Peak-Power-Aware Energy Management for Periodic Real-Time Applications

Mohsen Ansari, Amir Yeganeh-Khaksar, Sepideh Safari, and Alireza Ejlali

***Abstract*— Two main objectives in designing real-time embedded systems are high reliability and low power consumption. Hardware replication (e.g. standby-sparing) can provide high reliability while keeping the power consumption under control. In this paper, we consider a standby-sparing system where the main tasks on primary cores are scheduled by our proposed PPA-EDF policy while the backup tasks on spare cores are scheduled by our proposed PPA-EDL policy to meet the chip TDP constraint. These policies provide the best opportunity to shift the task executions as much as possible to minimize execution overlaps between main and backup tasks that consume high power consumption. Since TDP is the maximum amount of power generated by a chip that the cooling component is designed to dissipate under any workload, the total power consumption should not be higher than the TDP constraint. When a task finishes successfully a larger portion of its corresponding copy task can be canceled, resulting in a significant amount of peak/average power reduction. To achieve further peak/average power reduction, we use Dynamic Voltage and Frequency Scaling (DVFS) and Dynamic Power Management (DPM). The main reason of using DPM is that, once the first copy of each task has finished successfully, its corresponding copy task is terminated, and if there is no more task for execution, the core goes to a low-power mode. We evaluated our scheme under various system configurations. Experiments show that our scheme provides up to 47.6% (on average by 28.2%) peak power reduction compared to four state-of-the-art techniques.**

***Index Terms* – Peak Power Consumption, Real-time Embedded Systems, Thermal Design Power, Scheduling.**

I. INTRODUCTION

Technology scaling continues to allow more transistors to be integrated onto a multicore chip while power consumption increasingly constraints the design of multicore embedded systems [1][2][3][4][5][43]. As well, the scaling of feature size raises the susceptibility of systems to transient faults [3][8][9][10][11][12][13][41]. Task replication is a well-established technique to achieve high reliability against transient faults. Despite the huge potential for task replication, due to the Thermal Design Power (TDP) constraint, embedded systems designers face a challenge in using multicore platforms [1][2][4][6][7]**Error! Reference source not found.**. TDP is considered as the highest sustainable power that a chip can dissipate before being forced to exploit a performance throttling mechanisms, e.g. Dynamic Thermal Management [1]. The heat-sink and cooling units for a chip are designed based on the chip TDP characteristics. If the peak power consumption of a chip violates its TDP, it automatically restarts or significantly reduces its performance to prevent a permanent damage. Therefore, in the embedded systems, power consumption is an important design concern [1][2]. Since the continuous power density increments along with technology scaling, increasing power densities have led to violating the chip TDP and making the thermal problems [40]. Generally, low power consumption and high reliability are the most important metrics in hard real-time embedded systems [12][13][14][15][16][23]. To achieve the high reliability, most of the studies had used fault-tolerance techniques. Associated redundancy brings a number of penalties: increase in weight, size, cost and power consumption. Consequently, in multicore embedded systems, increased power densities have introduced the so-called Dark-Silicon problem [1][42]. As a result of this problem, a significant percentage of the cores in a multicore system cannot be concurrently active [4]. Increasing the integration degree along with using fault-tolerance techniques can increase the power consumption and raise the peak power which can lead to violating the TDP constraint. In order to contrast with the TDP constraint, some solutions like heat-sink and chip's cooling are proposed. However, due to their negative effects on the system reliability, these solutions are not used in real-time embedded systems. Therefore, peak power management (or minimization) is an efficient way to meet the TDP constraint which prevents the system from producing high heat and temperature. Using TDP to define the power constraint of a system can be very pessimistic. However, one main reason to the widespread use of TDP, despite the fact that it is pessimistic, is that it is easy to check and easy to be handled in thermal management.

In multicore embedded systems, reliability is one of the main design objectives that is subjected to different types of faults [3][16][19][35][29]. The examples of these systems are medical care devices, avionics systems, control of chemical reactions, and surveillance systems [28]. Transient faults are often induced by electromagnetic interference and cosmic ray radiations and will disappear after a short time interval [11][20][21]. Restoring the system state and repeating the computation are a common approach to deal with the transient faults [20]. Multicore systems have an inherent redundancy which provides opportunities to implement various redundancy-based fault-tolerant techniques. The other reason that affects the system reliability is a violation of the chip TDP. When TDP is violated (due to increase peak power), some cores may become reset (inactive) and as a result, the system reliability will be reduced. Thus, the occurrence of peak power must be reduced in fault-tolerant

Manuscript received July 12, 2018; revised Sep. 28, 2018 and Feb. 12, 2019; accepted February 13, 2019. Date of publication M D, Y; date of current version M D, Y. This paper was recommended by Associate Editor C.-L. Yang. (Corresponding author: Alireza Ejlali)

The authors are with the Department of Computer Engineering, Sharif University of Technology, Tehran 14588, Iran (e-mails: {mansari; ayeganeh; ssafari}@ce.sharif.edu; ejlali@sharif.edu).

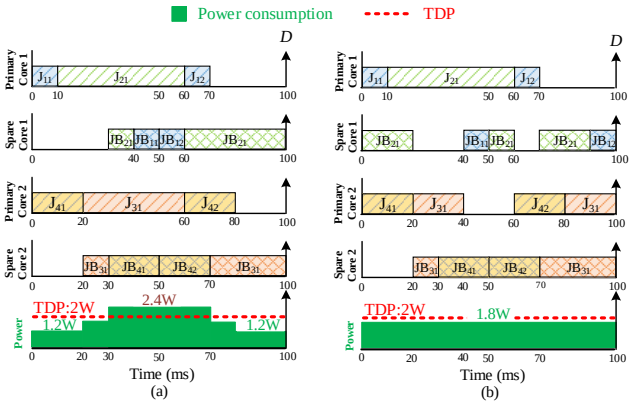


Fig. 1. A motivational example of peak power problem. a) Delayed execution of backup tasks[35], b) Peak-power-aware task scheduling.

embedded systems. The purpose of this paper is to reduce energy consumption while keeping the peak power consumption below the chip TDP at design time in fault-tolerant real-time embedded systems without violating any timing constraints. To achieve the mentioned purpose, we rely on scheduling tasks based on power profiles.

Motivational Example: Let us consider a 4-core chip with 2W of TDP that executes four tasks T_1, T_2, T_3 and T_4. We assume that the tasks arrive at time t=0 and have a period P_1=50ms, P_2=100ms, P_3=100ms and P_4=50ms. Therefore, the hyperperiod of the tasks is equal to h=100ms. Also, let us assume that the execution time of T_1, T_2, T_3 and T_4 are 10ms, 50ms, 40ms and 20ms, respectively. We also assume that each task consumes 0.6W of power throughout its execution and after finishing the task, the underlying core goes into sleep mode and consumes no power. Here, for ease of presentation, we temporarily assume that the tasks' peak power is equal to the task's average power. In the rest of this paper, when we present our technique we consider that the power consumption varies during a task execution and different tasks have different power traces. In this paper, we exploit task replication to achieve fault-tolerance and assume that any core can be dynamically coupled to another core to form a standby-sparing subsystem. Each standby-sparing subsystem executes two primary tasks and their backup tasks. Fig. 1a shows an execution scenario for these tasks where the primary tasks start as soon as possible and the backup tasks start as late as possible, hoping that the primary tasks will finish successfully and the backup tasks will be dropped to avoid excessive power consumption [35]. Since they do not consider the peak power consumption, they may result in violating the TDP constraint. This case is shown in Fig. 1a, wherein the time interval 30ms to 70ms all the four cores are active at the same time and hence the chip total power consumption is 2.4W that is higher than the chip TDP (i.e. 2W). Fig. 1b shows a possible execution scenario for the tasks that does not violate the TDP constraint. In this scenario, the first job of the backup task B_2 (JB_{21}) and the first job of the backup task B_3 (JB_{31}) are divided into three parts and two parts, respectively. Then, the other tasks and the parts of JB_{21} and JB_{31} are scheduled such that at any time instant at most three cores are active. Therefore, since each core consumes 0.6W, the total chip power consumption is less than or equal to 1.8W, and hence, the chip TDP is met.

Contribution: The main contributions of this paper are:

- A peak-power-aware energy management scheme that is conducted at offline phase through our proposed scheduling algorithm for the standby-sparing technique.
- Proposing two specific scheduling policies to manage energy and peak power consumption in the worst-case and actual-case fault scenarios.
- Proposing an online technique to achieve further reduction in energy/peak-power consumption through exploiting dynamic slacks.

Organization: In order to evaluate the effectiveness of the proposed method, we compared our scheme with four state-of-the-art techniques. The rest of this paper is formed as follows. In section II we review the related work. Section III presents our system model. In section IV, we present the details of our solution. The experimental results are shown in section V and we conclude the paper in section VI.

II. Related Work

A. Peak Power Reduction

Some related works [2], [18], and [1] focused on minimizing the peak power consumption under real-time constraints. [2] proposed a new scheduling algorithm for real-time tasks to minimize chip-level power consumption, without relying on any extra hardware for average power reduction. This work restricts the concurrent execution of tasks which are assigned to different cores. Lee *et al.* [18] have proposed a task scheduling mechanism for preventing the occurrence of peak power consumption for task-graph models. The proposed algorithm in this work schedules the tasks with the data dependency information while reduces the peak power. One other work in high correlation to our work is [1]. This work presented a scheme to minimize the peak power for frame-based and periodic tasks with real-time constraints on multicore systems. In order to minimize the peak power, [1] schedules the sleep cycles for each active core. It should be noted that researchers that try to minimize the peak power, do not consider any fault-tolerant techniques to deal with permanent, transient and intermittent faults. In this paper, we use a fault-tolerant technique (i.e. standby-sparing technique) on multicore embedded systems while minimizing the peak power consumption. In this paper, we aim at considering effects of fault-tolerant techniques on the peak power consumption in real-time embedded systems.

B. Reliability-Aware Energy Management

Some techniques, like [15], [24], [25], and [26], which consider both reliability and energy consumption, reserve a part of the available slack time to schedule a recovery task (to preserve the system reliability), and then utilize the remaining slack for energy savings. In these techniques, since both the main and recovery tasks are executed on the same core, tasks with utilization greater than 50% cannot be scheduled. Furthermore, these techniques cannot tolerate permanent faults, since both the main and recovery executions perform on the same core. The standby-sparing technique is a well-studied hardware replication technique to provide high reliability while keeping the energy consumption under control [21][27][35][29]. In standby-sparing, the system consists of two identical cores: primary and spare. Main tasks

are executed on primary and their backup tasks are executed on spare. When the primary core fails (due to either transient or permanent fault), it is replaced with the spare core to continue the execution of the backup task.

In order to reduce the energy consumption overhead of standby-sparing, [28] has proposed a technique where DVFS is used for the primary core while the spare core does not use DVFS to preserve the reliability of the system when a fault occurs. The work in [35] has proposed an energy-aware scheduling scheme for a standby-sparing system that executes preemptive periodic real-time applications. They apply Earliest-Deadline-First (EDF) scheduling with DVFS on the primary core, while the backup tasks are executed on the spare core according to Earliest-Deadline-Late (EDL) scheduling. Both EDF and EDL assign priorities based on the jobs' deadline, however, EDL delays the jobs as much as possible to obtain idle intervals as early as possible in the schedule. They aim at minimizing the execution overlap between main and backup tasks at run-time to reduce energy consumption. Haque *et al*. [29] have proposed an energy-management technique for a standby-sparing system that executes preemptive fixed-priority real-time tasks. Tasks on the primary core are scheduled by the Cycle-Conserving DVFS algorithm that has been proposed for Rate Monotonic Scheduling (RMS) in [11]. While the spare core uses DPM and dual-queue mechanism that tries to maximally delay the backup tasks to save more energy. It should be noted that, although RMS is optimal for fixed priority tasks, it lowers core utilization.

C. A Conclusion of Related Work Overview

Generally, the previous works in the context of real-time systems either present peak power minimization techniques without considering reliability like [1] and [2] or consider simultaneous management of energy and reliability without considering peak power reduction like [16], [28], and [35]. In this paper, we exploit a fault-tolerant technique (standby-sparing technique) to achieve high reliability for real-time embedded systems so that peak power consumption is kept below the chip TDP constraint.

III. Models and Assumptions

A. System and Task Model

We consider a multicore system with m cores and $m/2$ core pairs $C=\{\{C_1, SC_1\}, \ldots, \{C_{m/2}, SC_{m/2}\}\}$ similar to Intel SCC [34]. Also, we consider a set of periodic real-time tasks $\psi=\{T_1, \ldots, T_n\}$. Each task T_i has a period P_i and a worst-case execution time et_i under the maximum frequency. The j^{th} job of a task T_i (J_{ij}) arrives at time $r_{ij}=(j-1)\times P_i$ and must complete by its deadline $j\times P_i$. Hence, the relative deadline D_i of the job J_{ij} is equal to the period P_i. The utilization of the task T_i is defined as et_i/P_i. So, the sum of all tasks utilization is U_{tot}. We consider for each task T_i a backup task B_i. We denote the j^{th} job of B_i by BJ_{ij}.

B. Power Consumption Model

Each core can operate in active and sleep modes. The core executes tasks in the active mode and in this mode we compute its power consumption based on Eq. 1. The total power consumption of the system consists of static and dynamic power components [3]. The static power (P_s) is dominated by the leakage current. Dynamic power (P_d) is mainly consumed due to system activity [4][5][16][21][28][29]**Error! Reference source not found.**.

$$P_{total} = P_s + P_d \tag{1}$$

Under DVFS, the voltage V_i that is used for the execution of the task T_i should be less than the maximum voltage V_{max}. Let V_{max} be the maximum voltage corresponding to the maximum frequency f_{max}. We denote the normalized voltage ρ_i as [16][19][21][28][29][35]**Error! Reference source not found.**:

$$\rho_i = \frac{V_i}{V_{\max}} \tag{2}$$

Hence, the dynamic power consumption under the scaled voltage V_i can be written as:

$$P_d(V_i) = C_{eff} V_i^2 f_i \tag{3}$$

where C_{eff} is the average switched capacitance, V_i and f_i are supply voltage and operational frequency that is used to execute each task T_i. By considering the almost linear relationship between voltage and frequency [35], we can write: $\rho_i=V_i/V_{max}=f_i/f_{max}$. Therefore, Eq. 3 can be written as:

$$P_d(T_i) = C_{eff} V_{\max}^2 f_{\max} \rho_i^3 \tag{4}$$

Since $C_{eff}V_{max}^2 f_{max}$ is constant, the dynamic power consumption can be normalized by removing $C_{eff}V_{max}^2 f_{max}$. Therefore, the normalized power consumption of the core while executing the task T_i can be written as [3][21]:

$$NP_d(T_i) = \rho_i^3 \tag{5}$$

C. Fault Model

We consider a transient fault model similar to [29][35]. The average fault rate λ is dependent on the core frequency whereby decreasing core frequency, λ increases exponentially. The average fault rate on the frequency f can be expressed as:

$$\lambda(f) = \lambda_0 \times 10^{\frac{d(1-f)}{1-f_{\min}}} \tag{6}$$

where $\lambda_0=10^{-7}$ is the transient fault rate at f_{max} and d determines the sensitivity of the system to voltage scaling. Like the works [29][35]**Error! Reference source not found.**, we consider $d=2$ in this paper.

IV. Our Proposed Method

A. Concept Overview and Our Novel Contributions

In this paper, we consider a standby-sparing system that executes preemptive periodic real-time tasks. We propose the Peak-Power-Aware Earliest-Deadline-First (PPA-EDF) policy to schedule the main tasks on the primary cores with DVS and DPM. For the spare cores, we propose Peak-Power-Aware Earliest-Deadline-Late (PPA-EDL) policy. These policies postpone the execution of the tasks as much as possible.

Peak-Power-Aware Scheduling: The idea of peak power-aware scheduling is based on the power profile of the tasks. It is a strategy for scheduling periodic tasks with both soft and hard deadlines in real-time systems such that the chip TDP is met. It is assumed that each task has the special power trace. Then, we propose a peak-power-aware energy management scheme that enables task replication to achieve high reliability in multicore embedded systems under timing and TDP

constraints. To the best of our knowledge, the power management techniques for fault-tolerant systems that have been presented in the literature only try to reduce the average power and do not consider peak power constraints. In this paper, we propose a scheduling algorithm for periodic real-time applications on multicore embedded systems when the standby-sparing technique is used for fault tolerance. At first, our proposed scheme generates tasks' execution time and extract tasks' power trace through offline profiling. Then, our proposed scheme uses the Peak-Power-Aware Earliest-Deadline-First (PPA-EDF) policy for the main tasks and the Peak-Power-Aware Earliest-Deadline-Late (PPA-EDL) policy for their backup tasks. Finally, the tasks are spread over the schedule such that the peak power consumption is kept below the chip's TDP. To achieve further power reduction, if during the execution of the first copy of a task no fault has occurred, the second copy of the task is not required, and then its execution is canceled. We do not propose to execute the backup, as we might not require this. Indeed, we only propose to reserve a backup to execute it in case a fault occurs but usually as no fault occurs we do not require executing the backup. Also, it should be noted that in the offline phase we guarantee both primary and backup tasks are scheduled and will be executed at run-time. Therefore, the reliability of the proposed method is preserved at an acceptable level known as the reliability of the standby-sparing system.

B. Problem Definition and Schedulibility analysis

Dertouzos in [36] has demonstrated that EDF is optimal in feasibility, i.e. if there exists a feasible schedule for a task set ψ, then EDF may find it. However, EDF does not guarantee meeting TDP, reliability requirement and deadlines simultaneously. In the modern multicore hard real-time systems, in addition to meeting all deadlines, the system scheduling policy must satisfy the system reliability requirement and meet the chip-level power constraint [1][4][5][35]. In order to show the difference between meeting the three mentioned constraints and meeting deadlines without considering other constraints, we define our problem. Therefore, we use the following notation to represent energy and peak power consumption, voltage and frequency level and task-to-core mapping. In this formulation, n is the number of tasks, m is the number of cores, v is the number of available V-f levels for each core, and s is the number of time slots:

- The peak power consumption is represented by the matrix $P \in \mathbb{R}^{n\times m\times v\times s}$, in which each element P_{iklt} denotes the power consumption for the task i when the task is executed on the core k at the time slot t under the V-f level l.
- The task-to-core mapping and V-f level assignments are represented by the matrix $X \in \{0,1\}^{n\times m\times v}$. The task i is mapped to the core k and is executed under the V-f level l if and only if $X_{ikl} = 1$.

The goal of our method is to minimize the total energy consumption while keeping the instantaneous power consumption under chip-level power constraint ($P_{TDP,Chip}$) and meeting tasks timing constraints (deadlines). We formulate the above problem in the following.

Optimization Goal: Minimize the total energy consumption defined by the sum of the energy consumption of all tasks.

$$\text{Minimize } E_{total} = \sum_{i=1}^{i=n} E_{STANDBY-SPARING}(T_i, B_i) \tag{7}$$

Chip Power Constraint: The instantaneous total power consumption, i.e. the sum of the peak power of all underlying cores at each time slot t must be less than the chip TDP constraint. In the following equation, h is the least common multiple (LCM) of all task periods.

$$\forall t \in h : \sum_{i,k,l} X_{i,k,l} P_{i,k,l} \le P_{TDP,chip} \tag{8}$$

Tasks Timing Constraint: The worst-case execution time et_i/f_{kl} for a task i on the core k and at the frequency level l should not exceed the task timing constraint (defined by the D_i).

$$X_{i,k,l} \frac{et_i}{f_{kl}} \le D_i \tag{9}$$

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

$$\forall l : \sum_{i} \sum_{k} X_{i,k,l} = 1 \tag{10}$$

V-f Levels Assignment Constraint: Each task can be only executed under a single V-f level on a core (the V-f level does not change during the task execution).

$$\forall i,k : \sum_{l} X_{i,k,l} \le 1 \tag{11}$$

The order of slices of a task: In order to ensure that the order of slices of a task is met, we check that the next part of each task $T_{i(j+1)}$ does not place before the previous parts of the selected task.

$$\forall i,j : t_{T_{ij}} < t_{T_{i(j+1)}} \tag{12}$$

Since solving the above problem and finding a schedule for a multicore system to optimally minimize energy consumption is an NP-hard problem [22][35]**Error! Reference source not found.**, we present a heuristic to provide a solution for peak power reduction and energy minimization (see Section IV.C). It should be noted that in this paper we assume that the system must meet three constraints: *i*) the power constraint: Thermal Design Power (TDP) constraint, *ii*) the system reliability target, *iii*) timing constraints (Deadlines). Therefore, we propose a peak- power-aware energy management scheme that meets the mentioned constraints simultaneously. Such a scheme requires more time overhead as compared to other schemes that consider fewer constraints especially those previous works that do not guarantee to satisfy the TDP constraint like [16], [22], and [35]. Therefore, for meeting TDP, timing and reliability constraints simultaneously, we must consider more time overhead as compared to other schemes.

C. Algorithm Discussion

In this section, we present the details of our scheme in two phases: (*i*) Offline Phase (before we actually execute all jobs) and (*ii*) Online Phase. The main tasks are executed on the primary cores according to the PPA-EDF policy and based on the amount of the static and dynamic slack time, we apply DVFS and DPM. The spare cores are reserved for the execution of backup tasks based on our PPA-EDL policy. Also, when the idle time of each core is greater than the break to sleep time (see **Online Phase**), DPM is applied.

Offline Phase: Algorithm 1 presents the offline task mapping and scheduling part of our scheme (Design Time).

```
Function PPA_EDF(J_ij, k_ij, t)
1.  foreach free slot l=t→ j×P_i in S_c_i do
2.      if PA[l]+ peak_power(k_ij) ≤ P_TDP,Chip then
3.          S_c_i.add(l, J_ijk);
4.          PDA[l] = PDA[l]+ peak_power(k_ij);
5.          k_ij=k_ij+1;
6.          return k_ij;
7.      end if;
8.  end for;
END Function
```

```
Function PPA_EDL(BJ_ij, bk_ij,, t)
1.  foreach free slot l= j×P_i →t in S_sc_i do
2.      if PA[l]+ peak_power(bk_ij) ≤ P_TDP,Chip then
3.          S_sc_i.add(l, BJ_ijk);
4.          PA[l] = PA[l]+ peak_power(bk_ij);
5.          bk_ij=bk_ij -1;
6.          return bk_ij;
7.      end if;
8.  end for;
END Function
```

Algorithm 1. Design Time: Task Mapping and Scheduling

INPUT: ready tasks Ψ with the execution time and the deadline, set of free cores C, and the chip-level power constraint $P_{TDP,Chip}$.
OUTPUT: Task Mapping and Task Scheduling

```
BEGIN
1.  h=LCM(the periods of all tasks);                    //Calculate hyperperiod
2.  PA [1...h]={0};                                     //Initialize the total power consumption array
3.  S_c_i={Null, 1≤i≤m/2};   //Initialize the primary cores with an empty schedule
4.  S_sc_i={Null, 1≤i≤ m/2};        //Initialize spare cores with an empty schedule
5.  while (Ψ is not empty) do
6.      T_i= Ψ.remove();                                //Select a task
7.      Φ = C.min_utilization;                          //Find a pair core with lowest utilization
8.      Φ.C.add(T_i);
9.      Φ.SC.add(B_i);
10. end while
11. t=0;
12. for all jobs do                                     //Partition all the jobs into parts
13.     J_ijl = {J_ijl, 1≤l≤et_i};
14.     BJ_ijl={BJ_ijl, 1≤l≤et_i};
15. end for;
16. k_ij ={1, for all primary jobs};
17. bk_ij ={et_ij, for all backup jobs};
18. while (t≤h) do
19.     //Event at time t: A job of T_i (J_ij) is released at time t on a core pair
20.     if the primary core c_i is busy then
21.         if priority(J_ij) > priority(J_mn) then
22.             k_ij=PPA_EDF(J_ij, k_ij, t);
23.         else
24.             k_ij=PPA_EDF(J_mn, k_ij, t);
25.         end if
26.     else                                            //The primary core c_i is idle
27.         k_ij=PPA_EDF(J_ij, k_ij, t);
28.     end if
29.     if the spare core sc_i is busy then
30.         if priority(BJ_ij) > priority(BJ_mn) then
31.             bk_ij=PPA_EDL(BJ_ij, bk_ij, t);
32.         else
33.             bk_ij=PPA_EDL(BJ_mn, bk_ij, t);
34.         end if
35.     else                                            //The primary core c_i is idle
36.         bk_ij=PPA_EDL(BJ_ij, bk_ij, t);
37.     end if
38.     t= t+1;
39. end while
40. if not all the jobs are scheduled then
41.     return infeasible;
42. end if;
END
```

This algorithm calls two functions: (*i*) the peak-power-aware earliest deadline first function (*PPA_EDF*(J_{ij}, k_{ij}, t)), (*ii*) the peak-power-aware earliest deadline late function (*PPA_EDL*(BJ_{ij}, k_{ij}, t)). The *PPA_EDF* function implements the PPA_EDF policy to make a schedule on the primary cores. The *PPA_EDF* function schedules a time slot of the selected job on the schedule of its designated primary core. The function uses the variable t to determine the first time slot in which the current time slot of the selected job k_{ij} can be placed. The *PPA_EDF* function iterates until the deadline of job J_{ij} is not violated and then checks free time slots of the primary core one after another beginning from the released time of a job J_{ij}, and places the current part of the job on the first free time slot such that the total peak power consumption does not exceed the chip TDP constraint. On the other hand, the *PPA_EDL* function implements the PPA_EDL policy to make a schedule on the spare cores. Since the *PPA_EDL* function starts from the end of the execution time for scheduling the selected job, it places the current time slot of the selected backup job on the schedule of its designated spare core. The *PPA_EDL* function searches to find the best position for the current time slot of the selected backup job such that the total peak power consumption does not exceed the chip TDP constraint. The *PPA_EDL* function schedules the time slots of the backup beginning from the last part. Also, until the released time of the selected job j is not violated, the function does it.

Algorithm 1 gets the ready tasks with their execution time and deadline, the set of free cores and the chip TDP constraint $P_{TDP,Chip}$. In line 1, the algorithm computes hyperperiod h which is defined as the least common multiple (LCM) of all task periods. The algorithm employs a power array including h slots that each slot determines the power consumption of the system at a time slot. Then, the algorithm initializes two schedules S_c_i and S_sc_i to Null for each core of C. In line 5 to 10, the algorithm iterates until all the tasks are assigned to a core pair based on the lowest utilization first policy. To do this, we assign tasks one after another to a core pair which has the lowest utilization. In line 11, the variable t is initialized to zero to show the time of the system schedule. The algorithm partitions all the jobs in lines 12-15. The variables k_{ij} and bk_{ij} are initialized to one value and et_{ij} to show the current time slot of the main and backup jobs, respectively. In lines 18 to 39, the algorithm iterates until all parts of the main jobs and the backup jobs are scheduled based on PPA-EDF and PPA-EDL, respectively. In line 19, when a job of T_i (J_{ij}) is released at time t, the algorithm checks its designated primary core. If the primary core is idle, it is scheduled immediately. Otherwise, if the primary core c_i is busy and the priority of J_{ij} is higher than the priority of J_{mn} which is executing on the primary core, the algorithm schedules the current time slot of the selected job on the place of it. Therefore, a job preempts a primary core only if it has a higher priority than the currently executing job according to the PPA-EDF policy (lines 21-23). In this case, the current job is preempted. During preemption, we update the variable k_{ij} to the remaining time required to complete the job (J_{mn}). Otherwise, J_{mn} is executed on the primary core (line 24). This scenario also happens on the spare core with the difference that the *PPA_EDL* policy schedules the parts of the backup jobs beginning from the last part, i.e. the algorithm checks the free time slots one after another starting from the bk_{ij} and places BJ_{ijk} on the first free time slot. Finally, if not all the jobs are scheduled, the algorithm returns *infeasible* in line 41.

Online Phase: At runtime, when a job is released, if the primary core is idle, the job is executed on the core. However,

Online Manager 1: DVFS Function

1. **Function** DVFS(J_{ij});
2. *slack* ← *Extract_Slack*(*t*, deadline(J_{ij}));
3. $f_{ij} \leftarrow \max(f_{ee}, \frac{et_{ij}}{et_{ij}+slack})$;
4. Execute J_{ij} at frequency f_{ij};
5. **End Function**

Online Manager 2: Acceptance Test Function

1. **Event – A job completes at time *t***:
2. Run the acceptance test; //A low-cost hardware checker, Argus [37]
3. **If** no error is detected **then**
4. Cancel the corresponding copy of the job;
5. **end if**
6. **If** ready queue of the core is empty **then**
7. $\Delta_{er} \leftarrow$ time to earliest release time;
8. **If** $\Delta_{er} \geq \Delta_{critical}$ **then**
9. Put core to sleep mode for Δ_{er} units of time;
10. **end If**
11. **else** /* jobs are available for execution */
12. Execute a job;
13. **end If**

if the core is running another job, the execution priorities are assigned according to PPA_EDF. If preemption occurs, we update the minimum additional time required to complete the job in the worst-case (under the maximum frequency). Then, we find slack time during *t*, i.e. the time that a job is executed or resumed from preemption, and its deadline. It should be noted that if there is no slack time, the job runs with the maximum frequency on the primary. To find the minimum frequency for a job execution, we use [35]:

$$f_{ij} = \max(f_{ee}, \frac{et_{ij}}{et_{ij} + slack}) \quad (13)$$

where f_{ee} is called the energy-efficient speed which is a processing frequency that below it, the total energy consumption of a task increases [3][5][35][29]. f_{ee} is computed analytically in [15] and [17]. The function of our DVFS technique is shown in **Online Manager 1**. It should be noted that if postponing a task or DVFS is required, the task execution parts are shifted over the schedule such that the peak power consumption is kept bellow the chip TDP. Therefore, we always check the peak power consumption kept bellow the chip TDP. Also, when a job completes on each core, we call an acceptance test [37] to check the correctness of the task execution. If the acceptance test does not detect any fault, we cancel the corresponding copy task in the corresponding core. Then, the designated core continues with executing the next job in the ready queue. On the other hand, if the first copy of job is faulty, we can continue running the corresponding core as scheduled. If there is no ready task available for execution, the core will remain idle. The core will start executing jobs again when the next job arrives. By the use of the task period values, we can compute the earliest release times among all future jobs in linear time. In order to apply DPM on cores, let assume we have a *break to sleep time* ($\Delta_{critical}$) [29]. When the idle time of a core is greater than $\Delta_{critical}$, the core switches to sleep mode, and hence, all power components other than the static power P_s are removed. Therefore, if the idle time exceeds the break to sleep time ($\Delta_{critical}$), the core is put into sleep mode until next release. Finally, if there is ready task available for execution, the core will execute the task. The function of our DPM technique and task dropping method is shown in **Online Manager 2**. As the final discussion of this subsection, we explain the time required to meet the TDP, timing and reliability constraints simultaneously. Since we shift some tasks to the next time slots to reduce peak power, we need more time slots for meeting the deadline. In order to shift tasks to the next time slots for execution, we should find the exact execution time because tasks should not miss their deadlines. It should be noted that in this paper, we have focused on meeting TDP, timing and reliability constraints simultaneously. Therefore, our proposed scheme incurs more time overhead as compared to other schemes that consider fewer constraints, e.g. the references [29], [35], and [38]. Therefore, for meeting TDP, timing and reliability constraints simultaneously, we must consider more time slots.

D. Complexity Analysis

In our problem formulation, *n* is the number of tasks, *m* is the number of free cores, *h* is the total time slots, and *l* is the number of V-f levels. In Algorithm 1, the ready tasks and free cores are sorted in $O(n\ log\ (n))$ and $O(m\ log\ (m))$, respectively. In the rest of the algorithm, the main computation is performed to examine all V-f level scalings and scheduling parts (*h* parts) for all ready tasks and then putting them into a max-heap. Therefore, for *n* ready tasks and *l* V-f levels, building the max-heap is performed in

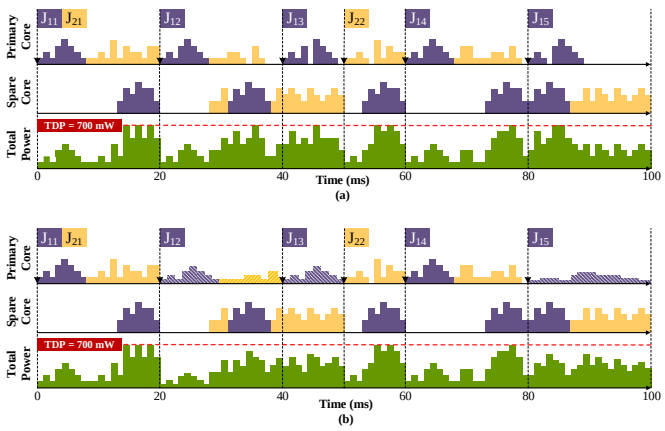


Fig. 2. Offline scheduling the tasks of the illustrative example, (a) Scheduling the tasks based on the PPA-EDF and PPA-EDL policies without DVFS, (b) Applying DVFS.

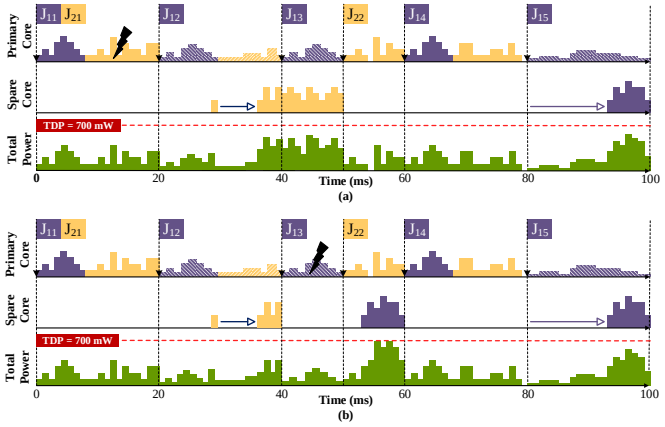


Fig. 3. Taking advantage of fault-free execution and shifting the remaining tasks, (a) Executing and shifting the backup job (bj_{21}) due to the faulty execution of the main job, (b) The other faulty condition in executing the tasks of the example.

Table 1. Characteristics of the benchmark applications

	BITCOUNT	SUSAN	MATH	CRC32	SHA	QSORT	JPEG	FFT	DIJKSTRA	LAME	GSM
Execution time (ms)	388	238	2198	4158	80	414	98	1924	182	8990	1412
Energy consumption (mJ)	138.45	88.12	773.82	1283.97	29.05	146.35	36.62	676.72	60.80	3158.55	482.96
Min. Power (mW)	**308.63**	**307.05**	**299.76**	**294.84**	**304.34**	**308.67**	**306.93**	**307.38**	**308.77**	**307.07**	**351.7**
Max. Power (mW)	**391.77**	**387.76**	**362.17**	**315.2**	**424.88**	**363.94**	**482.37**	**368.01**	**350.02**	**357.8**	**307.1**

$O(n\times h\times l)$. The first **while** loop iterates for $O(n\times m)$ times. The **For** loop iterates for $O(n\times h)$ times. The main **while** loop iterates for $O(n\times m\times h)$ times. Finally, the algorithm removes the V-f level scalings from the heap one after another and checks whether the TDP constraint is met. This step is also done in $O(m\times n\times l)$. Therefore, the order of the algorithm is $max\{O(m\ log\ (m)), O(n\ log\ (n)), O(m\times n\times l), O(n\times m\times h)\}$. For the final discussion of the complexity, it should be noted that the parameter h is defined as the least common multiple (LCM) of all task periods. Therefore, there is no reason that h increases exponentially as the input size increases. Even if h was exponentially proportional to task periods, this could cause problems for large task periods. However, in practical real-time embedded systems, the task period does not usually get that large even though we may have many tasks in the schedule. It is also noteworthy that when determining hyperperiod it is common to slightly increase basic periods to avoid large hyperperiods [2][5][35].

E. Illustrative Example

As an example let us consider two periodic task T_1 and T_2 with the period P_i: P_1=20ms and P_2 =50ms. We consider a dual-core chip with 700mW of TDP where the cores (C1 and C2) constitute a core pair. In this example, we consider an architecture model which is based on ARM's Big.LITTLE architecture [30] to show that due to the heterogeneity different applications have different values (the worst-case execution time, peak power, and etc.) when running on the different types of cores. Since heterogeneous multicore embedded systems have at least two types of cores (low-power type and high-performance type), we can use the high-performance type as the primary core and the low-power type as the spare core. Therefore, the worst-case execution time et_i is different between the primary core and the spare core. The worst-case execution time of the tasks on the primary core is: et_1=8ms and et_2=20ms; and the worst-case execution time of the tasks on the spare core are: et_1=7ms and et_2 =15ms. For this task set, the hyperperiod is h=100ms which is the least common multiple of all the task periods. Therefore, within a hyperperiod, 5 jobs of T_1 and 2 jobs of T_2 are executed. By the use of the tasks power traces, the peak power values for the parts of the tasks are determined. In this example, the different parts of each task have different peak power values.

Fig. 2 shows how this task set is scheduled and executed by our scheme. On the primary core, the tasks are scheduled by the use of the preemptive Peak-Power-Aware Earliest-Deadline-First (PPA-EDF) scheduling (Fig. 2a). Note that if the main and backup tasks are scheduled in the same way on the primary and spare cores (e.g. both are scheduled with EDF), the energy consumption will significantly increase. This is because main tasks are executed with their backups in parallel. The energy overhead can be reduced by delaying the execution of backup tasks on the spare core [16]. However, the deadlines of the backup tasks have to be guaranteed. To address this issue, for the spare core (Fig. 2a), we exploit the preemptive Peak-Power-Aware Earliest-Deadline-Late (PPA-EDL) scheduling while the total power consumption is kept below the TDP constraint. In Fig. 2b, we exploit the DVFS technique to reduce both the peak power and energy consumption. For example, the jobs J_{12}, J_{13}, and J_{15} in the primary core are executed using DFVS to reduce peak power and to save energy. On the spare core, the backup tasks are executed at maximum voltage/frequency (see Fig. 2b). Fig. 3 shows two faulty scenarios, where some tasks are faulty and some other tasks are executed successfully. We observe that the backup tasks can be avoided entirely in many scenarios. For instance, when J_{11} is executed successfully, the corresponding backup task is canceled in the time slot [13ms, 20ms] on the spare core and excessive power and energy consumption is avoided. Also, when the backup tasks are canceled on the spare core, we shift some other tasks to the next time slots to further reduce peak power and energy. In order to shift jobs to the next time slots for execution, we should find the exact promotion time since tasks should not miss their deadlines and the chip TDP. The first step in finding the exact promotion time is computing the worst-case response time of each task. There are multiple techniques to compute the worst-case response time of the tasks, e.g. [29]. For instance, when J_{21} is faulty, the corresponding backup job should be executed on the spare core. Since the backup job of the main job J_{12} is dropped in the time slot [30ms, 38ms], the two part of the backup job of J_{21} is shifted in the time slot [36ms, 38ms]. Therefore, by the use of the PPA-EDL scheduling further power reduction and energy saving can be gained.

V. Results and Discussion

In this section, we evaluate the effectiveness of our proposed scheme via simulation with various task sets including real-life embedded applications of MiBench

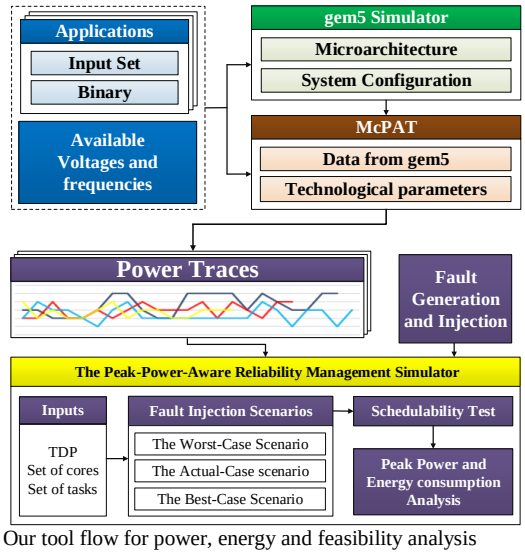
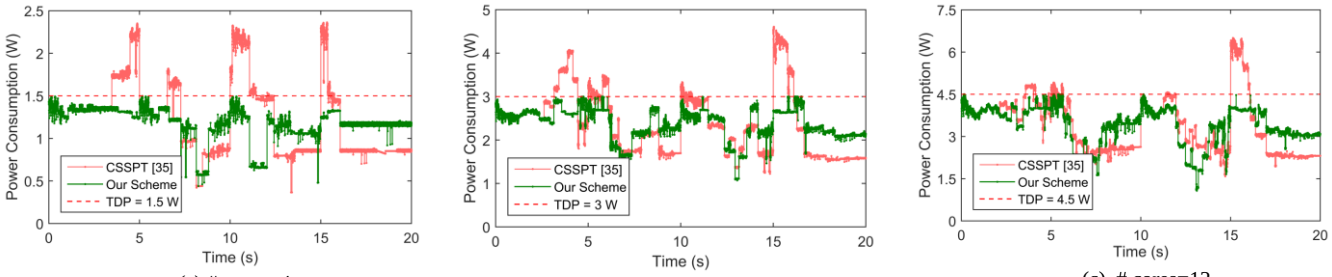


Fig. 4. Our tool flow for power, energy and feasibility analysis



(a) # cores=4 (b) # cores=8 (c) # cores=12

Fig. 5. Power consumption profile for a periodic task set in the actual-case execution scenario.

Benchmark suite [31] running on a target multicore chip. Our evaluation consists of the comparison between our scheme and four state-of-the-art schemes. We compared our scheme with the ASSPT [35], CSSPT [35], RAPM [38] and SSFP [29] algorithms:

- ASSPT and CSSPT [35]: These techniques are proposed to manage energy consumption in conjunction with fault-tolerance. ASSPT and CSSPT execute original tasks as soon as possible through EDF and backup tasks as late as possible through EDL. ASSPT allows a job to utilize the entire available slack and slow down as much as possible while CSSPT tries to achieve a balanced slack distribution among all jobs.
- RAPM [38]: This technique is proposed in [38]. For the fair comparison, we assumed that RAPM uses one backup task for each task to achieve fault tolerance. This technique proposed both individual-recovery and shared-recovery based reliability-aware power management heuristics.
- SSFP [29]: This technique has proposed an energy management technique for a standby-sparing system that executes preemptive fixed-priority real-time tasks.

In order to evaluate our scheme, we constructed a system-level simulator. In our simulations, for each data point, we generated 1000 task sets and the average results are reported. Each task set consists of 10 tasks. The task sets are selected randomly from Table. 1. To generate Table. 1, we use gem5 full-system simulator [32] and McPAT [33]. Since ARM processors are widely used in many embedded systems, we consider an ARM processor. Therefore, a detailed model of ARM processors provided by gem5 is used in this paper. This ARM core has an area of 9.74mm^2 with 32KB L_1 cache and a shared 1MB L_2 cache. Fig. 4 shows our tool flow and simulation setup with fault generation and injection, scheduling simulation, and power/energy evaluation. We considered that the system supports DVFS and can work at five different voltage and frequency levels be-tween [0.85Volt, 1GHz] and [1.1Volt, 2GHz]. Also, we evaluate the actual-case execution scenario where both faulty and fault-free execution scenarios were considered. To generate fault rate and pattern, in our experiments, transient faults were generated using a Poisson process where the fault rate λ corresponding to different voltage levels was modeled using Eq. 6 under the parameters λ_0=10^{-6} faults/us and d=2 [3]. Therefore, the fault rate varies between 10^{-6} faults/us and 10^{-2} faults/us corresponding to f_{max} and f_{min}, respectively. To do this, we exploited a system-level fault injection. At first, we generate a fault vector that determines at which times faults occur. Then, based on the fault vector, we decide which task becomes faulty during the execution of a task set.

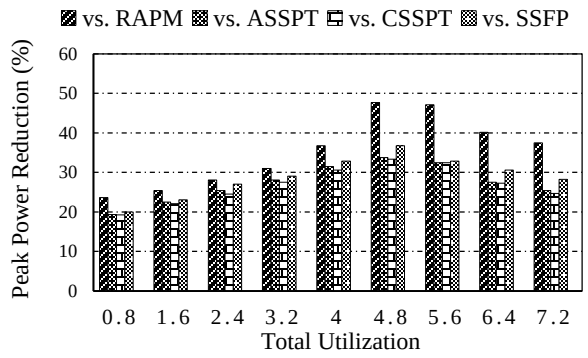


Fig. 6. Peak power reduction in different system utilizations.

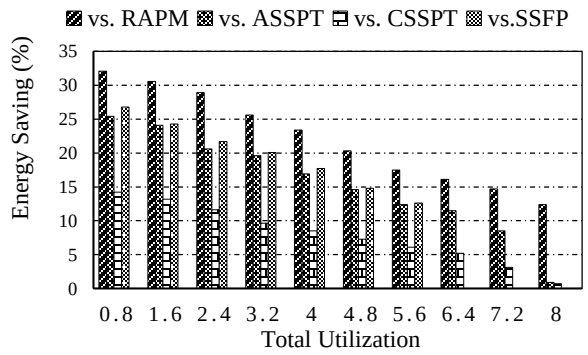


Fig. 7. Energy saving in different system utilizations.

Fig. 5 shows the power profile of our scheme and CSSPT [35] for a different number of cores (M=4, 8 and 12) and core workload (U_{per_core}=0.75). As Fig. 5 shows, our scheme reduce peak power through distributing power consumption over the whole execution hyperperiod. In this figure, CSSPT misses the TDP constraint. Then, we evaluated the ratio of peak power reduction and energy saving of our scheme versus RAPM, ASSPT, CSSPT, and SSFP across different system parameters including the total utilization (U_{tot}). We evaluated the impact of the total utilization varies from 0.8 to 8. Fig. 6 and Fig. 7 show the results for the cases when tasks worst-case execution times are generated based on Table. 1 and the number of cores is equal to 8. What can be inferred from these figures is that our scheme completely outperforms the other four schemes for all utilization values. This is achieved through reducing the overlap of the execution of main tasks on the primary core and their corresponding backup tasks on the spare core. Therefore, by further postponing the backup tasks at runtime, in many cases we can cancel the backup tasks on the spare core. Each case of Fig. 6 was simulated for 1000 times with different parameters of the

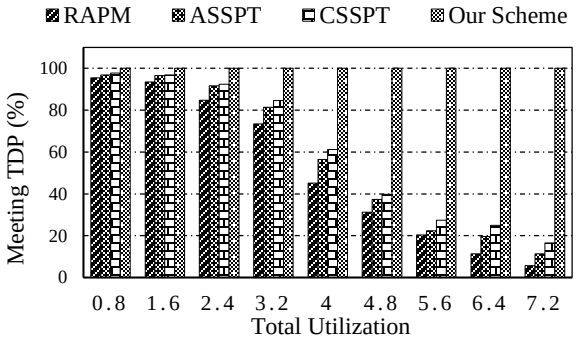


Fig. 8. The frequency of meeting power constraint.

applications and the average results are reported. This figure shows that our scheme provides up to 47.6% (on average by 28.2%) peak power reduction compared to four state-of-the-art techniques. From Fig. 6 it can be concluded that in all utilization points, the peak power reduction of our scheme is higher than other schemes. When the utilization point is between U_{tot}=3.2 to U_{tot}=6.4, the peak power reduction of our scheme is better than when it is U_{tot}=5.6 or U_{tot}=7.2. This is because the probability of TDP violation in the higher utilization points in all techniques is high and the amount of the slack times in the lower utilization points for peak power reduction is high. Also, in Fig. 7, for all utilization values our scheme can save more energy, compare to RAPM. The main reason is that RAPM is forced to run at high frequency and consequently consuming too much energy; on the whole, by increasing the utilization, the energy saving decreases (Fig. 7). Since, when utilization is low, more slack time can be achieved, this further slack time helps us to save more energy through DVFS and DPM. Experiments show that our scheme provides up to 32.1% (on average by 16.3%) energy savings compared to four state-of-the-art techniques. Also, it can be seen from Fig. 7 that when the utilization of the cores increases, the energy saving decreases. This is because the amount of static and dynamic slack times decreases, and hence we cannot achieve significant energy savings. However, the energy consumption of our scheme is always less than the other schemes. Also, it should be noted that the schedulability of the results shown in the figures 6 and 7 is 100%. Indeed, all the results in these figures meet all their deadlines. Moreover, our scheme always meets TDP constraint but other schemes violate TDP in some cases. As the final discussion, we discuss the feasibility of the proposed method. To demonstrate this, we generated 1000 task sets from the tasks shown in Table. 1 and repeated the simulations for several utilization values ($U_{per\text{-}core}$= 0.4, 0.5, 0.6, 0.7, and 0.8). Since the proposed approach never violates TDP, we show the frequency of power budget violations during execution. Fig. 8 shows the frequency of meeting TDP for our proposed scheme and other schemes. It can be seen from Fig. 8 that the proposed approach never violates TDP while the state-of-the-art schemes violate it because the state-of-the-art schemes do not consider it. Then, we verified the schedulability of the proposed method and other schemes for each generated task set. Then we obtain the percentage of meeting the deadlines and the TDP constraint simultaneously. The results show that our scheme meets the timing and TDP constraints on average by 78% while the other schemes meet the mentioned constraints on average by 63%. Therefore, our proposed method is more suitable and efficient than other schemes for meeting the timing and TDP constraints simultaneously.

VI. Conclusions

In this paper, we have considered two main objectives in designing real-time embedded systems, denoted as reliability and power consumption. To achieve these objectives, we proposed a scheduling scheme for standby-sparing systems that executes periodic real-time tasks. We have proposed the PPA-EDF and PPA-EDL policies on the primary and spare cores, respectively. The proposed scheme postpones the execution of tasks on spare cores. Since faults are a naturally rare event and also tasks often consume less than their worst-case execution time, tasks commonly complete early or successfully. Another feature of our scheme is that it provides a good opportunity to cancel the second copy of a task when the first copy of the task completes early or successfully; resulting in a reduced power consumption. We compared our scheme with the RAPM, SSFP, ASSPT and CSSPT schemes. Simulation results show that our scheme provides up to 47.6% peak power reduction compared to the other schemes.

References

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

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

[3] M. Salehi, A. Ejlali, and B.M. Al-Hashimi, "Two-Phase Low-Energy N-Modular Redundancy for Hard Real-Time Multi-Core Systems," *IEEE Trans. on Par. and Dis. Sys. (TPDS)*, vol. 25, no. 4, pp. 1024-1033, April 2015.

[4] 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. on Comp.*, vol. 66, no. 1, pp. 147-162, 2017.

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

[6] S. Pagani, H. Khdr, W. Munawar, J. J. Chen, M. Shafique, M. Li, and J. Henkel "TSP: Thermal Safe Power - Efficient Power Budgeting for Many-Core Systems in Dark Silicon," in *IEEE/ACM Intern. Conf. on Hard./Soft. Codesign and Sys. Syn.*, New Delhi, India, Oct. 2014.

[7] M. Al Faruque, J. Jahn, T. Ebi and J. Henkel, "Runtime Thermal Management Using Software Agents for Multi- and Many-Core Architectures," *IEEE Design & Test*, vol. 27, no. 6, pp. 58-68, 2010.

[8] D. Pradhan, *Fault-tolerant computer system design*. Upper Saddle River, N.J.: Prentice Hall PTR, 1996.

[9] X. Castillo, S.R. McConnel, and D. P. Siewiorek, "Derivation and calibration of a transient error reliability model," *IEEE Trans. on Comp.*, vol. 31, pp. 658-671, July 1982.

[10] R. K. lyer, D.J. Rossetti, and M. C. Hsueh, "Measurement and modeling of computer reliability as affected by system activity," *ACM Trans. Comput. Syst.*, vol. 4, pp. 214-237, August 1986.

[11] P. Pillai and K. Shin, "Real-Time Dynamic Voltage Scaling for Low-Power Embedded Operating Systems," *ACM SIGOPS Operating Systems Review*, vol. 35, no. 5, p. 89, 2001.

[12] J.-J. Chen, S. Wang, and L. Thiele, "Proactive Speed Scheduling for Real-Time Tasks under Thermal Constraints," *Proc. IEEE Real-Time and Embedded Tech. and App. Symp. (RTAS)*, pp. 141-150, 2009.

[13] N. Fisher, J.-J. Chen, S. Wang, and L. Thiele, "Thermal-Aware Global Real-Time Scheduling on Multicore Systems," *Proc. 15th IEEE Real-Time Tech. and App. Sym. (RTAS)*, pp. 131- 140, 2009.

[14] D. Ernst, S. Das, S. Lee, D. Blaauw, T. Austin, T. Mudge, N. S. Kim, and K. Flautner, "Razor: circuit-level correction of timing errors for low- power operation," *Micro IEEE*, vol. 6, pp. 10–20, 2004.

[15] D. Zhu, R. Melhem, and Mosse, "The effects of energy management on reliability in real-time embedded systems," in *Proceedings of Int'l Conf. Computer Aided Design*, pp. 35–40, 2004.

[16] A. Ejlali, B.M. Al-Hashimi, and P. Eles, "A Standby-Sparing Technique with Low Energy-Overhead for Fault-Tolerant Hard Real-Time System," in *Proc. International Conference on Hardware-Software* Codesign *and System Synthesis (CODES+ISSS 2009)*, pp. 193-202, Grenoble, France, October 2009.

[17] R. Jejurikar, C. Pereira and R. Gupta, "Leakage aware dynamic voltage scaling for real-time embedded systems," *Design Automation Conference, 2004. Proceedings. 41st*, pp. 275-280, 2004

[18] B. Lee, J. Kim, Y. Jeung, J. Chong, "Peak power reduction methodology for multi-core systems," *International SoC Design Conference (ISOCC)*, 2010, pp.233-235, 22-23 Nov. 2010.

[19] S. Aminzadeh, and A. Ejlali, "A Comparative Study of System-Level Energy-Management Methods for Fault-Tolerant Hard Real-Time Systems," *IEEE Trans. on Comp.*, vol. 60, no. 9, pp. 1288-1299, 2011.

[20] T.D. Burd, T.A. Pering, and A.J. Stratakos, "A dynamic voltage scaled microcore system," *IEEE J. Solid-State Circuits (JSSC)*, vol. 35, no. 11, pp. 1571-1580, 2000.

[21] M. Salehi, and A. Ejlali, "A Hardware Platform for Evaluating Low-Energy Multicore Embedded Systems Based on COTS Devices," *IEEE Trans. on Industrial Electronics*, vol. 62, no. 2, pp. 1262-1269, 2015.

[22] A. Ejlali, B.M. Al-Hashimi, and P. Eles, "Low-Energy Standby-Sparing for Hard Real-Time Systems," *IEEE Trans. on Computer Aided Design of Integrated Circuits and Systems*, vol. 31, no. 3, pp. 329-342, 2012.

[23] T. Ebi, D. Kramer, W. Karl and J. Henkel, "Economic learning for thermal-aware power budgeting in many-core architectures," *Proceedings of the Ninth IEEE/ACM/IFIP Intern. Con. on Hardware/Software Codesign and Sys. Syn. (CODES+ISSS)*, Taipei, 2011, pp. 189-196.

[24] I. Koren, and C.M. Krishna, Fault-Tolerant Systems, Morgan Kaufman, 2007.

[25] 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.

[26] R. Melhem, D. Mosse, and E. Elnozahy, "The Interplay of Power Management and Fault Recovery in Real-Time Systems," *IEEE Trans. on computers*, vol. 53, pp. 217-231, 2004.

[27] B. Zhao, H. Aydin, and D. Zhu, "Enhanced reliability-aware power management through shared recovery technique," in *Proc. int'l conf. on Computer Aided Design (ICCAD)*, 2009.

[28] M. Khavari Tavana, M. Salehi, and A. Ejlali, "Feedback-Based Energy Management in a Standby-Sparing Scheme for Hard Real-Time Systems," in *Proc. of the 32nd IEEE Real-Time Sys. Sym.*, 2011.

[29] M.A. Haque, H. Aydin, and D. Zhu, "Energy Management of Standby-Sparing Systems for Fixed-Priority Real-Time Workloads," *Green Computing Conf. (IGCC)*, Arlington, June 2013.

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

[31] 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.

[32] N. Binkert, *et. al*, "The gem5 simulator," *ACM SIGARCH Computer Architecture News*,vol. 39, no. 2, pp. 1–7, May 2011.

[33] S. Li, *et. al*, "McPAT: An integrated power, area, and timing modeling framework for multicore and manycore architectures," in *MICRO*, pp. 469–480, 2009.

[34] Intel Corporation, "Single-chip cloud computer (SCC)," 2009. [Online]. Available: http://www.intel.com/content/www/us/en/research/intel-labs-single-chip-cloud-overview-paper.html

[35] M.A. Haque, H. Aydin, and D. Zhu, "Energy-Aware Standby-Sparing Technique for Periodic Real-Time Applications," *Proc. IEEE 29th Int'l Conf. Comput. Design (ICCD'11)*, pp. 190-197, Oct. 2011.

[36] M. L. Dertouzos. "Control robotics: the procedural control of physical processes," *Information Processing*, 74, 1974.

[37] A. Meixner, M. E. Bauer and D. Sorin, "Argus: Low-Cost, Comprehensive Error Detection in Simple Cores," *40th Annual IEEE/ACM Intern. Symp. on Microarchitecture (MICRO)*, Chicago, IL, pp. 210-222, 2007.

[38] 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.

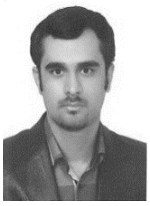
[39] M. Ansari, S. Safari, A. Yeganeh-Khaksar, M. Salehi and A. Ejlali, "Peak Power Management to Meet Thermal Design Power in Fault-Tolerant Embedded Systems," in *IEEE Transactions on Parallel and Distributed Systems*, vol. 30, no. 1, pp. 161-173, 1 Jan. 2019.

[40] T. Ebi, M. A. A. Faruque and J. Henkel, "TAPE: Thermal-aware agent-based power econom multi/many-core architectures," *IEEE/ACM International Conference on Computer-Aided Design - Digest of Technical Papers*, pp. 302-309, San Jose, CA, 2009.

[41] K. Chen, J. Chen, F. Kriebel, S. Rehman, M. Shafique and J. Henkel, "Task Mapping for Redundant Multithreading in Multi-Cores with Reliability and Performance Heterogeneity," in *IEEE Transactions on Computers*, vol. 65, no. 11, pp. 3441-3455, 1 Nov. 2016.

[42] M. Shafique, D. Gnad, S. Garg and J. Henkel, "Variability-aware dark silicon management in on-chip many-core systems," *Design, Auto. & Test in Europe Conf. & Exh. (DATE)*, Grenoble, pp. 387-392, 2015.

[43] A. K. Singh, M. Shafique, A. Kumar and J. Henkel, "Resource and Throughput Aware Execution Trace Analysis for Efficient Run-Time Mapping on MPSoCs," in *IEEE Trans. on Comp.-Aid. Design of Integrated Circ. and Sys.*, vol. 35, no. 1, pp. 72-85, Jan. 2016.



Mohsen Ansari received the M.Sc. degree in computer engineering from Sharif University of Technology, Tehran, Iran, in 2016. He is currently working toward the PhD degree in computer engineering at Sharif University, Tehran, Iran, from 2016 until now. He is now the member of Embedded Systems Research Laboratory (ESR-LAB) at the department of computer engineering, Sharif University of Technology. His research interests include low-power design of embedded systems, peak power management in embedded systems, and multi-/many-core systems with a focus on dependability/reliability.

Amir Yeganeh-Khaksar is currently a M.Sc. student in the Department of Computer Engineering at Sharif University of Technology, Tehran, Iran. He received the B.Sc. degree in computer engineering from Ferdowsi University of Mashhad. His research interest lies in computer architecture, especially in Low Power Design and Embedded Systems.

Sepideh Safari received the M.Sc. degree in computer engineering from Sharif University of Technology, Tehran, Iran, in 2016. She is currently working toward the PhD degree in computer engineering at Sharif University of Technology. Her research interests include low-power design of cyber-physical systems, energy management in fault-tolerant embedded systems, and multi-/many-core systems with a focus on dependability/reliability.

Alireza Ejlali received the PhD degree in computer engineering from Sharif University of Technology in, Tehran, Iran, in 2006. He is currently an associate professor of computer engineering at Sharif University of Technology. 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. His research interests include low power design, real-time embedded systems, and fault-tolerant embedded systems.