion" to consider both meeting the deadlines and correctness of tasks' execution. This is because, as we use the checkpointing mechanism, a fault within the task can be tolerated but with a time penalty which reduces the probability of timely completion. Note that in this paper transient faults are considered as the source of fault occurrence, i.e., bit upsets in the underlying hardware. Such transient faults are typically assumed to follow a Poisson process with the rate λ [7][8][12][21][23][46]. The fault rate varies exponentially as the supply voltage V changes. Therefore, the raw fault rate $\lambda(V)$ corresponding to the supply voltage V can be written as follows:

$$\lambda(V) = \lambda_0 10^{\frac{V_{max}-V}{\Delta}} \quad (3)$$

where λ_0 is the fault rate corresponding to the maximum voltage (V=V_{max}) and Δ is a parameter that shows how the fault rate increases with voltage decrease. Since the voltage and frequency scaling increases the fault rate (see Eq. 3) and degrades the task's reliability due to increasing the worst-case execution time (see Eq. 4), we do not employ DVFS. Therefore, λ_0 is the fault rate of this paper. A transient fault in the underlying hardware may ultimately result in a software failure. To measure the software failures due to transient faults, we use a state-of-the-art software reliability model called the *Function Vulnerability Index* (*FVI*) [9]. This measures the software failure probability and accounts for both spatial and temporal vulnerabilities of different instructions [9]. Therefore, the software failure rate due to transient faults can be modeled as $\lambda_0 \times FVI$. The

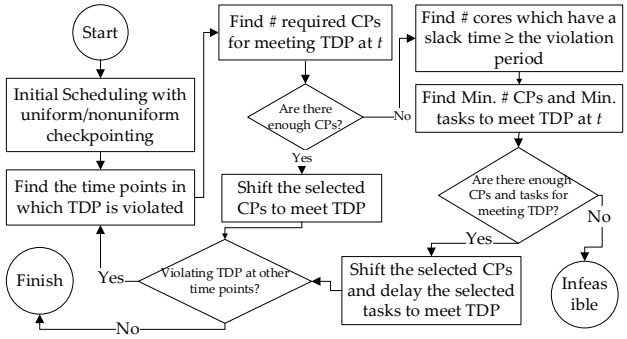

Fig. 2. Flow diagram of the PPAC algorithm

reliability of task i execution is computed as:

$$R_i = e^{\lambda_0 \times FVI \times wc_i} \quad (4)$$

It should be noted that, due to the Function Vulnerability Index, different applications have different reliability.

3 Problem Definition

Given a multicore system with m cores that executes a set of n frame-based hard real-time tasks, where the tasks arrive at the same time and have a common deadline D, and each task T_i has a worst-case execution time wc_i. The goal is to tolerate *a given number of faults* in hard real-time embedded systems such that the power and timing constraints are met. The problem is how to find the task-to-core assignment, the scheduling of the tasks, the required number of checkpoints of each task, and their timing to achieve the target goal.

- The power consumption is represented by the matrix $P \epsilon \mathbb{R}^{n \times m \times l}$, in which each element P_{ijt} denotes the power consumption for the task i the core j at a time point t.
- The task-to-core mapping is represented by the matrix $X \epsilon \{0,1\}^{n \times m}$. The task i is mapped to the core j if and only if $X_{ij} = 1$.

In the following, we formulate the constraints that need to be jointly satisfied by the proposed method.

Chip Power Constraint: The power consumption of the chip, i.e., the sum of the power of all underlying cores, should be less than TDP.

$$\sum_{i,j} X_{i,j} P_{i,j,t} \leq P_{TDP,chip}, \text{ at each time point } t \quad (5)$$

Task Timing Constraint: The worst-case execution time of all assigned tasks to the core j with their checkpointing time overhead *(CK_t)* should not exceed the task timing constraint (defined by the D).

$$\forall j: \sum_{i} (X_{i,j} wc_i + CK\_t(T_i)) \leq D \quad (6)$$

The minimum number of checkpoints: The proposed technique determines the number of checkpoints for the tasks such that they can tolerate k faults. As long as the number of faults does not exceed this value, the task can recover from faults using the employed checkpoints. The required number of uniform checkpoints that tolerates k faults is calculated by Eq. 7 as explained in [30].

Table 1. The Notation of the Parameters used in Algorithm 1

Notation	Description
Violations	A list of the violation points of TDP
cur.vio	The current TDP violation point to be handled
cur.vio.period	The period of the current TDP violation point to be handled
cur.vio.p	The power consumption of the current TDP violation point to be handled
cur.vio.t	The exact time of the current TDP violation point in the frame to be handled
Ψ	A subset of available checkpoints with minimum number of checkpoints that meet TDP at *cur.vio.t*;
C	A set of cores whose slack time ≥ cur.vio.period
$\boldsymbol{P}$	The power consumption of the system at each time

$$n_i(k, wc_i) = \begin{cases} n^- = \left\lfloor \sqrt{\frac{k \times wc_i}{CK\_t(T_i)}} \right\rfloor, & \text{if } wc_i \le \frac{n^-(n^-+1)}{k} CK\_t(T_i) \\ n^+ = \left\lceil \sqrt{\frac{k \times wc_i}{CK\_t(T_i)}} \right\rceil, & \text{if } wc_i > \frac{n^-(n^-+1)}{k} CK\_t(T_i) \end{cases} \quad (7)$$

However, non-uniform checkpointing can tolerate the same number of faults with a smaller number of checkpoints, by using the algorithm proposed in [12], which considers the resulting number of checkpoints from equation 7 as an input. In this paper, we follow the same strategy proposed in [12] to employ the minimum number of checkpoints and thereby the minimum time and power overhead. To determine the non-uniform checkpoint intervals, the method in [12] postpones checkpoint insertions as much as possible such that fewer checkpoints are inserted in fault-free scenarios. To reserve the minimum possible time for faulty states, at first, they use the optimal uniform checkpointing and then reduce the number of checkpoints based on the fault occurrence (see Sections 2 and 3 in [12]). Note that the optimal uniform checkpointing minimizes the application execution time in the worst-case fault scenario. They have proposed a non-uniform checkpointing scheme that enables checkpointing at selective locations while considering applications deadline, execution time and a user-defined number of tolerable faults to curtail the time and power overheads.

Core Assignment Constraint: Each task can be only mapped to a core.

$$\forall i: \sum_j X_{i,j} = 1 \quad (8)$$

The mentioned problem is categorized as an NP-Hard problem [2][6][31]. The optimal solution is finding a feasible scheduling that meets all of the mentioned constraints if it exists. It should be noted that an exhaustive search to find all combinations results in an exponential time complexity equal to $O(m^n)$.

Hence, this paper proposes a heuristic-based algorithm to provide a solution for the presented problem in polynomial time.

Algorithm 1. Peak-Power-Aware Checkpointing

Inputs: A set of n tasks, Set of cores, Tasks' power profile, and Chip-level TDP constraint.
Output: The tasks scheduling, checkpoint timing.

BEGIN:
1: *Initial scheduling with uniform/non-uniform checkpointing;*
2: *Initialize the vector of total power consumption* ***P*;**
3: *Violations* ← *Violation points of TDP*; //a list (*t*, *p*, *period*)
4: **while** (*Violations*!=Φ) **do**
5: *cur.vio* ← *the current TDP violation point to be handled;*
////////////////// **Adjusting Checkpoint Timing** //////////////////
6: Ψ ← Find available CPs for *cur.vio.t*;
7: **if** ($\sum_{i\in\psi} T_i.P - \sum_{i\in\psi} T_i.CK\_P \le cur.vio.p$)
8: Ψ ← Find a subset of Ψ with min. # of CPs that meet TDP
-- at *cur.vio.t*;
9: *Shift*(Ψ); // set the checkpoint time to *cur.vio.t*
10: $\boldsymbol{P}(cur.vio.t) \leftarrow \boldsymbol{P}(cur.vio.t) - \sum_{i\in\psi} T_i.P + \sum_{i\in\psi} T_i.CK\_P$;
11: **else**
////////////////// **Delaying Task Execution** //////////////////
12: C ← Set of cores whose slack time≥ *cur.vio.period*;
13: Ψ_1 ← Find tasks which are in C_1 and executing within
-- *cur.vio.period*;
14: *CL* ← *create_all_combinations*(Ψ_1, Ψ);
15: (Ψ_2, Ψ_3) ← Find a combination of *CL* with min. # tasks
-- that meet TDP at *cur.vio.t*;
17: **if** (Ψ_2= Φ & Ψ_3= Φ)
18: **return** *infeasible*;
19: **break**;
20: *Shift*(Ψ_2); // set the checkpoint time to *cur.vio.t*
21: *Delay*(Ψ_3, *cur.vio.period*);//delay the tasks with *cur.vio.period*
22: $\boldsymbol{P}(cur.vio.t) \leftarrow \boldsymbol{P}(cur.vio.t) - \sum_{i\in\psi_2\&\psi_3} T_i.P + \sum_{i\in\psi_2} T_i.CK\_P$;
23: *Update*(*Violations*);
24: **end while;**
END

4 Peak-Power-Aware Checkpointing

To solve the aforementioned problem, we propose a peak-power-aware checkpointing (PPAC) that tolerates *a given number of faults* in hard real-time embedded systems, while at the same time satisfying the predefined power constraint, which is TDP.

As discussed earlier, non-uniform checkpointing allows uneven distribution of the checkpoints throughout the execution time, in contrast to the uniform checkpointing which distributes the checkpoints evenly, so that they are separated by identical time intervals. Therefore, we employ non-uniform checkpointing to be able to adjust the timing of the checkpoints so that the TDP constraint is satisfied.

An overview of our proposed method (PPAC) is shown in Fig. 2, while the details of our method are explained in the following subsections. The first step of the proposed method is to perform initial scheduling of the tasks with non-uniform checkpointing. Then, PPAC finds the time points at which TDP is violated. For meeting TDP at the found time points, PPAC first tries to find the required checkpoints that meet TDP if shifted to the violation time points. If there are enough checkpoints for meeting TDP, PPAC shifts them to the selected time point. Otherwise, the available slack time on the cores will be exploited to delay some tasks so that the TDP constraint is satisfied. To do this, PPAC should find the minimum number of

checkpoints that need to be shifted and the minimum number of tasks that need to be delayed, in order to satisfy TDP. If exploiting both checkpoints and delaying the tasks does not satisfy TDP, that means there is no feasible schedule for the given tasks.

These functionalities, i.e., initial scheduling, adjusting checkpoint timing, exploiting slack time for delaying task execution, are conducted at design time, using a heuristic, whose pseudo-code is listed in Algorithm 1. Additionally, at runtime, we also exploit released slack times of fault-free scenarios in order to further power/energy reduction, by applying the Dynamic Power Management technique. The following subsections illustrate the details of the proposed techniques.

4.1 Initial scheduling and checkpointing

Before inserting checkpoints, the tasks need to be mapped on the cores. In particular, tasks are mapped based on Worst Fit Decreasing (WFD) bin packing, which is a well-known technique that is typically used in state-of-the-art scheduling techniques (e.g., [5][32]). In the WFD bin packing, the cores are sorted in increasing order in terms of utilization, while the tasks are sorted in decreasing order in terms of the worst-case execution time. Then, the first task in the list is mapped to the first core in the list, and the order of the list of the core will be updated after each mapping.

After mapping the tasks, the algorithm determines the initial scheduling with non-uniform checkpointing according to the priority-based scheduling policy [11] in line 1 of Algorithm 1. Note that Table 1 describes the variables used in Algorithm 1. To do that, the number of checkpoints needs to be determined. As explained in Section 3, we select the minimum non-uniform checkpoints to tolerate k faults in order to reduce the overheads of checkpointing. Moreover, the initial timing of the checkpoints has been also inserted as suggested by the state-of-the-art non-uniform checkpointing technique [12]. Although the employed non-uniform checkpointing has a lower power overhead compared to the uniform one, TDP violations might be observed. The reason is that checkpointing requires to re-execute some parts of the tasks. That might lead to simultaneous execution of task parts with high power consumption, which eventually might result in a peak power beyond thermal design power (TDP). In line 2, the algorithm initializes a list to the total power consumption of the chip at each time point according to the initial scheduling. In line 3, the algorithm finds time points where TDP is violated and puts it in the list *Violations*. Each entry of the list has three parameters: the time point of the violation, the period of violation, and the power consumption of the TDP violation. Next, the algorithm iterates until TDP is met at all violation points (lines 4-24) by means of adjusting checkpoint timing and exploiting available slack time for delaying task execution, as explained in the following subsections.

4.2 Adjusting Checkpoint Timing

The goal of adjusting checkpoint timing is to have more checkpoints at the time points of TDP violations, because during the period of the checkpoint the task execution will be suspended, and thus the total power consumption is significantly reduced. Executing the checkpoint has power overhead, but it is much less than the power consumption of the tasks, as will be shown later in Section 5.

To achieve this goal, our proposed method (PPAC) will try to shift the available checkpoints, that are originally inserted after the violation time point, to the violation time point. Before shifting the checkpoints, it is necessary to check if shifting the available checkpoints is enough to satisfy TDP or not (see the condition in line 7 of Algorithm 1). Note that the variables $cur.vio.p$, $T_i.CK\_P$, and $T_i.P$ represent the current amount of power violation of the time point, the power overhead of the checkpoint of T_i, and the power consumption of T_i, respectively. If this condition is satisfied, then PPAC will shift only the minimum number of checkpoints that lead to satisfying TDP, if they are shifted to the violation time point (line 8). In particular, PPAC will try first to find a task (T_i) that has a minimum power consumption that is equal or greater than ($cur.vio.p+T_i.CK\_P$). If found, the checkpoint of that task will be shifted, because that will satisfy TDP by shifting one checkpoint. If such a task is not available, PPAC will find a list of checkpoints of multiple tasks that satisfies the TDP constraint with a minimum number of checkpoints. After finding the required list of checkpoints, these checkpoints will be shifted (line 9), i.e., their time will be set to $cur.vio.t$. Then, the algorithm updates the power consumption list (line 10).

If the available checkpoints are not enough for meeting TDP, i.e., the condition shown in line 7 is not satisfied, the algorithm needs to use the second function of the proposed method, which is explained in the following subsection.

In conclusion, we should compute the total power consumption of the chip at each time slot to check the TDP constraint, and hence, the location of some checkpoints should be changed. Moreover, if there are not enough checkpoints for meeting TDP, shifting the tasks can be meet the TDP constraint. Therefore, the maximum computational overheads of the proposed method are changing the locations of some checkpoints and shifting some tasks.

4.3 Delaying Task Execution

Intuitively, delaying the execution of a task at the violation time point will significantly reduce the total power consumption and that will help to satisfy TDP constraint. However, this might lead to missing the deadline of the frame. Therefore, PPAC will delay only the tasks executing on the cores that have slack time equal to or greater than the violation period. Hence, in line 12, the algorithm finds a set of cores C that have a slack time greater than or equal to $cur.vio.period$. In line 13, the algorithm assigns the tasks which are in C and executing at $cur.vio.period$ to the list Ψ_1. This list of tasks contains only the possible tasks to delay without violating timing constraints. Generating this list will significantly reduce the search space when the algo-

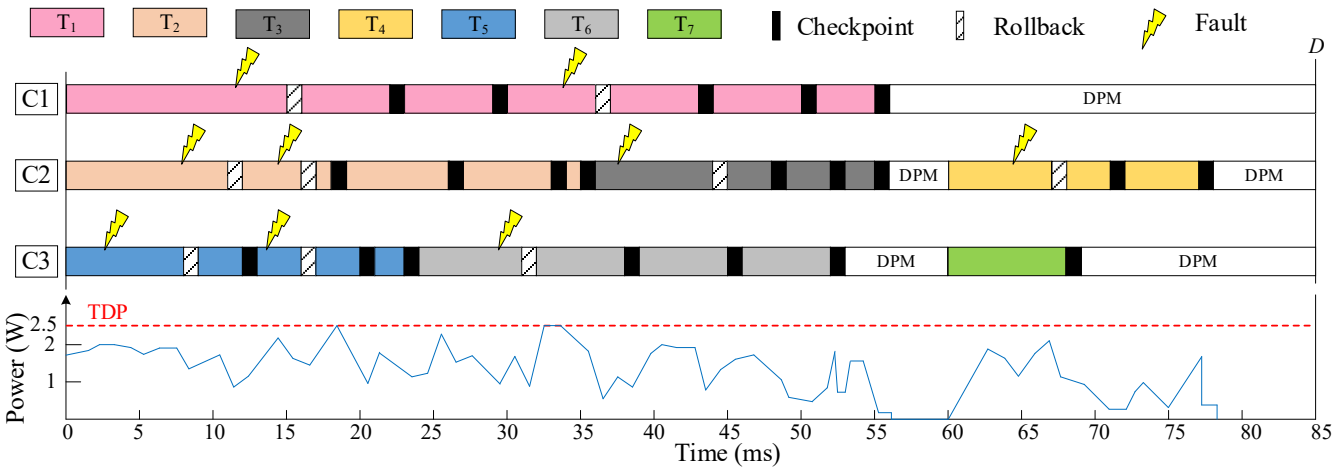

Fig. 3. The online part of the PPAC re-scheduling in the worst-case fault scenario.

rithm needs to select the best tasks to delay with the purpose of satisfying TDP.

Note that TDP might be satisfied by delaying only a subset of these tasks and shifting some of the available checkpoints. Therefore, our PPAC will find all the combinations between the list of the potential tasks to delay and the list of the available checkpoints that satisfy TDP (line 15). To do this, the algorithm finds the minimum number of checkpoints that need to be shifted and the minimum number of tasks that need to be delayed, to satisfy TDP. Afterward, the selected tasks will be delayed by *cur.vio.period* and the selected checkpoints will be shifted to the *cur.vio.t*. If no combination satisfies TDP, the algorithm returns *infeasible* and we cannot meet the TDP constraint for the considered time slot. Finally, the algorithm updates the power consumption list and violations in lines 22 and 23, respectively.

4.4 Runtime Opportunities

In fault-tolerant techniques for real-time systems, the worst-case scenario needs to be considered at design time while scheduling the tasks. The worst-case scenario assumes that the k faults will occur. As aforementioned, the checkpointing technique adds time overhead to the task execution, since it requires to re-execute some parts of the tasks. However, at runtime, it is possible that no fault occurs, and thereby long slack time will be available on the cores[1].

Therefore, we propose to exploit the available slack time resulting after the actual execution of the tasks at runtime, to further reduce power/energy consumption. That, in turn, will help to increase the life-time of the embedded systems that are battery-based [8][12]. To achieve this, we employ a runtime control unit. This unit needs first to monitor the accuracy of the task execution, and when no fault occurs, it cancels the execution of the replicated parts of the tasks (that have been added by the checkpointing technique to tolerate faults), resulting in slack times in the schedule. The online control unit puts the system during those resulting slack times into a low power state (apply DPM). The time overhead of DPM is typically represented by the *break to sleep time* which is about 1ms [2][8]. The power overhead of this technique is about 500uW which is negligible compared to the power consumption of tasks which is about 1W [25]. In our technique, if the slack time is more than the *break to sleep time*, the core is set to sleep mode. Hence, by applying DPM during slack times, our technique can achieve further power and energy reductions at runtime.

4.5 Illustrative Example

To show how the PPAC technique works in runtime, we use Fig. 3 which is the online part of the motivational example (explained in subsection 1.1). In Fig. 3, the actual-case fault scenario is shown. In this scenario, we consider that two, one, and zero faults occur on the tasks $\{T_1, T_2, T_5\}$, $\{T_3, T_4, T_6\}$, and T_7 respectively. Note that checkpoints are represented by black rectangles, and rollback recovery processes are shown as white diagonal patterned rectangles. In the checkpointing technique, when a fault occurs, the system rolls back to the most recent checkpoint and re-executes the part of the task that is executed during the last checkpoint. In this figure, by applying PPAC, TDP is met at all time points by shifting five checkpoints at design time (see Fig. 1d).

At runtime, the online control unit monitors the accuracy of the task execution. When no fault occurs, it cancels the execution of the replicated parts of the tasks that have been added by the checkpointing technique to tolerate faults. As soon as a task finishes successfully, the online control unit applies DPM. To do this, we exploit the slack times that create after that each task completes its execution without faults to further reduce power/energy consumption through DPM. For example, when only one fault occurs at the first part of T_6 and other its parts are executed successfully, a slack time at the time slot [54ms, 60ms] on C3 is created. Since there is no task being executing at the

[1] A comparison between faulty and fault-free scenarios is shown in [12].

mentioned time slot on C3, DPM is applied on C3 to further reduce power consumption, and so on. It is worthy to mention that this figure shows the actual fault rate but in Fig. 1, we showed the worst-case fault scenarios. Moreover, we have changed the locations of the checkpoints for meeting TDP. Therefore, there are some instances in this figure where after a fault occurs, the time required to complete the task after rollback is much lower than the time from the checkpoint up to the fault.

5 Experimental Evaluation

5.1 Setup

In order to evaluate our PPAC technique, we conducted experiments on various real-life applications of MiBench Benchmark [15] executed on a cycle-accurate system-level simulator (gem5 [16]), which is instrumented with precise power and performance characteristics for ARM Cortex-A7 [20] and an emerging non-volatile memory (NVM) technology. The details of the processor and memory configuration are summarized in Table 2. The applications and processor characteristics were obtained through gem5 [16], and McPAT [17]. We have deployed various applications of an embedded MiBench Benchmark suite [15] (listed in Table 3), as it is commonly used by state-of-the-art real-time scheduling techniques [5][12]. These applications are single-threaded because real-time systems require predictability [44][45]. In order to consider a wide range of applications, we chose the applications from all program groups of MiBench (i.e., automotive, consumer, network, office, security, and telecomm). Additionally, the applications were selected such that they introduce a variety of values for the simulation parameters, i.e., peak power consumption, average power consumption, worst-case execution time, energy consumption, and checkpointing overheads (see Table 3). As we mentioned in the system and application model, we have considered n frame-based hard real-time tasks. Each task is a benchmark from MiBench. Note that in this model tasks have the same release time (arrival time) and they do not have a dependency between them [1][2][3]. Therefore, each task executed individually and then the results are provided for the system. The maximum length of the power vector for each frame is equal to the deadline of the frame. We calculate these values in the design time and we do not need any computational overhead for runtime. It is worthy to mention that the granularity of power measurement is millisecond of the time frame and the unit is Watt. With respect to the time and power characteristics of the checkpoint memory, we assumed that the system uses a non-volatile memory (NVM) as stable storage to hold checkpoints.

Table 2. The details of processor and memory configuration

Core Type	Instruction Set Architecture	Machine Type	Core Voltage	Core Freq.	L1d Cache Size	L2 Cache Size	Ram Size
ARM	ARMv7A	Out of Order	1.318V	2GHz	32KB	512KB	512MB

For the evaluation of our proposed method, we have created a scheduler that maps and schedules the tasks by considering the proposed checkpointing mechanism in multicore embedded systems. To provide checkpointing overheads for simulation, at first, we determined the checkpoint size for each benchmark application. We considered full-scale checkpointing where the entire address space used by an application is written to the memory during each checkpoint. We selected ReRAM because it presents a relatively better time and power characteristics compared to the other types of memory [12]. Moreover, we assume that the system is equipped with Argus [14] as a fault-detection mechanism and the time overhead for fault detection (~ 4% of the worst-case execution time of each task [12]) is added to the checkpointing overheads. For finding the accurate time, the power and memory overhead of the checkpoints for each application we employ CRIU [19] (Checkpoint/Restore in Userspace), the most powerful software in Linux that is used for checkpointing in many applications such as ducker [19]. For the first time, we have implemented CRIU in gem5 with an ARM processor. To do this, we have created a new image of Linux and the new kernel that supports checkpoints in the userspace. Indeed, we have compiled a new version of the Linux kernel for gem5 and have added some functions and modifications to the kernel to enable

Table 3. The characteristics of the Benchmark Applications

	Patricia	GSM	FFT	CRC32	dijkstra	Blowfish	Susan	QSORT	Bitcount
Execution time (ms)	253	473	233	240	88	272	108	132	196
Average power consumption (w)	1.026	1.021	1.036	1.125	1.079	1.081	1.038	1.031	0.984
Peak power consumption (w)	2.428	4.749	3.960	4.532	2.622	3.838	1.124	1.053	5.792
Energy consumption (J)	0.258	0.482	0.240	0.269	0.939	0.294	0.112	0.136	0.192
Average power consumption of checkpointing in processor (W)	0.024	0.0172	0.047	0.052	0.286	0.0197	0.129	0.021	0.015
Peak power consumption of checkpointing in processor (W)	0.073	0.427	0.088	0.165	0.946	0.124	0.166	0.398	0.099
Memory overhead of checkpoint (KByte)	4336	212	732	204	260	216	440	1864	96
Checkpointing time (ms)	39.2	20.3	14.1	10.9	5.2	12.3	7.2	17.7	8.4
Checkpointing time overhead	15.5%	4.2%	6%	4.5%	5.9%	4.5%	6.6%	13.4%	4.3%

Checkpointing time and power values are obtained for the ReRAM technology.

checkpointing so that we can get the checkpoint in the gem5 environment. Then, we get the results of the gem5 environment and send them to our scheduler. Note that in order to run CRIU in gem5 with armv7 architecture, first, we cross-compile CRIU for arm32, then we compile a new kernel with functions that are required for running CRIU and take a checkpoint following the instruction in [48]. As the last step, we created an image with Ubuntu core 18.04.2 (Bionic Beaver) for the aforementioned kernel. The kernel and image ran on the VExpress_GEM5_V1 machine type and we managed to get checkpoints from our benchmarks.

In order to generate task sets, we produce a set of tasks that have different worst-case execution times. We set the deadlines of the task randomly, to have slack times between 10% to 80%. These various scenarios of slack times result in different task utilization on the target cores. We categorize the resulting set of tasks into various utilization scenarios; i.e., 0.1, 0.2, …, 0.8, in order to evaluate our technique and the comparison candidates for those scenarios, as shown in the figures in this section. Note that, these utilization scenarios represent the task utilization of the core before applying the checkpointing technique. We compared our PPAC technique with three state-of-the-art checkpointing techniques in terms of feasibility, schedulability, and peak power consumption. The comparison candidates are:

- **Standard uniform checkpointing (StaU)** [13]: This technique determines the uniform checkpoint intervals for each application so that k faults are tolerated and the timing constraint of the applications is satisfied.
- **Non-uniform checkpointing (NonU)** [12]: This technique is an implementation of the non-uniform checkpointing presented in [12]. It exploits non-uniform checkpoint intervals to tolerate faults and reduces the overhead of checkpointing mechanism.
- **RAPM** [18]: This technique is proposed in [18]. RAPM uses a backup task for each faulty task to achieve fault tolerance (re-execution technique).

We compared PPAC with the three mentioned techniques for two following scenarios: *i*) the worst-case scenario when all the faults occur and the system consumes the maximum possible power (Section 5.2) and *ii*) the actual-case (realistic) scenario when the faults occur based on Eq. 3 (the realistic scenario in the nature) and the system consumes real power (Section 5.3).

5.1.1 Evaluation Metrics

The major evaluation metrics of the comparison candidates are (1) feasibility, (2) schedulability, (3) the percentage of TDP enforcement throughout the execution time, and (4) normalized peak power consumption to TDP. We define feasibility as the percentage of simulations that the TDP and deadline constraints are satisfied simultaneously. Indeed, in this paper, the meaning of feasibility is the percentage of meeting the deadline and the TDP constraint simultaneously.

In this section, we consider the *per-core utilization factor* (U) that is the fraction of core time spent in the execution of the task set. Since wc_i/D_i is the fraction of core time spent in executing task T_i, the utilization factor for n tasks is given by

$$U = \sum_{i=1}^{n} \frac{wc_i}{D_i} \quad (9)$$

The core utilization factor provides a measure of the computational load on the chip due to the task set. If this factor is less than 1 for each core, the maximum load for each core is performed. It is worthy to mention that all of the figures in this paper have a utilization less than 1.

5.2 The Worst-case Scenario

The worst-case scenario considers that the given number of faults k will occur, and thereby all replicated parts of the tasks and their checkpoints are executed in this scenario, which potentially leads to high power consumption. Therefore, it can be considered a good condition for comparing peak power reduction and feasibility. Note that since we generate a different set of tasks, we have different deadlines and utilizations. Each case of simulations was simulated 1000 times with different deadlines and utilizations, and the average results are reported.

Note that our evaluations in this subsection consist of two sets of simulations. In the first set, we evaluate the techniques along with different number of faults. In the second set, various system utilizations are considered.

5.2.1 The impact of different number of faults

In this subsection, we show the impact of different *number*

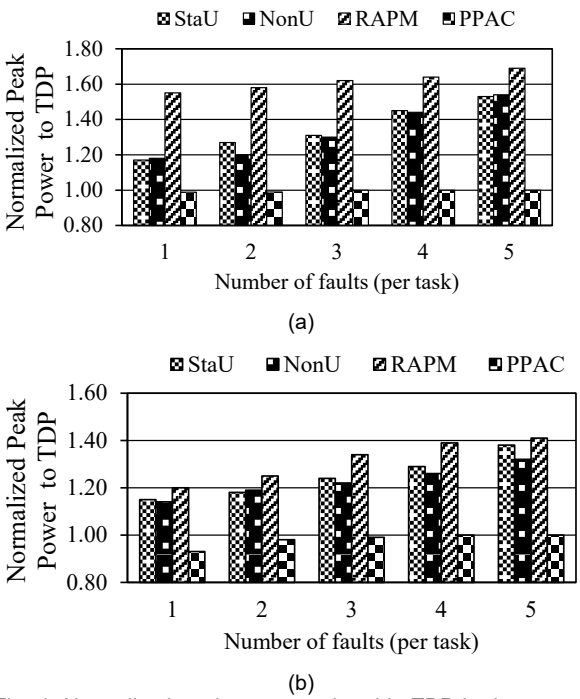

Fig. 4. Normalized peak power to the chip TDP in the worst-case scenario by considering different k faults occurrence; (a) # of cores = 4, (b) # of cores = 8.

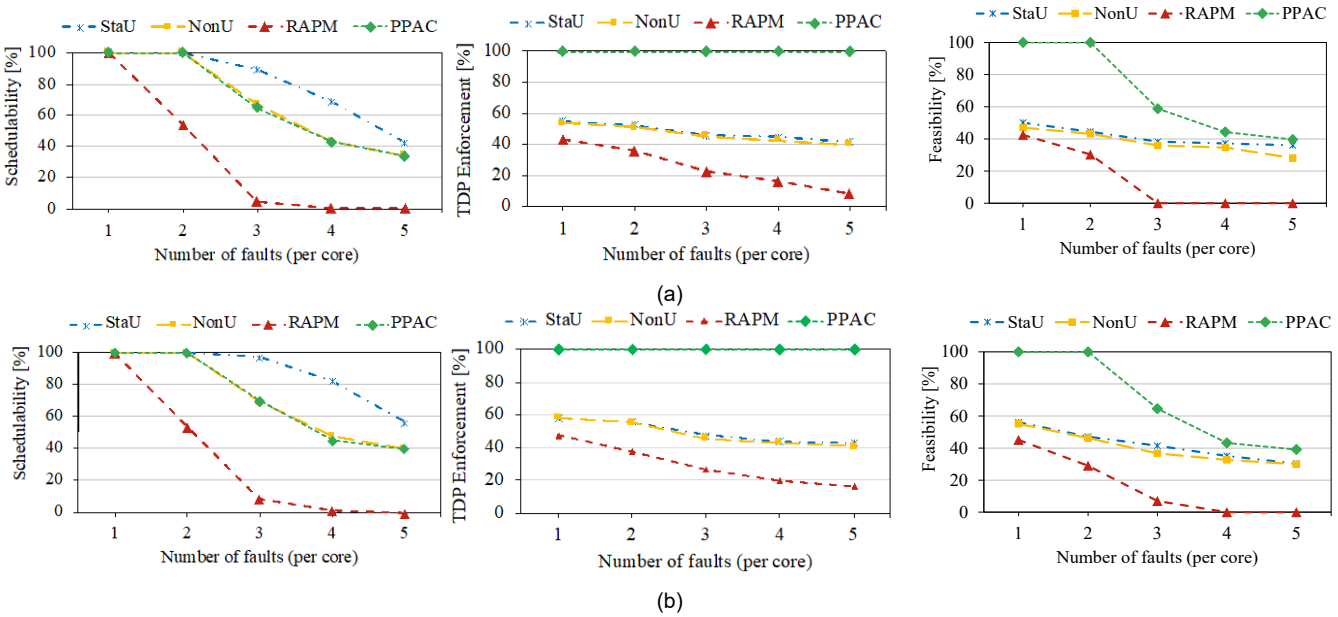

Fig. 5. The impact of different *k* faults occurrence in the worst-case scenario for random utilizations on schedulability, the percentage of TDP enforcement throughout the execution time, and feasibility; a) # of cores = 4, b) # of cores = 8.

of faults on the adopted evaluation metrics for all comparison candidates. Fig. 4 shows the normalized peak power to the chip TDP for a chip with 4 and 8 cores when characteristics of tasks are generated based on Table 3. It should be noted that the value of TDP for a chip with 4 cores and 8 cores is 8W and 16W, respectively. It can be seen from Fig. 4 that PPAC never violates TDP while the state-of-the-art techniques violate it because the state-of-the-art techniques do not consider it. Furthermore, Fig. 4 indicates that with the higher numbers of cores RAPM can meet TDP, but with higher fault occurrences and a lower number of cores, it cannot meet TDP. Both StaU and NonU mechanisms are performing poorly regarding peak power reduction. It should be noted that when the number of the cores increases (Fig. 4a to Fig. 4b), the slack times on the cores increase, and then TDP and deadline violations decrease. Our PPAC technique reduces the peak power consumption by 34% and up to 62% compared to the state-of-the-art techniques.

Fig. 5 shows the effect of increasing the number of fault occurrences on the schedulability, the percentage of TDP enforcement throughout the execution time, and feasibility for different mechanisms on a chip with 4 and 8 cores. Since we shift some tasks to the next time slots to manage peak power consumption, we need more time slots for meeting the deadlines. In this paper, we have focused on meeting TDP and timing constraints simultaneously. Therefore, our proposed technique incurs more time overhead as compared to other techniques that consider fewer constraints, e.g., the reference [13]. It should be noted that for the feasibility comparison, as other techniques do not consider TDP, so they will definitely be worse. It may be good that we show whether the deadline violation will be increased when the proposed technique is used. One of the overheads of our proposed method is the required time for shifting the tasks. This is a consequence of having two constraints: 1) deadline, 2) TDP. When we have two constraints, regardless of what is the benchmark, we need some time overhead for meeting both of them.

As shown in the right side of Fig. 5, our PPAC has higher feasibility than all other comparison candidates for different *k* faults occurrence in the worst-case scenario for random utilizations when (a) # of cores = 4, and (b) # of cores = 8. The resulting feasibility of PPAC is 69%, while the StaU, NonU, and RAPM techniques are 41.6%, 38.9%, 15.4%, respectively. To provide a deeper insight, we additionally show both implicit metrics of feasibility; i.e., the

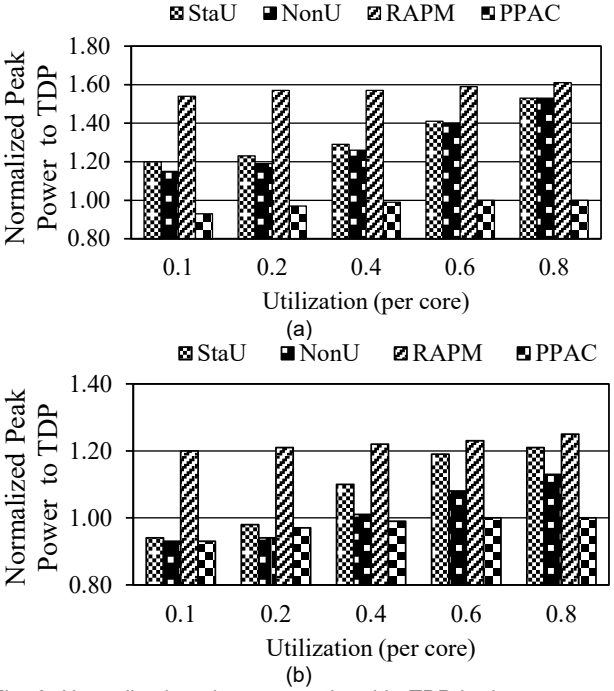

Fig. 6. Normalized peak power to the chip TDP in the worst-case scenario by considering various utilizations; (a) # of cores = 4, (b) # of cores = 8.

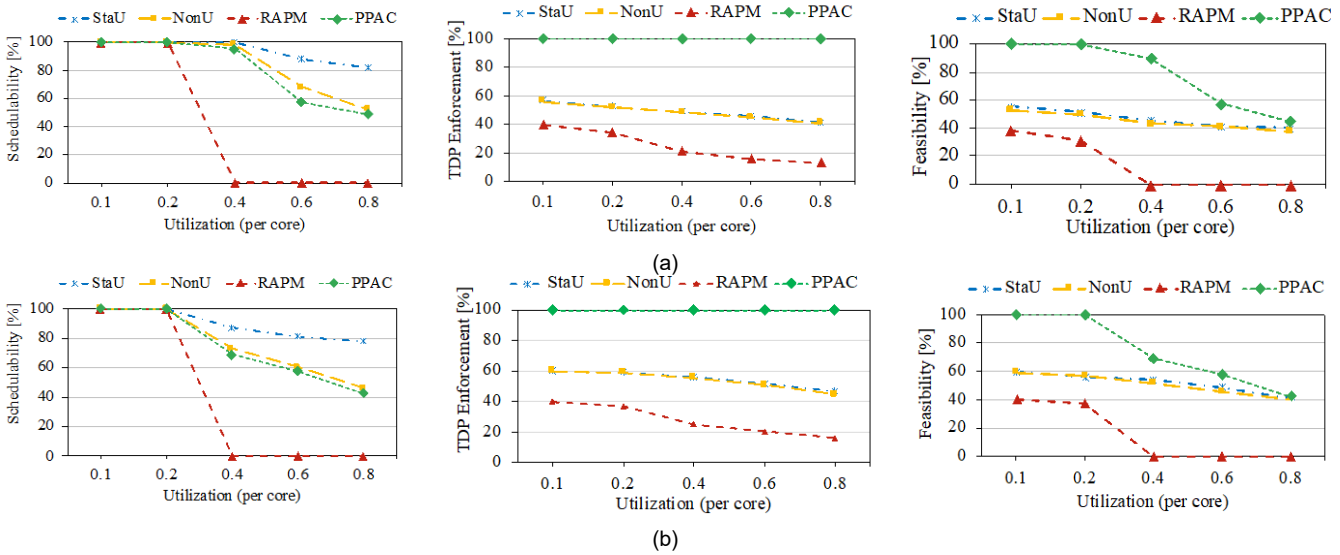

Fig. 7. The impact of various utilizations in the worst-case scenario on schedulability, the percentage of TDP enforcement throughout the execution time, and feasibility; a) # of cores = 4 and *k*=2, b) # of cores = 8 and *k*=3.

percentage of TDP enforcement throughout the execution time and the schedulability. As it can be observed from the charts in the middle of the figure, PPAC has always met TDP constraint, while the StaU, NonU, and RAPM techniques have met the TDP constraint by only 48.8%, 47.6%, 27.6%, respectively. However, there is a time overhead to be paid by PPAC for meeting the TDP. In particular, PPAC shifts some parts of the tasks to the next time slots to satisfy TDP constraints. This time overhead has led to reducing the schedulability of PPAC in some worst-case scenarios. The schedulability of the four comparison candidates; PPAC, StaU, NonU, and RAPM are 69.7%, 83.7%, 70.1%, 32.2%, respectively.

5.2.2 The impact of various utilizations

In this subsection, we show the impact of various utilizations on the adopted evaluation metrics. Fig. 6 shows the normalized peak power to the chip TDP for a chip with 4 and 8 cores. It can be seen from Fig. 6 that PPAC never violates TDP while the state-of-the-art techniques violate it. Our PPAC technique reduces the peak power consumption by 28% and up to 60.1% compared to the state-of-the-art techniques in various utilizations.

The original utilization of tasks without applying the checkpointing technique is varied between 0.1 to 0.8 for each core. To demonstrate this, we generated 1000 task sets from the tasks shown in Table 3 and repeated the simulations for several per-core utilization values ($U_{per\text{-}core}$= 0.1 to 0.8). Since the proposed approach never violates the two mentioned constraints, we show the frequency of meeting these constraints (as defined feasibility).

As shown in the right side of Fig. 7, our PPAC has higher feasibility than all other comparison candidates in scenario (a) # of cores = 4 and *k*=2, and (b) # of cores = 8 and *k*=3. Particularly, the resulting feasibility of PPAC is 76.1%, while the StaU, NonU, and RAPM techniques are 49.3%, 47.9%, 14.7%, respectively. As it can be observed from the charts in the middle of the figure, PPAC has always met TDP constraint, while the StaU, NonU, and RAPM techniques have met the TDP constraint by only 52%, 51.3%, 26.4%, respectively. The schedulability of the four comparison candidates; PPAC, StaU, NonU, and RAPM are 77.1%, 91.7%, 79.8%, 40%, respectively, for the worst-case scenario. However, for the realistic scenario, we can notice from Fig. 8 that our PPAC has much better schedulability, in average of 99.5%, while only the technique "NonU" has a higher schedulability with an average of 99.7%. This is however a negligible overhead to pay to satisfy power constraint in addition to timing constraint. Note that NonU uses non-uniform checkpointing without shifting the parts of the tasks to reduce the time overhead in the realistic scenario.

5.3 Realistic Scenario

In this case, we investigate the actual conditions (the realistic scenario in nature) where the actual fault rate is considered. To inject the faults, we have generated them using a Poisson process where the fault rate λ was modeled using Eq.3 under the parameters λ_0=10^{-4} faults/us [1]. It should be noted that these fault rates are much higher than real fault rates (e.g., 10^{-12} faults/us). Since, using fewer faults will require much number of fault injections to cover different parts of applications, which will require months of simulations to ensure high coverage, we used the high fault rates to evaluate our technique. Due to the stochastic nature of transient faults, the same application is executed for 1000 times and the average results are reported.

In this scenario, like the worst-case scenario, we show the schedulability, the percentage of TDP enforcement throughout the execution time, and the feasibility of the proposed method for different utilizations. Similar to the previous scenario, we consider that the original utilization of tasks without applying the checkpointing technique is varied between 0.1 to 0.8 for each core.

As shown in the right side of Fig. 8, our PPAC has higher feasibility than all other comparison candidates in the realistic scenario. The resulting feasibility of PPAC is 99%, while the StaU, NonU, and RAPM techniques are

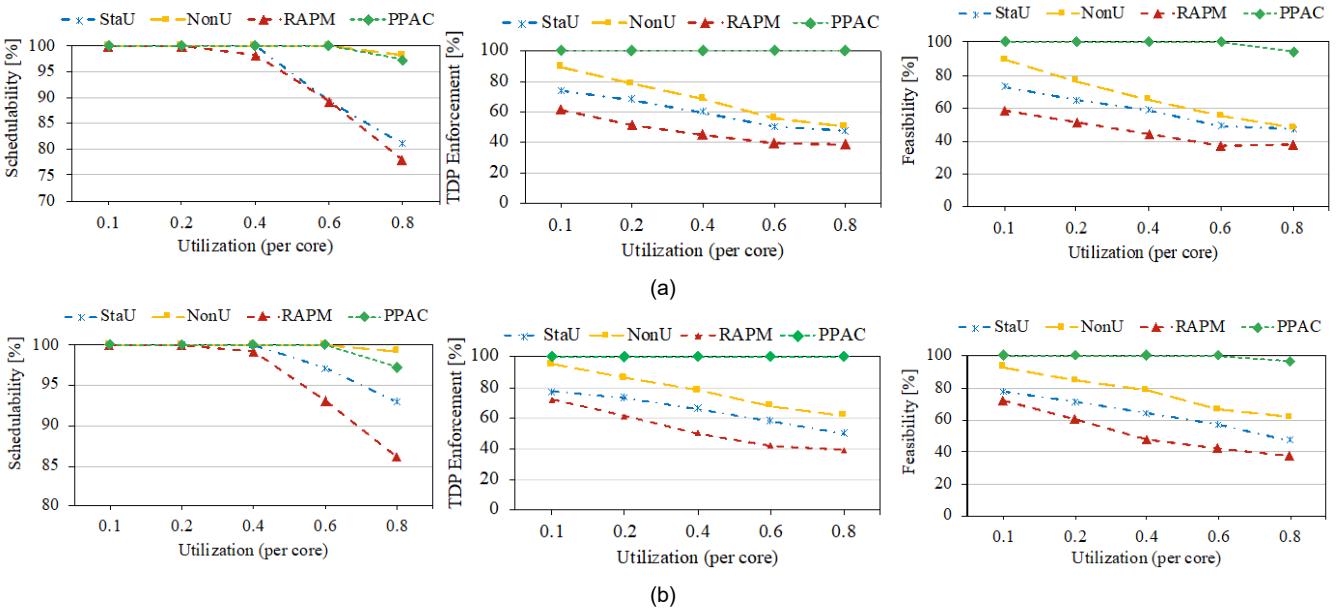

Fig. 8. The impact of various utilizations in the realistic scenario on schedulability, the percentage of TDP enforcement throughout the execution time, and feasibility; a) # of cores = 4 and k=2, b) # of cores = 8 and k=3.

61%, 71.8%, 48.9%, respectively. As it can be observed from the charts in the middle of the figure, PPAC has always met TDP constraint, while the StaU, NonU, and RAPM techniques have met the TDP constraint by only 62.3%, 73%, 49.9%, respectively. Moreover, the schedulability of the four comparison candidates; PPAC, StaU, NonU, and RAPM are 99.5%, 96%, 99.7%, 94.4%, respectively.

Moreover, Fig. 9 shows the normalized peak power to the chip TDP for a chip with 4 and 8 cores. It can be seen from Fig. 9 that Our PPAC technique reduces the peak power consumption by 27% and up to 53% compared to the state-of-the-art techniques in various utilizations.

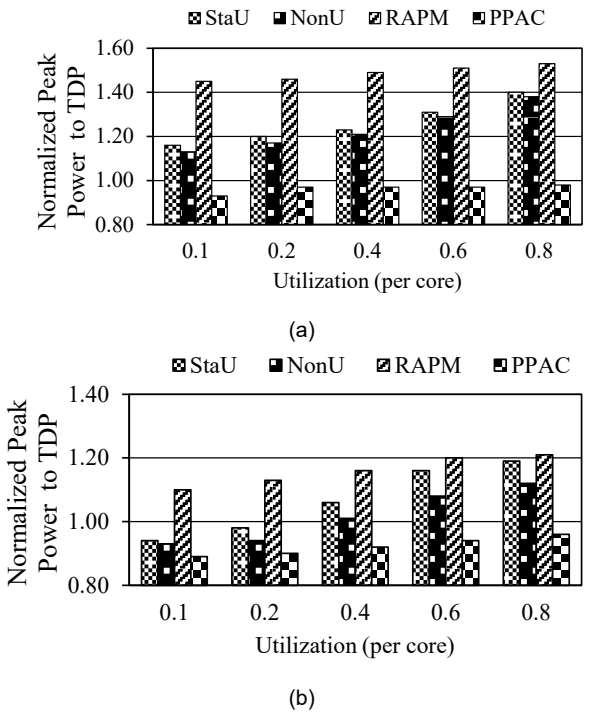

Fig. 9. Normalized peak power to the chip TDP in the worst-case scenario by considering different k faults occurrence; (a) # of cores = 4, (b) # of cores = 8.

In Fig. 10, we have evaluated the aforementioned methods on 2, 4 and 8 core systems with the mentioned fault rate. We executed up to 100 tasks on each core for 1000 task sets and reported the average results. As it is shown in the figure, our worst-case evaluation broadly matches the real-world scenario. By increasing the number of cores, the energy consumption for all four methods decreases because the utilization of the cores decreases. Although PPAC always keeps the peak power of the system below TDP constraint, other methods cannot guarantee meeting TDP constraint. The experiments show that PACC completely outperforms the two techniques (SptU and RAPM) from the energy consumption viewpoint. However, the energy consumption of PPAC is worse than the NonU technique because NonU is designed as a technique that manages energy consumption. The PPAC technique provides on average 17.28% (up to 61.1%) energy reduction as compared to the three mentioned techniques.

6 Related Work

At first, checkpointing was proposed for database systems where availability/reliability is the main criterion [33].

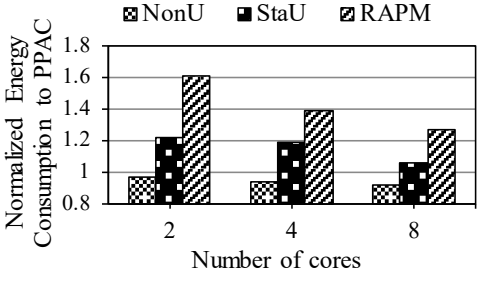

Fig. 10. Normalized energy consumption to PPAC in the actual-case scenario.

However, in real-time embedded systems, other requirements like high reliability and timeliness as well as low power consumption should be carefully considered [13]. Example applications of such real-time embedded systems where high reliability and low power consumption are needed include, but not limited to, medical care devices, avionics systems, control of chemical reactions, space and surveillance systems, and condition monitoring systems [12]. Although checkpointing with rollback recovery increases the execution time of the tasks in the absence of faults, it reduces the recovery time of faulty tasks. This is because there is no need to replicate or re-execute the whole task, and only the part of the task, which begins from the last safe checkpoint, is required to be executed again. Checkpointing is a fault-tolerant technique that can result in lower power and time overheads, if intelligently used, compared to re-execution, replication, TMR, standby-sparing, and etc. [12][25][28][29].

In [34], Pop et al. have exploited the combination of checkpointing with rollback recovery and active replication to tolerate transient faults in the hard real-time systems. Moreover, they have proposed global optimization of the number of checkpoints and have integrated checkpointing into an assignment and mapping optimization strategy. Zhu et al. in [35] have proposed a method that whenever the slack is not enough for re-execution of the tasks, checkpointing is employed as the fault-tolerant technique. Punnekkat et al. [36] have proposed a feasibility test for periodic and sporadic task sets which can be scheduled based on any fixed-priority preemptive scheduling under checkpointing fault-tolerant technique for uniprocessor systems to tolerate transient faults. They claim that the results are applicable to distributed multiprocessor systems where tasks are statically allocated to individual processors. Indeed, they have minimized the execution time of the tasks for equidistant checkpoints in the worst-case fault scenario. In [37], a checkpointing mechanism is presented to achieve higher lifetime reliability of the system. In [38], a checkpointing scheme is presented for ASIP-based embedded systems. Kwak et al. in [39] have proposed a checkpointing scheme that tolerates timing and control flow errors in hard real-time systems. However, these works do not consider the power overhead of adding the checkpoints. Some works try to maximize the probability of meeting deadlines while using probabilistic analysis [40][41]. However, for hard real-time systems, tolerating a given number of faults (up to k faults) should be guaranteed [10][12][30][42]. Melhem et al. in [13] have considered the interaction between checkpointing and average power management and proposed uniform and non-uniform checkpointing schemes for tolerating only one fault. The work in [12] has proposed a non-uniform checkpointing scheme with very low time and energy overheads to tolerate k faults for hard real-time systems. They use Dynamic Voltage Scaling (DVS) for reducing the average power consumption of the system.

As it is mentioned, none of the previous work in the context of real-time embedded systems did consider the power constraint for different checkpointing mechanisms. This paper presents for the first time, a peak-power-aware checkpointing (PPAC) technique that tolerates k faults in hard real-time embedded systems, while at the same time meets the power and timing constraints.

7 CONCLUSIONS

In this paper, a Peak-Power-Aware Checkpointing (PPAC) technique to achieve low-power fault tolerance for hard real-time systems has been proposed. According to the power profile of applications, PPAC determines the minimum number of non-uniform checkpoints that tolerate k faults in hard real-time embedded systems, while at the same time meets the power constraints. In the experiments, we verified the schedulability of the proposed technique and other techniques for each generated task set. The results show that PPAC meets the timing and TDP constraints on average by 80.38% while the other techniques meet the mentioned constraints on average by 31.9%. Moreover, our proposed technique provides up to 37.9% (on average by 23%) peak power reduction compared to state-of-the-art techniques.

REFERENCES

[1] S. Pagani, H. Khdr, 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. Comp.*, vol. 66, no. 1, pp. 147-162, 2017.

[2] 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 and Distr. Sys.*, vol. 30, no. 1, pp. 161-173, 1 Jan. 2019.

[3] H. Khdr, H. Amrouch, and J. Henkel, "Aging-Constrained Performance Optimization for Multi Cores," in *ACM/EDAC/IEEE 55rd Design Automation Conference (DAC)*, USA, pp. 24-28, 2018.

[4] D. K. Schroder and J. A. Babcock, "Negative bias temperature instability: Road to cross in deep submicron silicon semiconductor manufacturing," *Journal of Applied Physics*, pp. 1–18, 2003.

[5] 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," in *IEEE Trans. Parallel and Distr. Sys.*, vol. 30, no. 10, pp. 2338-2354, 2019.

[6] W. Munawar, H. Khdr, S. Pagani, M. Shafique, J. Chen, and J. Henkel, "Peak Power Management for Scheduling Real-time Tasks on Heterogeneous Many-Core Systems," *in 20th IEEE ICPADS,* Hsinchu, Taiwan, December 2014.

[7] B. Zhao, H. Aydin, and D. Zhu, "Enhanced reliability-aware power management through shared recovery technique," in *Proc. ICCAD*, pp. 63–70, 2009.

[8] M. A. Haque, H. Aydin and D. Zhu, "On Reliability Management of Energy-Aware Real-Time Systems Through Task Replication," *IEEE Trans. Parallel and Distr. Sys.*, vol. 28, no. 3, pp. 813-825, 2017.

[9] S. Rehman, A. Toma, F. Kriebel, M. Shafique, J. Chen, and J. Henkel, "Reliable Code Generation and Execution on Unreliable Hardware under Joint Functional and Timing Reliability Considerations," *RTAS*, pp. 273-282, 2013.

[10] K.H Kim and J. Kim, "An Adaptive DVS Checkpointing Scheme for Fixed-Priority Tasks with Reliability Constraints in Dependable Real-Time Embedded Systems," *ICESS*, pp. 560-571, May 14-16, 2007.

[11] G. Buttazzo, "Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications," New York, NY: Springer, 2011.

[12] M. Salehi, M. Khavari Tavana, S. Rehman, M. Shafique, A. Ejlali, and J. Henkel, "Two-State Checkpointing for Energy-Efficient Fault Tolerance in Hard Real-Time Systems," in *IEEE Trans. VLSI*, vol. 24, no. 7, pp. 2426-2437, 2016.

[13] R. Melhem, D. Mosse, and E. Elnozahy, "The interplay of power management and fault recovery in real-time systems," *IEEE Trans. Comput.*, vol. 53, no. 2, pp. 217-231, 2004.

[14] A. Meixner, M.E. Bauer, and D. Sorin, "Argus: Low-Cost, Comprehensive Error Detection in Simple Cores," *Proc. IEEE/ACM 40th Int'l Symp. MICRO,* pp. 210-222, Dec. 2007.

[15] M.R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown, "MiBench: A Free, Commercially Representative Embedded Benchmark Suite," *Proc. Fourth IEEE Ann. Workshop on Workload Characterization*, pp. 3-14, 2001.

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

[17] 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 *MICRO*, pp. 469–480, 2009.

[18] Y. Guo, D. Zhu, and H. Aydin, "Reliability-Aware Power Management for Parallel Real-Time Applications with Precedence Constraints," in *Proc. Int'l Green Computing Conf. and Work. (IGCC)*, pp.1-8, 2011.

[19] "Checkpoint/Restore In Userspace" [Online]. Available: https://criu.org/Main_Page

[20] P. Greenhalgh, "big.LITTLE processing with ARM Cortex-A15 & Cortex-A7," ARM Limited, White Paper, September 2011.

[21] M. Ansari, J. Saber-Latibari, M. Pasandideh, and A. Ejlali, "Simultaneous Management of Peak-Power and Reliability in Heterogeneous Multicore Embedded Systems," in *IEEE Trans. Parallel and Distr. Sys.*, vol. 31, no. 3, pp. 623-633, 2020.

[22] 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," in *IEEE Access*, vol. 8, pp. 142843-142857, 2020.

[23] M. Ansari, A. Yeganeh-Khaksar, S. Safari and A. Ejlali, "Peak-Power-Aware Energy Management for Periodic Real-Time Applications," in *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems,* vol. 39, no. 4, pp. 779-788, 2020.

[24] 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 and Distr. Sys.*, vol. 33, no. 7, pp. 1678-1694, 1 July 2022.

[25] A. Roy, H. Aydin and D. Zhu, "Energy-aware standby-sparing on heterogeneous multicore systems," *2017 54th ACM/EDAC/IEEE Design Automation Conference (DAC)*, Austin, TX, 2017, pp. 1-6.

[26] S. Safari *et al.*, "A Survey of Fault-Tolerance Techniques for Embedded Systems from the Perspective of Power, Energy, and Thermal Issues," in *IEEE Access*, vol. 10, pp. 12229-12251, 2022.

[27] H. Khdr, S. Pagani, E. Sousa, V. Lari, A. Pathania, F. Hannig, M. Shafique, J. Teich, and J. Henkel, "Power Density-Aware Resource Management for Heterogeneous Tiled Multicores," in *IEEE Trans. on Comp.*, vol. 66, no. 3, pp. 488-501, 1 March 2017.

[28] I. Koren and C. M. Krishna. Fault-tolerant systems. Morgan Kaufmann, 2010.

[29] Q. Han, M. Fan and G. Quan, "Energy minimization for fault tolerant real-time applications on multiprocessor platforms using checkpointing," *International Symposium on Low Power Electronics and Design (ISLPED)*, Beijing, 2013, pp. 76-81.

[30] Kim, K. Hoon, and J. Kim, "An adaptive DVS checkpointing scheme for fixed-priority tasks with reliability constraints in dependable real-time embedded systems," *International Conference on Embedded Software and Systems* (Springer), S. 560-571, 2007.

[31] J. Lee, B. Yun and K. G. Shin, "Reducing Peak Power Consumption in Multi-Core Systems without Violating Real-Time Constraints," *IEEE Transactions on Parallel and Distributed Systems,* vol. 25, no. 4, pp. 1024-1033, April 2014.

[32] H. Aydin and Qi Yang, "Energy-aware partitioning for multiprocessor real-time systems," Proceedings International Parallel and Distributed Processing Symposium, Nice, France, 2003.

[33] E. Gelenbe "On the optimum checkpoint interval," *Journal of the ACM (JACM)*, vol. 26, no.2, pp. 259-270, April 1979.

[34] P. Pop, V. Izosimov, P. Eles, and Z. Peng, "Design optimization of time- and cost-constrained fault-tolerant embedded systems with checkpointing and replication," *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, vol. 17, no. 3, pp. 389–402, 2009.

[35] D. Zhu, "Reliability-aware dynamic energy management in dependable embedded real-time systems," *ACM Trans. Embed. Comput. Syst. (TECS),* vol. 10, no. 2, 2010.

[36] S. Punnekkat, A. Burns, and R. Davis, "Analysis of checkpointing for real-time systems," Real-Time Syst., vol. 20, no. 1, pp. 83–102, 2001.

[37] M.I bin Bandan, S. Bhattacharjee, R.A. Shafik, D.K. Pradhan, and J. Mathew, "Lifetime Reliability-Aware Checkpointing Mechanism: Modelling and Analysis," *Proc. Int'l Symp. Electronic Syst. Design (ISED)*, pp. 128-132, 10-12 Dec. 2013.

[38] T. Li, R. Ragel, S. Parameswaran, "Reli: Hardware/software Checkpoint and Recovery scheme for embedded processors," *Proc. Design, Automation & Test in Europe Conf. & Exhibition (DATE'12)*, pp. 875-880, 12-16 March 2012.

[39] S. W. Kwak, B. J. Choi, and B. K. Kim, "An optimal checkpointing-strategy for real-time control systems under transient faults," *IEEE Trans. Reliab.*, vol. 50, no. 3, pp. 293–301, Sep. 2001.

[40] S. W. Kwak and J.-M. Yang, "Optimal Checkpoint Placement on Real-Time Tasks with Harmonic Periods," *J. of Computer Science and Technology*, vol. 27, no. 1, pp. 105-112, January 2012.

[41] Z. Li, L. Wang, S. Ren, and G. Quan, "Energy minimization for checkpointing-based approach to guaranteeing real-time systems reliability," *Proc. IEEE 16th Int'l Symp. Object/Component/Service-Oriented Real-Time Distrib. Computing (ISORC)*, pp. 1-8, 19-21 June 2013.

[42] Y. Zhang and K. Chakrabarty, "A unified approach for fault tolerance and dynamic power management in fixed-priority real-time embedded systems," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 25, no. 1, pp. 111-125, Jan. 2006.

[43] Z. Chen, D. Stamoulis and D. Marculescu, "Profit: Priority and Power/Performance Optimization for Many-Core Systems," in *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 10, pp. 2064-2075, Oct. 2018.

[44] H. Kopetz, "Real-Time Systems: Design Principles for Distributed Embedded Applications," in Springer US, pp I-378, 2011.

[45] P. Marwedel "Embedded System Design," in Springer US, 2006.

[46] M. Ansari, S. Safari, S. Yari-Karin, P. Gohari-Nazari, H. Khdr, M. Shafique, J. Henkel, and A. Ejlali, "Thermal-Aware Standby-Sparing Technique in Hetereogeneous Real-Time Embedded Systems," *IEEE Transactions on Emerging Topics in Computing*, 2021.

[47] Y. Zhang and K. Chakrabarty, "Dynamic adaptation for fault tolerance and power management in embedded real-time systems," *ACM Trans. Embed. Comput. Syst.*, vol. 3, no. 2, pp. 336–360, May 2004.

[48] [Online]. Available: https://criu.org/Linux_kernel

Mohsen Ansari received his Ph.D. degree in computer engineering from the Sharif University of Technology, Tehran, Iran, in 2021. He is currently an assistant professor of computer engineering at the Sharif University of Technology, Tehran, Iran. He was a visiting researcher in the Chair for Embedded Systems (CES), Karlsruhe Institute of Technology (KIT), Germany, from 2019 to 2021. Also, he was a postdoctoral researcher and a research group leader of Embedded Systems Research Laboratory (ESR-LAB), and a lecturer at the department of computer engineering, Sharif University of Technology. His research interests include Cyber-Physical and Hybrid systems with a focus on dependability/reliability.

Sepideh Safari received a Ph.D. degree in computer engineering from Sharif University of Technology, Tehran, Iran, in 2021. She was a visiting researcher in the Chair for Embedded Systems (CES), Karlsruhe Institute of Technology (KIT), Germany, from 2019 to 2021. She is now a postdoctoral researcher at Institute for Research in Fundamental Sciences (IPM), Tehran Iran. Her research interests include the low-power design of cyber-physical systems, energy management in fault-tolerant embedded systems, and multi-/many-core systems with a focus on dependability/reliability.

Heba Khdr is a postdoctoral researcher and a group leader at the Chair for Embedded Systems (CES) at Karlsruhe Institute of Technology (KIT) in Germany. She received her Ph.D. (Dr.-Ing.) in Computer Science from Karlsruhe Institute of Technology (KIT) in 2018.

In 2005, she received her Diploma in Informatics Engineering from Aleppo University in Syria with an excellent grade and the first rank. From 2005 until 2007 she worked as a software engineer in the industry sector in Syria. She worked as an assistant at Aleppo University from 2008 until 2010. In 2011 she did an equivalent master thesis at KIT.

Her research interests are thermal management and resource management in multi- and many-core systems. In 2012 she received Research Student Award from KIT. She received Best Paper Award from IEEE/ACM International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS) in 2014, and four HiPEAC paper awards.

Pourya Gohari-Nazari received the MSc degree in computer engineering from the Sharif University of Technology, Tehran, Iran, and the BSc degree in computer engineering from the University of Isfahan. His research interests include thermal management in many-core systems and embedded systems with a focus on low-power and reliability.

Jörg Henkel (M'95-SM'01-F'15) 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. His research work is focused on co-design for embedded hardware/software systems with respect to power, thermal, and reliability aspects. Dr. Henkel has received six best paper awards from, among others, ICCAD, ESWeek, and DATE. He served as the Editor-in-Chief for the ACM TECS and IEEE Design&Test. He is/has been an Associate Editor for major ACM and IEEE journals. He was a General Chair ICCAD, ESWeek, etc., and serves as a Steering Committee chair/member for leading conferences and journals. He coordinates the DFG Program SPP 1500 "Dependable Embedded Systems" and is a site coordinator of the DFG-TR89 collaborative research center on "Invasive Computing." He is the Chairman of the IEEE Computer Society, Germany Chapter, and a Fellow of the IEEE.

Alireza Ejlali is an Associate Professor of Computer Engineering at Sharif University of Technology, Tehran, Iran. He received a Ph.D. degree in computer engineering from Sharif University of Technology in 2006. From 2005 to 2006, he was a visiting researcher in the Electronic Systems Design Group, University of Southampton, UK. In 2006 he joined Sharif University of Technology as a faculty member in the department of computer engineering and from 2011 to 2015, he was the director of Computer Architecture Group in this department. He is now the director of Embedded Systems Research Laboratory (ESR-LAB) and the head of the department of computer engineering, Sharif University of Technology. His research interests include low power design, fault tolerance, real-time embedded systems, and Internet of Things (IoT).

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