Peak Power Management to Meet Thermal Design Power in Fault-Tolerant Embedded Systems

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

Abstract—Multicore platforms provide a great opportunity for implementation of fault-tolerance techniques to achieve high reliability in real-time embedded systems. Passive redundancy is well-suited for multicore platforms and a well-established technique to tolerate transient and permanent faults. However, it incurs significant power overheads, which go wasted in fault-free execution scenarios. Meanwhile, due to the Thermal Design Power (TDP) constraint, in some cases, it is not feasible to simultaneously power on all cores on a multicore platform. Since TDP is the maximum sustainable power that a chip can consume, violating TDP makes some cores automatically restart or significantly reduce their performance to prevent a permanent damage. This may affect timeliness of the system, and hence, designers face a challenge in deciding how to use multicore platforms in real-time embedded systems. In this paper, at first, we study how the use of passive redundancy (especially for Triple Modular redundancy) can violate TDP on multicore platforms. Then, we propose a scheme for scheduling real-time tasks in multicore systems to conquer the peak power problem in NMR systems. This is because in multicore embedded systems an efficient solution for meeting the TDP constraint is reducing the peak power consumption. The proposed scheme tries to remove overlaps of the peak power of concurrently executing tasks to keep the maximum power consumption below the chip TDP. In the proposed scheme, we devised a policy called PPA-LTF to manage peak power consumption. This policy prevents tasks execution that consume higher power according to the tasks' power traces.
Our experimental results show that our scheme provides up to 50 percent (on average by 39 percent) peak power reduction compared to state-of-the-art schemes.

Index Terms—Peak power consumption, fault tolerance, embedded systems, multicore platforms, thermal design power

1 Introduction

With the advance of VLSI technology, due to the performance and power efficiency, multicore platforms are becoming the dominant trend in embedded systems [1], [2], [3], [4]. This is the main reason for moving from single-core to multicore platforms to balance the power consumption and computation performance. Meanwhile, technology scaling has increased the number of transistors onto a multicore chip while power budget constraints restrict the design of multicore embedded systems [1], [3], [5], [6]. In spite of the high potential for fault-tolerance techniques in multicore platforms, due to the Thermal Design Power (TDP) constraint, designers of fault-tolerant embedded systems face a challenge in deciding how to use them. TDP is considered as the highest sustainable power that a chip can dissipate without triggering any performance throttling mechanisms [19], e.g., Dynamic Thermal Management (DTM) [20]. Keeping the peak power consumption below the TDP value causes that the system can execute its tasks without reducing reliability and performance. If a chip violates its TDP, it automatically restarts or significantly reduces its performance to prevent a permanent damage. Therefore, reducing the peak power consumption is the main step towards dealing with thermal constraints such as TDP [1], [3], [5]. Meanwhile, the scaling of the feature size raises the susceptibility of digital systems to transient faults [7], [8], [9], [10]. Transient faults in underlying hardware (e.g., Soft errors [11]) are the major reliability concerns in digital systems, especially due to the continuously decreasing feature size [12]. Multicore systems provide a great opportunity to implement reliability mechanisms against transient faults, such as redundant multithreading (RMT) [13], [14], process level redundancy [15] and task-level redundancy [2], [16], [17]. The task-level redundancy is a well-established technique to achieve high reliability against different fault types [2] and is well-suited for multicore platforms. On the other hand, passive redundancy performs fault masking on the basis of voting. In the passive redundancy, N copies of each module are combined as an N-modular redundant set that tolerates multiple faults [16]. The best-known example of this technique is TMR, which consists of three identical copies whose results are voted on [2].

In this paper, at first, we show how the TMR technique may increase peak power consumption and consequently may result in a chip TDP violation (see *Motivational*

- *M. Ansari, S. Safari, A. Yeganeh-Khaksar and A. Ejlali are with the Department of Computer Engineering, Sharif University of Technology, Tehran 14588, Iran.*
E-mail: {mansari, ssafari, ayeganeh}@ce.sharif.edu, ejlali@sharif.edu.
- *M. Salehi is with the University of Guilan, Rasht, Iran.*
E-mail: mohammad.salehi@guilan.ac.ir.

Manuscript received 14 Jan. 2018; revised 22 June 2018; accepted 8 July 2018. Date of publication 25 July 2018; date of current version 12 Dec. 2018.
(Corresponding author: Alireza Ejlali.)
Recommended for acceptance by Dr. B. Urgaonkar.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TPDS.2018.2858816

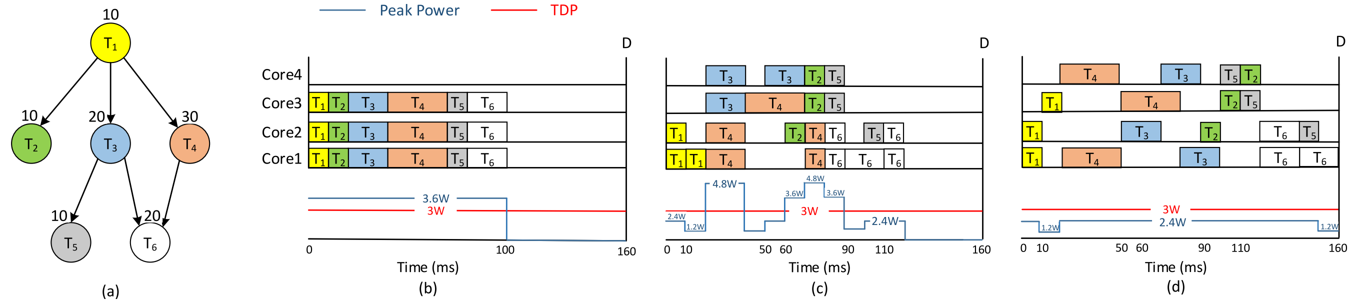


Fig. 1. Motivational example of peak power problem of a TMR system (i.e., NMR with $N = 3$) on a multicore system with 4 cores. a) An example task graph, b) Parallel execution of all tasks (the conventional triple modular redundancy), c) Delayed execution of the third copy of the tasks, d) Scheduling the tasks according to the TP3M policy (our scheme).

Example). Then, we propose a two-phase peak power management (TP3M) scheme which manages peak power consumption for the NMR technique on multicore platforms (Section 4). This scheme schedules hard real-time tasks on cores in a multicore system without violating real-time and TDP constraints. Our TP3M scheme aims at removing overlaps of the peak power of concurrently executing tasks to keep the power consumption below the chip TDP. To do this, considering the tasks' power traces, at first, we partition the tasks into parts where different parts have different peak power values (Section 4.3). Then, we have used two phases for scheduling the partitioned tasks. In these phases, we have devised a policy called PPA-LTF to reduce peak power consumption. In the first phase, the proposed scheme schedules more than half the number of copies for each task based on the Peak-Power-Aware Longest Task First (PPA-LTF) policy. When no fault occurs during this phase, the remaining copies of the tasks are not required. Otherwise, the remaining copies of the tasks are scheduled in the second phase to perform a complete majority voting. In this phase, we use PPA-LTF to manage peak power consumption. This leads to a smoothly consumed power and results in a reduced peak power. In summary, our scheme tries to separate execution of the essential tasks and the redundant tasks to remove overlaps of the peak power of them. In this scheme, the tasks that consume higher power overlap with the other tasks that consume lower power with the aim of keeping the total peak power below the chip TDP.

Motivational Example. This example provides some insight into how the different scheduling algorithms work to meet power budget (TDP). Let us consider a quad-core chip with 3W of TDP that executes an application tasks graph with six tasks $\{T_1, T_2, T_3, T_4, T_5, T_6\}$. Fig. 1a shows dependencies between the tasks where the number above each task is its worst-case execution time at the maximum supply voltage and the maximum operational frequency. The tasks share a common deadline $D = 160$ ms. For simplicity of presentation, we temporarily assume that the tasks' peak power is equal to a constant value so that each task consumes 1.2W of power during its execution. After finishing the task, the underlying core goes to sleep mode and consumes no power. In the rest of this paper, when we present our method each task consumes a different amount of power during its execution, depending on its characteristics and computational load and, different tasks have different power traces. In this example, we consider a TMR system (i.e., NMR with $N = 3$) where each task has three copies and the result of them are compared to perform a complete majority voting. Fig. 1 shows three possible schedules where meet timing constraint. One way to execute this task graph is the parallel execution of all copies of each task on three cores of the chip (the conventional triple modular redundancy), as shown in Fig. 1b. In this way, all copies of T_1 to T_6 are scheduled from $t = 0$ ms to $t = 100$ ms and all cores go to sleep mode after $t = 100$ ms (Fig. 1b). Here, the total peak power of the system is 3.6W during the time interval between 0 and 100 ms, and hence, it violates the chip TDP of 3W. Another possible execution scenario for this task set is shown in Fig. 1c where the system operates in two phases, which has been presented in [2]. At first, the system operates in its indispensable phase where two copies of each task are scheduled using list scheduling with the longest task first (LTF) policy. Then, in the conservative phase, the third copy of each task is scheduled on the schedule to obtain three results for performing a complete majority voting. This method effectively reduces energy consumption through dropping the third copy of the tasks when no fault occurs. Since this method does not consider peak power consumption, it may violate the chip TDP. As shown in Fig. 1c, in the time interval 20 ms to 40 ms and the time interval 70 ms to 80ms all the four cores are active at the same time, and hence, the total power consumption of the chip is 4.8W that is higher than the chip TDP (i.e., 3W). In Fig. 1d, a scheduling method is shown that does not violate the chip TDP. In this method, at first from the beginning of the execution frame, two copies of each task are scheduled on cores with the lowest utilization such that the peak power consumption is kept below the chip TDP. Then, for scheduling the third copy of the tasks, starting from the end of the execution of two other copies of the same task, the third copy is scheduled on a core with the lowest utilization such that the peak power consumption is kept below the chip TDP. In this execution scenario, at most time instants at most two cores are active. Since each task consumes 1.2W on each core, the maximum total power consumption in this scenario is equal to 2.4W (i.e., less than the chip TDP).

Objective. The objective of this paper is to present a Two-Phase Peak-Power Management (TP3M) scheme. TP3M is a method that uses N-modular redundancy (NMR) technique to achieve fault tolerance in real-time multicore embedded systems such that timing and TDP constraints are met. In this method, we focus on scheduling a task set in two phases. In the first phase, more than half the number of copies for each task are scheduled based on the Peak-Power-Aware Longest Task First policy. If no fault occurs during this phase, the results must be identical and hence the remaining copies are not required. Otherwise, the remaining copies must be scheduled and executed in the second phase based on PPA-LTF to perform a complete majority voting. 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 consumption and cannot provide a deterministic guarantee to keep power consumption below the chip TDP.

Our Contribution. The main contributions of this work are:

- Proposing a peak-power-aware reliability management method that manages peak power overlaps between concurrently executing tasks.
- Enabling task replication such that the system reliability is preserved while guaranteeing to keep the total power consumption of cores below the chip TDP and the power consumption of each underlying core below the core TDP constraint.
- Developing a new scheduling algorithm that avoids concurrent execution of tasks based on the peak-power-aware longest task first policy.
- Determining the voltage-frequency levels such that the tasks meet their timing constraints while keeping the total power consumption under the chip TDP at each time interval.

Evaluation. We ran simulations with gem5 [34] and McPAT [35] to compare our TP3M method with state-of-the-art methods (especially with LE-NMR presented in [2]) for the worst-case and actual-case scenarios. Our experiments show that TP3M provides up to 50 percent (on average by 39 percent) peak power reduction compared to the other schemes in the worst-case scenario. Also, TP3M provides up to 44.3 percent energy saving in the actual-case condition through canceling unnecessary execution when no fault occurs.

Organization. The remainder of this paper is organized as follows. In Section 2 we review related work. Section 3 presents models and assumptions. In Section 4, we present our TP3M scheme in details. The experimental results are presented and discussed in Section 5. Finally, we conclude the paper in Section 6.

2 Related Work

Some related works have addressed both fault tolerance and low power consumption in fault-tolerant real-time embedded systems with two processors [21], [22], [23]. To reduce the average power consumption, Ejlali et al. [21] have proposed a technique where DVS is used for the first processor (primary processor) while the second processor (spare processor) does not use DVS to preserve the reliability of the system when a fault occurs. The scheme proposed in this work is suitable for non-preemptive and aperiodic tasks while most of the real-time applications on embedded systems are inherently periodic [22]. The work in [22] 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 DVS on the primary processor, while the backup tasks are executed on the spare processor according to Earliest-Deadline-Late (EDL) scheduling. Haque et al. [23] have proposed an energy-management technique for a standby-sparing system that executes preemptive fixed-priority real-time tasks. Tasks on the primary processor are scheduled by the Cycle-Conserving DVS algorithm that has been proposed for Rate Monotonic Scheduling (RMS) in [24]. While the spare core uses DPM and dual-queue mechanism that tries to maximally delay the backup tasks to save more energy. These works have not considered multiple faults per task execution. Some research works, e.g., References [2] and [25] have proposed voltage-scaling techniques to reduce the energy consumption of N-modular redundancy (NMR). The reference [25] reduces the energy consumption of triple-modular redundancy (TMR) by exploiting voltage-scaling techniques. Salehi et al. [2] have proposed an N-modular redundancy (NMR) technique with low energy consumption for hard real-time multicore systems. All of these works have focused on reducing the average power and energy consumption and have not considered peak power management.

Some studies concentrated on thermal management in multicore systems [26], [27], [3]. Fisher et al. [26] have proposed a global thermal-aware scheduling to reduce the temperature for sporadic tasks. Jejurikar et al. [27] reduce energy consumption by using deferment interval for each task by considering real-time constraints. [3] has presented a new power budget concept, called Thermal Safe Power (TSP), which is an abstraction that provides safe power and power density constraints as a function of the number of simultaneously active cores. Some related works have focused on reducing the peak power consumption under real-time constraints [1], [5], [28]. Lee et al. [1] have proposed a new scheduling algorithm for real-time tasks to reduce chip-level peak power consumption, without relying on any extra hardware (e.g., DVFS controller). This algorithm restricts the concurrent execution of tasks that are assigned to different cores, and perform its schedulability analysis. Lee et al. [28] have proposed a task scheduling that prevents the occurrence of the peak power consumption for task-graph models. The proposed algorithm in this work schedules the tasks by considering data dependency information while reduces the peak power. As one of the most related work, Munawar et al. [5] have presented a scheme to minimize the peak power for frame-based and periodic tasks with real-time constraints on multicore systems. The reference [5] schedules the sleep cycles for each active core to manage the peak power. Pagani et al. [29] have presented a solution both for energy minimization and peak power reduction for periodic real-time tasks on multicore systems. These researches that try to reduce the peak power do not consider any fault-tolerance techniques to deal with transient and permanent faults.

Generally, the previous works in the context of multicore embedded systems either propose peak power reduction techniques without considering reliability like [1] and [5] or consider reliability without considering peak power reduction like [2], [8], [11], and [22]. In this paper, we exploit a fault-tolerance technique (N-modular redundancy) to achieve high reliability for real-time multicore systems and propose a scheme to keep the chip peak power consumption under its TDP constraint. In this section, we discussed the differences between our work and the previous works.

3 Models and Preliminaries

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

3.1 System and Application Model

This paper focuses on a multicore system with m cores $C = \{C_1, C_2, \ldots, C_m\}$ similar to Intel SCC [38]. The system executes frame-based applications with hard real-time requirements consisting of n dependent tasks $\Phi = \{\mathrm{T}_1, \mathrm{T}_2, \ldots, \mathrm{T}_n\}$. These tasks share a common global deadline D, which is also the period (or frame) of the task set [30], [31], [32]. The group of tasks is scheduled based on the precedence which modeled by a task graph. A sample task graph is composed of some nodes and vertices (see Fig. 1a). Each node in the task graph represents a task while the directed edges represent data dependencies between the tasks. The worst-case execution time for the task T_i at the maximum frequency f_{max} is denoted by W_i and has been written above each node. The utilization of a task T_i is defined as $u_i = W_i/D$. Also, the total utilization of the system U_{tot} is the sum of all the task utilizations.

3.2 Power Consumption Model

We adopt a system-level power model where total power consumption consists of a static and a dynamic component. The dynamic power P_d includes a frequency-independent power consumption and a frequency-dependent power consumption that are defined as P_{ind} and P_{dep}, respectively. P_{ind} is driven by the peripheral modules such as I/O in the activation mode. On the other hand, the static power, P_s, consists of the reverse and sub-threshold leakage power that are consumed even when no computation are carried out. Since the sub-threshold leakage and the frequency-dependent power are dominant in the static and dynamic power, respectively, the total power of each core can be written as [2], [4], [8], [11], [21], [22], [23]:

$$P = P_s + P_d = I_{sub}V + C_{eff}V_{dd}^2 f. \tag{1}$$

where C_{eff}, V_{dd}, and f are the effective switched capacitance, supply voltage and operational frequency, respectively. In this paper, we use Dynamic Power Management (DPM) to manage peak power where whenever a core is temporarily idle, it goes into sleep mode to reduce power consumption. In addition, we use Dynamic Voltage Scaling (DVS) for reducing total power consumption. When DVS is used, each task T_i is executed at a voltage V_i, which is less than V_{max} (the maximum supply voltage). By considering an almost linear relationship between voltage and frequency [2], [25], [31], when a task T_i is executed at the scaled voltage $V_i = \rho_i V_{\max}$, the operational frequency can be written as:

$$f_i = \rho_i f_{\max}, \tag{2}$$

where f_i is the operational frequency corresponding to V_i and f_{max} is the maximum operating frequency corresponding to V_{max}. Therefore, the total power consumption which is consumed to execute the task T_i is given by:

$$P = I_{sub}\rho_i V_{\max} + C_{eff}\rho_i^2 V_{\max}^2 \rho_i f_{\max} = \rho_i P_s + \rho_i^3 P_d, \tag{3}$$

where P_s and P_d are respectively the static and dynamic powers at the maximum voltage and frequency.

3.3 Fault Model and Reliability Analysis

Computer systems are susceptible to faults due to various runtime errors. Faults can be categorized into transient and permanent faults [16], [21]. Transient faults may manifest in the form of single event upset or soft errors with incorrect results. These faults are commonly caused by alpha particles and cosmic rays that strike chips in unpredictable ways. These errors are called soft because they do not lead to permanent failure. Transient faults are typically modeled using a Poisson distribution with an average arrival rate $\boldsymbol{\lambda}$ [30]. When the frequency is scaled down using DVS, the fault rate $\boldsymbol{\lambda}$ increases significantly [21]. Therefore, the fault rate at frequency f is modeled as [22], [23]:

$$\lambda(f) = \lambda_0 10^{\frac{d(1-f)}{1-f_{\min}}}, \tag{4}$$

where λ_0 is the average fault rate at the maximum frequency and the exponent d (sensitivity factor) is a measure of how the transient fault rate increases when the supply voltage and frequency are scaled [30], [31]. Considering (4) the reliability of a task T_i running at frequency f_i can be expressed as [2], [22]:

$$R_i(f_i) = e^{-\lambda(f_i)\frac{t_i}{f_i}}, \tag{5}$$

where $\lambda(f_i)$ is given by (4) and t_i is the actual execution time of the task T_i. Conversely, the probability of failure of the task T_i is given by [21]:

$$F(f_i) = 1 - R_i(f_i) = 1 - e^{-\lambda(f_i)\frac{t_i}{f_i}}. \tag{6}$$

The reliability of the proposed method is calculated by considering the two conditions, i.e., (1) the fault-free condition when all $\lceil N/2 \rceil$ copies of each task are executed correctly, (2) the faulty condition when some tasks become faulty and we require the results of the remaining $\lfloor N/2 \rfloor$ copies of tasks. Therefore, the reliability of each task in the fault-free condition can be calculated as [2]:

$$R_{fault-free}(T_i) = R_i(f_i)^{\lceil N/2 \rceil}. \tag{7}$$

where $R_i(f_i)$ is given by (5). When up to $\lfloor N/2 \rfloor$ copies of a task T_i become faulty, the reliability of a task T_i can be calculated as [2], [17]:

$$\begin{aligned} R_{faulty}(T_i) = {} & P(k \leq \lfloor N/2 \rfloor) = \sum_{l=1}^{\lfloor N/2 \rfloor} P(\#\ of\ faults = k) \\ & = \sum_{l=1}^{\lfloor N/2 \rfloor} \binom{N}{l} F_i(f_i)^l R_i(f_i)^{N-l}, \end{aligned} \tag{8}$$

where $F_i(f_i)$ is given by (6) and k is the number of faults. According to Eqs (7) and (8), the reliability of a task T_i in both the fault-free and faulty conditions can be written as:

$$\begin{aligned} R_{total}(T_i) &= R_{fault-free}(T_i) + R_{faulty}(T_i) \\ &= R_i(f_i)^{\lceil N/2 \rceil} + \sum_{l=1}^{\lfloor N/2 \rfloor} \binom{N}{l} F_i(f_i)^l R_i(f_i)^{N-l}. \end{aligned} \tag{9}$$

Generally, the reliability of a system with n tasks running by our proposed method can be calculated as:

$$R_{system} = \prod_{i=1}^{n} R_{total}(T_i). \tag{10}$$

4 Our Proposed Method

In this section, at first, we represent a high-level overview of our TP3M system. Then, in Section 4.2, we define the problem of the task scheduling and mapping on a multicore system. In Section 4.3, we explain our proposed Two-Phase Peak-Power

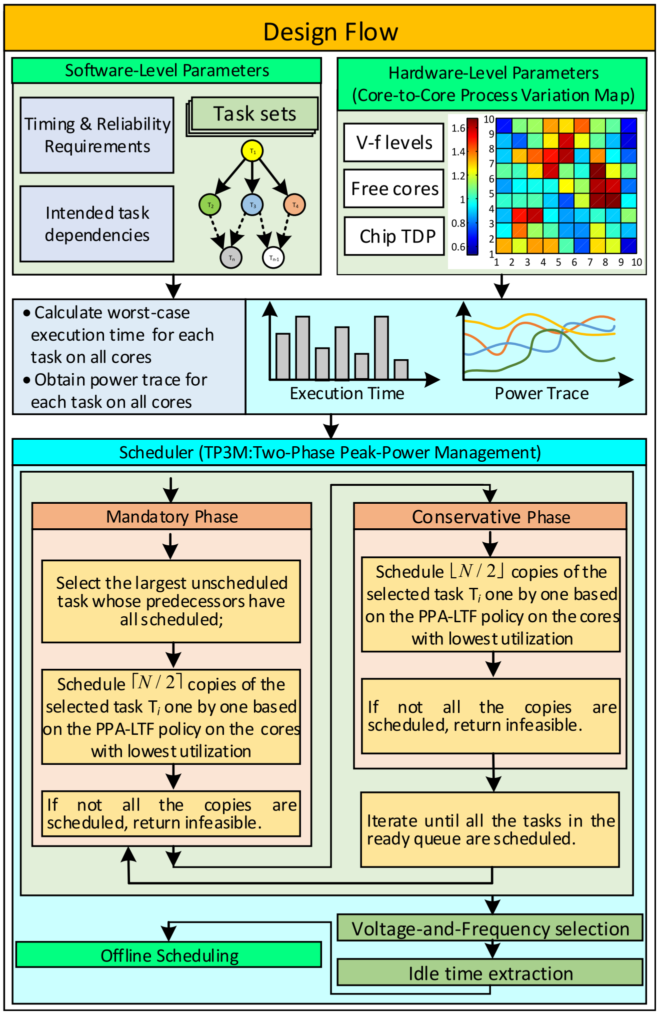

Design Flow
Software-Level Parameters
Timing & Reliability Requirements
Task sets
Intended task dependencies
T1
T2
T3
T4
T5
Tn-1
Hardware-Level Parameters (Core-to-Core Process Variation Map)
V-f levels
Free cores
Chip TDP
1.6
1.4
1.2
1
0.8
0.6
1 2 3 4 5 6 7 8 9 10
• Calculate worst-case execution time for each task on all cores
• Obtain power trace for each task on all cores
Execution Time
Power Trace
Scheduler (TP3M:Two-Phase Peak-Power Management)
Mandatory Phase
Select the largest unscheduled task whose predecessors have all scheduled;
Schedule ⌈N / 2⌉ copies of the selected task Ti one by one based on the PPA-LTF policy on the cores with lowest utilization
If not all the copies are scheduled, return infeasible.
Conservative Phase
Schedule ⌊N / 2⌋ copies of the selected task Ti one by one based on the PPA-LTF policy on the cores with lowest utilization
If not all the copies are scheduled, return infeasible.
Iterate until all the tasks in the ready queue are scheduled.
Voltage-and-Frequency selection
Idle time extraction
Offline Scheduling

Fig. 2. The operational flow of our TP3M system.

Management (TP3M) scheme, and in Section 4.4 we use an example to illustrate how our proposed scheme works.

4.1 System Overview

To fulfill the objective of the paper, we propose a peak-power management scheme that provides a right design of the system and ensures that the power dissipation of cores meets peak power constraints. Based on a chip-level power constraint, the proposed scheme parcels this power budget into local power budgets for each core. The local power budget is calculated by determining the worst-case mapping of cores. These local power budgets can be used as power constraints for any possible mapping of cores and, as a result, DTM is not triggered at any time. As different application tasks have different power traces, reliability, failure rates, and execution time properties, the proposed method considers all of them. Our proposed Two-Phase Peak-Power Management (TP3M) scheme consists of an offline part and an online part that are explained in Section 4.3. Also, the TP3M scheme consists of the mandatory and conservative phases to take the advantages of fault-free scenario. In the offline phase, after scheduling the tasks, we apply two different techniques, DVFS and DPM, to reduce peak power dissipation and average power consumption. For the sake of completeness, in the mandatory phase, only half-plus-one copies of each task are scheduled according to the PPA-LTF policy and the remaining copies must be scheduled in the conservative phase according to the PPA-LTF policy to perform a complete majority voting. Meanwhile, for the proposed scheme, we have exploited different types of slack time to reduce the average power consumption through DVS. Algorithm 1 in Section 4.3 shows the pseudo code of the task scheduling mechanism of our TP3M scheme. Also, Fig. 2 shows the overview of our TP3M system along with different inputs from the hardware and software components. In the following (Section 4.4), we use an example to illustrate how our proposed scheme works.

4.2 Problem Definition

In the following, we define the problem of the task scheduling and mapping on a multicore system when exploiting the NMR technique and different V-f levels for different application tasks such that the peak power consumption is kept below the chip TDP. To do this, we use the following notation to represent the peak power consumption, the reliability of the system, V-f levels, and task mapping. In the problem formulation, n is the number of tasks, m is the number of available and free cores and l is the number of V-f levels for each core. Table 1 shows the notation used for variables throughout this section.

- The peak power consumption is represented by the matrix $PPC \in \Re^{n \times m \times l \times t}$, in which PPC_{ijkh} is the peak power consumption for the task i when is executed on the core j under the V-f level k at time h.
- The V-f level assignment and task mapping are represented by the matrix $Y \in \{0, 1\}^{n \times m \times l}$, in which the task i is mapped to the core j and is executed under the V-f level k if and only if Y_{ijk} is equal to 1.
- The reliability of the system is represented by the matrix $R \in \Re^{n \times m \times l}$, in which each element R_{ijk} denotes the reliability of task i when is executed on the core j under the V-f level k.

Besides the technique introduced in this paper, we formulate the peak-power-aware scheduling problem as a constrained 0-1 integer linear program (ILP). Since the goal of the paper is to meet reliability, TDP and timing constraints in a multicore embedded system that consists of a set of tasks with different arrival times and deadlines, the objective function should be the constant value.

TABLE 1
The Notation of the Parameters

Notation	Description
PPC	Peak power consumption matrix
PPC_{ijkh}	The peak power consumption for task i when is executed on the core j under the V-f level k at time h
$P_{TDP,Chip}$	Chip TDP constraint
$P_{TDP,k}$	Core TDP constraint
Y	The V-f level assignment and task mapping matrix
Y_{ijk}	The task i mapped to the core j and executed under the V-f level k
R	Reliability matrix
R_{ijk}	The reliability of the task i when is executed on the core j under the V-f level k
R_{req}	Reliability requirement

$$\begin{aligned}&\text{Minimize } F(x) = 1\\ &\forall x \in \Re.\end{aligned} \tag{11}$$

The total power consumption (i.e., the sum of the instantaneous power of all underlying cores) should be less than the chip TDP constraint at each time interval. Furthermore, the peak power of each underlying core should be less than the core TDP constraint.

$$\sum_{i,j,k,l} Y_{ijkh} PPC_{ijkh} \leq P_{TDP,chip} \tag{12}$$

$$Y_{ijkh} PPC_{ijkh} \leq P_{TDP,k}. \tag{13}$$

For timing constraints, the worst-case execution time W_i/f_{jk} for the task i on the core j and at the V-f level k should not exceed the task timing constraint (defined by the D_i).

$$Y_{ijkh} \frac{W_i}{f_{jk}} \leq D_i. \tag{14}$$

The system reliability is defined by the successful execution of all tasks. Therefore, the reliability mechanism satisfies the reliability requirement R_{req} when:

$$\prod_{i,j,k} X_{i,j,k} R_{i,j,k} \geq R_{req}. \tag{15}$$

Also, each task can be only mapped to a single core.

$$\forall i,j,l : \sum_{h} Y_{ijkh} = 1. \tag{16}$$

The formulated problem is usually classified as an NP-complete problem [1], [5], and hence we use a heuristic method.

4.3 Algorithm Discussion

Our proposed Two-Phase Peak-Power Management (TP3M) scheme consists of the offline part and online parts that are explained as follows. Meanwhile, the offline part consists of two phases: mandatory and conservative phases. In the offline phase, after scheduling the tasks, we apply two different techniques, DVFS and DPM, to reduce peak power dissipation and average power consumption.

Algorithm 1 shows the pseudo code of the task scheduling mechanism of our TP3M scheme that receives an application task graph (Φ) to make schedules for the mandatory and conservative phases. At first, we determine the size of the time slots in line 1. For this purpose, the execution frame is divided into $h = D/CLK\_C$ slots (*CLK_C* is the clock cycle time). In this algorithm, we use a peak power array including h slots that determines the peak power consumption of the system in each time slot (i.e., the list *PPL* in line 2). In line 3 to 5, we partition all the tasks into parts with different peak power values. Here, the number of parts of the task T$_i$ is W_i/CLK_C. In line 6, the algorithm initializes a schedule S_i to *Null* for each core of *C* (*C* is the set of cores). Next, the algorithm iterates until all the tasks are selected (lines 7-61). In line 8, we select the largest unscheduled task T$_i$ whose predecessors have all been scheduled. We use the variable q to count the number of scheduled copies of each task and make the temporary set of available cores *MC* (lines 9 and 10). In line 11 to 31, the algorithm iterates until $\lceil N/2 \rceil$ copies of each task are scheduled based on the PPA-LTF policy. In order to provide core usage efficiency, we select a core with the lowest utilization to schedule the selected task on it (denoted by φ in line 12). We use the variable k to determine where the current part of T$_i$ (T_{ij}) can be placed. The variable k is initialized to the first free time slot after which all predecessors of the selected task have scheduled (line 13). Now, starting from k, we check free time slots of the core φ one after another and place each part T$_{ij}$ on the first free time slot t ($t = k \rightarrow h$) such that the peak power consumption of T$_{ij}$ does not exceed the core TDP constraint and also does not increase the total power consumption beyond the chip TDP. Therefore, we place T$_{ij}$ in t^{th} time slot of *φ.S* in line 18 and update the power consumption list *PPL* in line 19, and update the variable k in line 20. Otherwise, if the core TDP constraint (*φ.TDP*) is not met, the scheduled parts of the selected task T$_i$ are deleted from *φ.S* and the selected core φ is removed from *MC*, then, the algorithm goes back to line 12 to try again for scheduling the selected task T$_i$ on another core. If not all the copies are scheduled in the mandatory phase, the algorithm returns *infeasible* in line 33. After the mandatory phase, we schedule $\lfloor N/2 \rfloor$ copies of the tasks based on the PPA-LTF policy in the conservative phase. In line 37 to 57, the algorithm iterates until $\lfloor N/2 \rfloor$ copies of each task are scheduled based on the PPA-LTF policy. To do this, we choose a core with the lowest utilization to increase the efficiency of the cores. In this phase, the variable k is initialized to the first free time slot after the finish time of the last copy of the selected task in the mandatory phase (line 39). We place the parts of the tasks, beginning from the first part, on time slots that come sooner in the schedule *φ.S*. Then, we check free time slots of the core φ one after another and place each part T$_{ij}$ on the first free time slot t ($t = k \rightarrow h$) such that the peak power consumption of T$_{ij}$ does not exceed the core TDP constraint and also does not increase the total power consumption beyond the chip TDP (lines 40-43). In this case, T$_{ij}$ is placed in the time slot t of *φ.S* in line 44. The power consumption list *PPL* is updated in line 45 and the variable k is updated in line 46. If the core TDP constraint (*φ.TDP*) is violated, the scheduled parts of the selected task T$_i$ are deleted from *φ.S* and the selected core φ is removed from *MC*, then, the algorithm goes back to line 38 to choose another core. It should be noted that when the current part of the selected task T$_i$ is placed on a time slot t, the next part T_{ij+1} can be placed on the next time slot starting from $t + 1$. Finally, if not all the copies are scheduled, the algorithm returns *infeasible* in line 59.

As explained later, when no fault occurs, we do not execute $\lfloor N/2 \rfloor$ copies for each task, which results in considerable power saving as compared with conventional NMR. In the following, we explain how the proposed method exploits static and dynamic slack times to reduce power consumption. The dynamic slack is created when $\lceil N/2 \rceil$ copies of a task are successfully finished during the mandatory phase; therefore, the additional $\lfloor N/2 \rfloor$ copies of the task are not executed in the conservative phase. In the offline part, we assume that no dynamic slack time exists because the amount of dynamic slack times is unknown at the design time. The challenge of the offline phase is to make appropriate decisions to guarantee the timing requirements while also considering the chip TDP in a system. As each part of the task T$_{ij}$ is executed at the scaled supply voltage $\rho_{ij} V_{\max}$, its worst-case execution time of each part increases from W_{ij} to W_{ij}/ρ_{ij}. Since the power trace of a task depends on the supply voltage, operational frequency, and input data switching activity, the power traces of the tasks are changed whenever DVFS is used. For applying DVFS, we exploit free time slots on all the schedules such that the power constraints are met. When a part of a task T_{ij} is executed at the scaled voltage $V_{ij} = \rho_{ij} V_{max}$, considering a linear relationship between voltage and frequency, we have: $f_{ij} = \rho_{ij} f_{max}$, where f_{ij} is the operational frequency corresponding to V_{ij} and f_{max} is the maximum operational frequency. Therefore, the execution time of each part of the task T_{ij} is lengthened from W_{ij} to $W_{ij}/\boldsymbol{\rho}_{ij}$, and by considering $V_{ij} = \rho_{ij} V_{max}$ and $f_{ij} = \rho_{ij} f_{max}$, the total power dissipation which is consumed to execute each part of the task is given by Eq. 3. But, when the execution time of a part T_{ij} is lengthened from W_{ij} to W_{ij}/ρ_{ij}, this part may overlap with other tasks that have high power consumption. In this case, if the chip TDP constraint is violated, we do not use DVFS and reduce peak power and average power consumptions through DPM. In order to apply DPM, let assume we have a break to sleep time ($t_{critical}$). Therefore, if the idle time of a core is greater than $t_{critical}$, the core switches to sleep mode. To identify when to apply DVFS and DPM in the above discussion, core utilization is often used to assist the determination of the use of the techniques. The core utilization refers to the fraction of the time that the core spends non-idle. When the core utilization is low and slack time is greater than $t_{critical}$, we can use DPM. If slack time is less than $t_{critical}$, we try to apply DVFS such that the chip TDP and core TDP constraints are met.

Algorithm 1. The task scheduling mechanism of our TP3M scheme

Inputs: Φ: Application task graph, D: Deadline, N: Parameter N of NMR, C: Set of cores, Available V-f levels for each core, Tasks' power trace, and Chip TDP and Core TDP constraints.
Output: The task scheduling S_i on each core C_j.

```
BEGIN:
1:  $h = D/CLK\_C$; //Total # of time slots in the frame
2:  $PPL[1 \ldots h] = \{0\}$; //Initialize the total power consumption array
3:  for all tasks in $\Phi$ do
4:        $T_i = \{T_{ij}, 1 \leq j \leq W_i/CLK\_C\}$; //partition all the tasks into parts
5:  end for;
6:     $S_i = \{Null, 1 \leq i \leq m\}$; //Initialize $S$ with an empty schedule
7:  while (all tasks in $\Phi$ are not selected) do
8:     $T_i = \Phi.select()$; //Select the largest unscheduled task whose predecessors have all scheduled;
9:  $q = 1$;
10: $MC = C$;
–   //Mandatory phase: Schedule $\lceil N/2 \rceil$ copies
11: while ($q \leq \lceil N/2 \rceil \& MC \neq \phi$) do
12: $\varphi = MC.min_{utilization}$;
13: $k = Finish\_time\_of\_predecessor(T_i)$;
14: foreach part $T_{ij}$ starting from the first part do
15:    foreach free slot $t = k \rightarrow h$ in $\varphi.S$ do
16:       if $PPL[t] + peak\_power(T_{ij}) \leq Chip\_TDP$ then
17:          if $peak\_power(T_{ij}) \leq \varphi.TDP$ then
18:             $\varphi.S.add(t, T_{ij})$;
19:             $PPL[t] = PPL[t] + peak\_power(T_{ij})$;
20:             $k = t + 1$;
21:             break;
22:       else
23:             $\varphi.S.delete(T_i)$; //Delete the task $T_i$ from $\varphi.S$
24:             $MC.remove(\varphi)$; //Remove $\varphi$ from $MC$ for the task $T_i$
25:             goto line 12;
26:          end if;
27:       end if;
28:    end for;
29:  end for;
30:  $q = q + 1$;
31: end while;
32: if not all the copies are scheduled then
33:    return infeasible;
34: end if;
35: $q = 1$;
36: $MC = C$;
–   //Conservative phase: Schedule $\lfloor N/2 \rfloor$ copies
37:    while ($q \leq \lfloor N/2 \rfloor$ & $MC \neq \phi$) do
38:    $\varphi = MC.min_{utilization}$;
39:      $k = Finish\_time\_of\_last\_copy(T_i)$; //Last copy in the Mandatory phase
40:    foreach part $T_{ij}$ starting from the first part do
41:       foreach free slot $t = k \rightarrow h$ in $\varphi.S$ do
42:          if $PPL[t] + peak\_power(T_{ij}) \leq Chip\_TDP$ then
43:             if $peak\_power(T_{ij}) \leq \varphi.TDP$ then
44:                $\varphi.S.add(t, T_{ij})$;
45:                $PPL[t] = PPL[t] + peak\_power(T_{ij})$;
46:                $k = t + 1$;
47:                break;
48:             else
49:                $\varphi.S.delete(T_i)$; //Delete the task $T_i$ from $\varphi.S$
50:                $MC.remove(\varphi)$; //Remove $\varphi$ from $MC$ for the task $T_i$
51:                goto line 36;
52:             end if;
53:          end if;
54:       end for;
55:    end for;
56:    $q = q + 1$;
57: end while;
58: if not all the copies are scheduled then
59:       return infeasible;
60:  end if;
61: end while;
END
```

As the final discussion of this section, we discuss 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 [1], [2], and [5]. For example, in the motivational example, the proposed scheme in [2] (Fig. 1c) schedules tasks before $t = 120$ ms, however, it violates TDP in several time slots. Therefore, for meeting TDP, timing and reliability constraints simultaneously, we must consider more time slots.

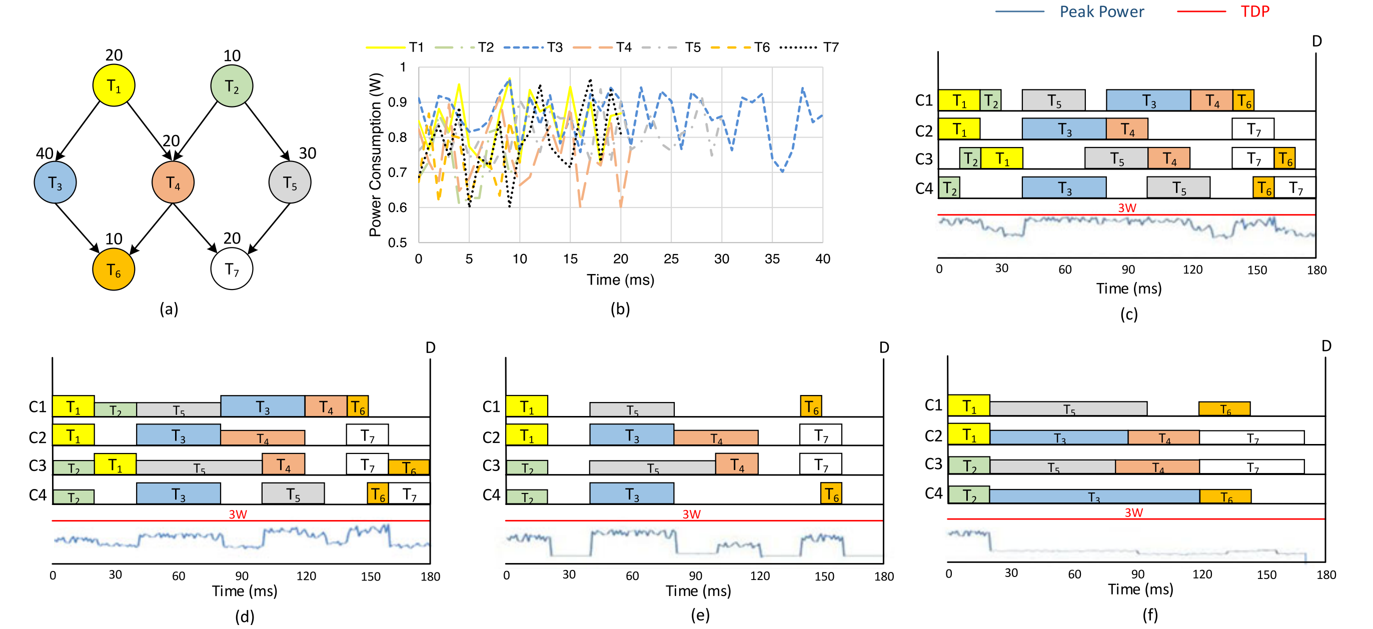


Fig. 3. An example of how our proposed method works on a multicore system with 4 cores. a) An example task graph, b) Power traces of the tasks, c) the schedule with two copies of each task for the mandatory phase and one copy for the conservative phase, d) Final schedule for the worst-case scenario with applying DVS, e) Taking advantage of the fault-free execution, f) Applying DVS and DPM in the fault-free scenario.

4.4 An Illustrative Example

In the following, using an example we illustrate how the algorithm works. For simplicity in presentation, this example is considered to show the effectiveness of our scheme, however, the proposed scheme works for more complex and larger task graphs. Let us consider a quad-core chip with 3W of TDP that executes an application tasks graph with 7 tasks $\{T_1, T_2, T_3, T_4, T_5, T_6, T_7\}$. In this example, we consider a TMR system (i.e., NMR with $N = 3$). Fig. 3 shows the step by step generation of our proposed schedule for a given task graph (Fig. 3a) using list scheduling with our proposed policy. Fig. 3a shows dependencies between the tasks where the number placed above each task is its worst-case execution time at the maximum supply voltage and the maximum operational frequency. These tasks share a common deadline $D = 180$ ms. At first, by the use of the tasks' power traces in Fig. 3b, we determine the peak power values for the different parts of the tasks. Fig. 3c shows the schedule with two copies of each task for the mandatory phase and with one copy of each task for the conservative phase. Based on the lowest utilization first policy, in the mandatory phase, two copies of the task T_1 are respectively scheduled on C1 and C2 from the beginning of the execution frame of their designated core. In the conservative phase, the third copy of T_1 is placed in the time slots between 20 ms and 40 ms on the schedule of C3. Then, based on the level-based longest task first policy, we select T_2 and schedule two copies of T_2 on C4 and C3, respectively, such that the chip TDP and the core TDP constraints are met. The third copy of T_2 is mapped to C1 and is scheduled in the time slots between 20 ms and 30 ms. For simplicity, it is assumed that the tasks are not partitioned into parts and the task T_i has only one part. For the next selected task T_3, we map two copies of T_3 to C2 and C4 separately and schedule T_3 in the time slots between 40 ms and 80 ms on the schedule of C2 and C4. After scheduling two copies of T_3, we schedule another copy of T_3 in the time slots between 80 ms and 120 ms on the schedule of C1 to obtain three results for performing a complete majority voting. Then, the algorithm selects T_5 and schedules respectively two copies of it on C1 and C3 in the time slot [40 ms, 70 ms] and the time slot [70 ms, 100 ms], respectively. Of course, we can schedule the second copy of T_5 in the time slots between 40 ms and 70 ms on the schedule of C3, but scheduling T_5 in these time slots can result in the chip TDP violation. For the next selected task T_4, we place two copies of this task after execution of T_3 and T_5 on C2 and C3, respectively. Of course, we can schedule the second copy of T_4 in the time slots between 40 ms and 70 ms on the schedule of C3, but scheduling T_4 in these time slots can result in the chip TDP violation. The third copy of T_4 is scheduled on the schedule of C1 in the time slots between 120ms and 140 ms after the execution of the second copy. To schedule the tasks of the third level of the graph, at first, we select T_7 and schedule two copies of it in the time slots between 140ms and 160ms on C2 and C3, respectively. After scheduling the two copies, we schedule another copy of T_7 on the core with lowest utilization C4 in the time slots between 160ms and 180ms. For the next selected task T_6, we place two copies of it on C1 and C4 in the time slot [140 ms, 150 ms] and [150 ms, 160 ms], respectively. Of course, we can schedule the second copy of T_5 in the time slots between 140 ms and 150 ms on the schedule of C4, but scheduling T_6 in these time slots can result in the chip TDP violation. Finally, we schedule the third copy of T_6 in the time slot [160 ms, 170 ms] such that the power constraints are met. Fig. 3c shows the final schedule where the peak power consumption of the system is kept below the chip TDP constraint. For applying DVS and DPM, we select tasks that can exploit slack times to achieve even further power reduction. For this purpose, we allocate static slack times to tasks at design time. When we allocate static slack times, we assume that no dynamic slack exists, as the availability and the amount of dynamic slack times are not known at design time. However, at run-time, we also exploit dynamic slacks through our online power management for further

TABLE 2
The Details of Simulation Configuration

Processor		Single-core, five different voltage and frequency levels between [0.85 Volt, 1 GHz] and [1.1 Volt, 2 GHz].
Memory	Main Memory	4 GB, 1 channel, 2 ranks, 8 banks per rank, Access time: 100 cycles, DRAM
	L1	32 KB, 8 KB block-width, 4-way, Access time: 2 cycles, SRAM
	L2	1 MB, 16-way, 64 B line, Write back, write: 20 cycles, STT-RAM

power reduction. In this example, to meeting task precedence constraints, we apply the DVS technique to some tasks such as T_2, T_4, T_5, and T_6 (see Fig. 3d). In Fig. 3, the blue straight line shows the real power values at run-time.

As we explained earlier, dynamic slack may create at run-time due to correct execution and early completion of tasks. As the actual execution time of a task is unknown at design time, the amount of the dynamic slack time is also not known. Meanwhile, when during the execution of tasks no fault occurs, the execution of the tasks in the conservative phase can be canceled. Fig. 3e shows the case where no fault occurs during the execution of the tasks in the mandatory phase and causes the tasks in the conservative phase are dropped because when $\lceil N/2 \rceil$ copies of a task are successfully finished during the mandatory phase, the additional $\lfloor N/2 \rfloor$ copies of the task are not required. Therefore, we drop all tasks of the conservative phase to reduce further power consumption. For instance, in this example, we drop the third copy of the tasks T_2, T_3 and T_4 from the schedule of C1, the tasks T_1 and T_6 from the schedule of C3 and the task T_5 from the schedule of C4 (see Fig. 3e). For the use of the dynamic slack, we apply the DVFS technique to tasks that meet task precedence constraints after applying DVFS. In this example, we apply DVFS to some tasks from the schedule of all the cores (see Fig. 3f).

5 Results and Discussion

In this section, we evaluate the effectiveness of our proposed method via simulation with various task sets including real-life embedded applications of MiBench Benchmark suite [33] running on a target multicore chip. At first, we describe how to generate the task sets and obtain the tasks' power traces. Then we demonstrate substantial quantitative improvements by the proposed method.

5.1 Experimental Setup

In order to evaluate TP3M, we use gem5 full-system simulator [34]. 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 study. ARM processors adopt a RISC architecture where only load/store instructions are allowed to access the memory. Each component of this processor is characterized by its static and dynamic power consumption. Meanwhile, we considered that the system supports DVS and can work at five different voltage and frequency levels between [0.85 Volt, 1 GHz] and [1.1 Volt, 2 GHz]. The details of simulation configurations for a single core system are summarized in Table 2. This ARM core has an area of 9.74 mm^2 with 32KB L1 cache and a shared 1 MB L2 cache.

To determine the peak power consumption, we ran several embedded benchmark applications with their various inputs in MiBench benchmark suites on gem5 [34] and McPAT [35]. For each task, 100 inputs were randomly generated as task inputs to obtain the power traces on different executions. Also, the applications were selected such that they introduce a variety of values for the simulation parameters, i.e., execution time, min/max power consumption, energy consumption (see Table 3). Based on the peak power values shown in Table 3, we set the peak power consumption of each task between the minimum and maximum values of this table.

Previous work has studied reliability and energy issues in embedded systems, but they do not consider peak power management, therefore, we focus on peak power reduction such that preserve the reliability of the system at an acceptable level. To the best of our knowledge, this paper is the first attempt that addresses peak-power management and fault-tolerance in conjunction. Therefore, we compare our method with state-of-the-art power management techniques. The comparison partners in our evaluations are:

- LE-NMR: An implementation of the technique that was presented in [2]. This technique proposed an N-modular redundancy with low energy overhead for multicore embedded systems. This technique executes the tasks in two phases: the indispensable phase and on-demand phase. In this paper, when a task has no faulty during the indispensable phase, the time which is reserved for its copies in the on-demand phase is reclaimed to significantly reduce energy. We chose LE-NMR to highlight the important differences between

TABLE 3
Characteristics of the Benchmark Applications

		BITCOUNT	SUSAN	MATH	CRC32	SHA	QSORT	JPEG	FFT	DIJKSTRA	LAME	GSM
Execution time (ms)		193.15	118.09	1098.40	2078.51	39.36	206.82	47.89	960.88	89.90	3055.44	704.46
Energy consumption (mJ)		112.21	67.95	604.26	1107.95	22.51	120.18	29.44	554.07	56.59	1925.32	409.51
Min. Power (mW)	Dynamic	255.83	272.51	253.19	217.18	272.74	197.61	281.73	252.22	288.13	282.229	282.34
	Static	293.327										
	Total	**549.16**	**565.83**	**546.52**	**510.51**	**566.07**	**490.93**	**575.05**	**545.55**	**581.45**	**575.56**	**575.67**
Max. Power (mW)	Dynamic	576.54	562.61	473.69	431.94	515.79	479.84	536.81	494.01	431.47	458.52	437.3
	Static	293.327										
	Total	**869.87**	**855.94**	**767.01**	**725.27**	**809.12**	**773.17**	**830.14**	**787.33**	**724.80**	**751.85**	**730.63**

peak-power management and energy management. Another reason to select LE-NMR for the comparison is that it is a recent work with similar situations to our proposed method. Our implantation of the technique in [2] only considers the two-phase execution of tasks along with a simple DVS technique and the DVS optimization proposed in [2] is not considered in this implementation.

- RAPM: This technique is proposed in [30]. For the fair comparison, we assumed that RAPM uses *N*-1 backup tasks for each task to achieve fault tolerance. This technique proposed both individual-recovery and shared-recovery based reliability-aware power management heuristics.
- CNMR: The conventional NMR scheme, called CNMR, we consider that each task has *N*-1 copies. All *N* copies of each task are executed in parallel (see Fig. 1b). The CNMR technique only uses the static slack time to reduce average power consumption.
- [5]-NMR: This technique is proposed in [5], and we have assumed that [5] uses *N*-1 backup tasks for each task to achieve fault tolerance. This technique is proposed by scheduling the sleep cycles for each core to reduce peak power consumption.

To compare TP3M with state-of-the-art power management techniques, we used both synthetic and practical application task graphs. In order to cover both synthetic and practical application task graphs, we used the Graphs For Free (TGFF) task graph generator [36] and the Standard Task Graph set (STG) [37]. The tasks of synthetic task graphs were randomly selected from the MiBench benchmark and the different parameters of the selected tasks were taken from Table 3. In the experiments, task graphs with 10, 50, 100, 500 tasks were considered that each task graph has different parallelism degree [2]. In our experiments, we considered three classes of task graphs with different parallelism degrees (like the work [2]). We need the mentioned parameter to analyze the effectiveness of our technique. It is known the height of the task graph can be used to take the parallelism degree for task graphs with the specific number of tasks [2]. Based on it, in the experiments, three classes of task graphs with different parallelism degrees are considered. If *n* be the number of tasks in a task graph and *h* be the task graph height, *h* can vary between 1 (the highest parallelism degree) and *n* (a chained task graph with the lowest parallelism degree). Therefore, in the experiments, we consider the following classes: *i*) task graphs with high parallelism degree that the height of them is $1 \leq h \leq n/3$, *ii*) task graphs with medium parallelism degree that the height of them is $n/3 \leq h \leq 2n/3$, and *iii*) task graphs with low parallelism degree that the height of them is $2n/3 \leq h \leq n$. We also considered a different number of cores in our experiments. We conducted experiments on chips with 4, 8, 12 and 16 cores. We compared TP3M with the four selected schemes (LE-NMR, RAPM, CNMR, and [5]-NMR) for: *i*) the worst-case scenario when the system consumes the maximum possible power (Section 5.2) and *ii*) the average-case scenario including both faulty and fault-free scenarios when the system consumes real power (Section 5.3).

5.2 Worst-Case Analysis

The worst-case scenario determines the maximum power consumption by the system because all *N* copies of each task are executed in this scenario ($N = 5$ in this subsection). Therefore, it can be considered a good condition for comparing peak power and average power management techniques. Fig. 4 shows the power consumption of our TP3M scheme and the LE-NMR technique for different parallelism degree with 100 tasks on a 4-core system. This figure shows that TP3M consumes less peak power than LE-NMR because TP3M distributes power consumption over the whole execution frame. In this figure, the dashed line is the TDP constraint. As Fig. 4 shows, the LE-NMR technique misses this TDP constraint. In Fig. 4, we have used only one random task set for each system configuration. To provide a more detailed analysis, for each system configuration, we used more task sets and then the average results are shown in Fig. 5. Each case of this figure was simulated for 1000 times with different parameters of the applications (i.e., tasks' worst-case and actual execution times and application deadline) and the average results are reported. This figure shows the maximum peak power consumption for TP3M, LE-TMR, RAPM, CNMR, and [5]-NMR. From Fig. 5 it can be concluded that:

- When the number of cores increases, the peak power reduction of TP3M is higher than other schemes. In this case, TP3M provides up to 45.3, 47.6, 50 and 27.5 percent peak power reduction as compared to the LE-NMR, RAPM, CNMR, and [5]-NMR schemes, respectively.
- It can be seen from Fig. 5 that, when the parallelism degree of task graphs increases, the peak power consumption of TP3M do not increase, while other

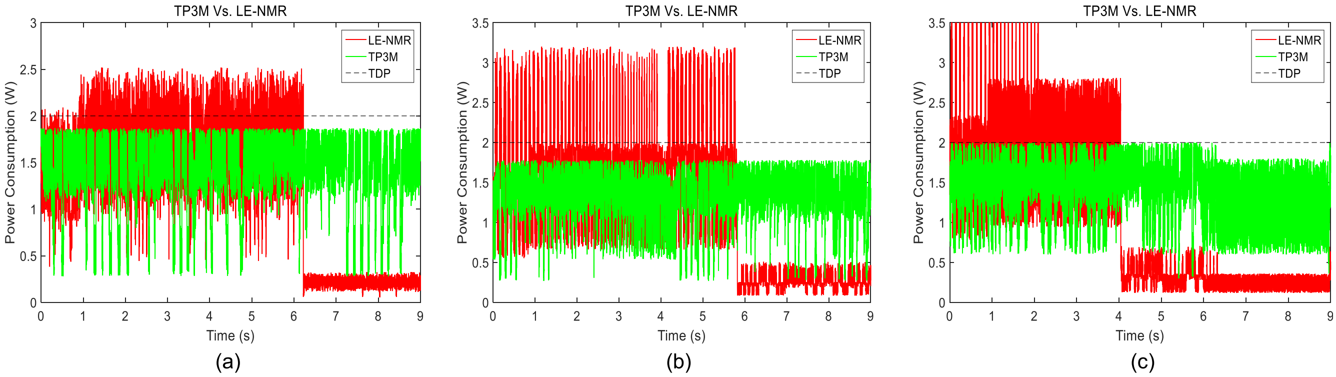
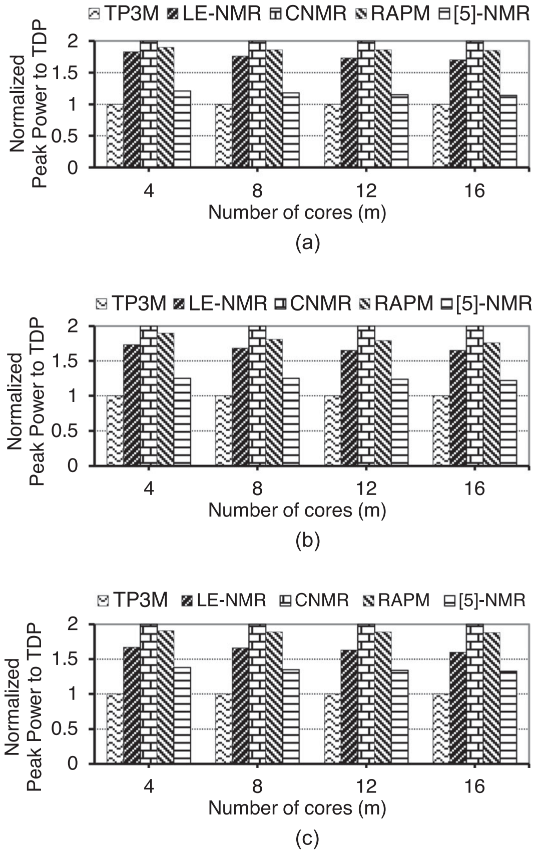


Fig. 4. Power consumption profile in the worst-case scenario on a 4-core system, a) Low parallelism degree, b) Medium parallelism degree, c) High parallelism degree.



(a)

(b)

(c)

Fig. 5. Normalized peak power to the chip TDP in the worst-case scenario. a) Low parallelism degree, b) Medium parallelism degree, c) High parallelism degree.

schemes increase it. The peak power consumption of TP3M is always less than the other four systems.

- When the parallelism degree of task graphs increases, the difference between the peak power consumption of TP3M and the four schemes increase. In this case, the effectiveness of our TP3M scheme than all the other schemes has been demonstrated.

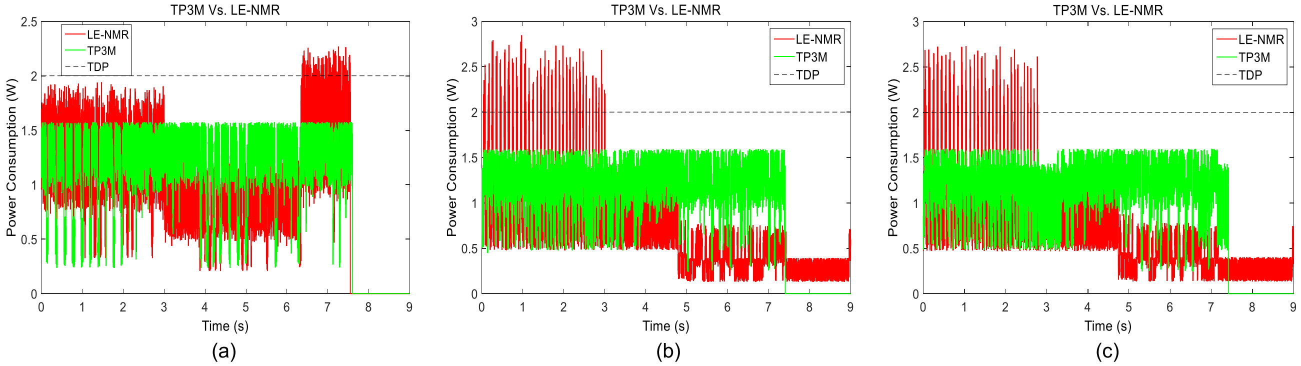
We also compared TP3M with $N = 3$ and $N = 7$ with LE-NMR, RAPM, CNMR, and [5]-NMR. The experiments demonstrate that TP3M completely outperforms the four schemes from the peak power consumption viewpoint. The TP3M method with $N = 3$ and $N = 7$ provides on average respectively 40.1 percent (up to 50 percent) and 38.7 percent (up to 50 percent) peak power reduction as compared to the four schemes.

5.3 Actual-Case Analysis

In this case, we investigate the actual conditions 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 (4) under the parameters $\lambda_0 = 10^{-6}$ faults/us and $d = 2$ [22]. Therefore, the fault rate varies between 10^{-6} faults/us corresponding to $f_{\max}$ and 10^{-2} faults/us corresponding to $f_{\min}$. Therefore, 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. Since transient faults are rare in nature, our TP3M scheme achieves further power reduction at runtime beyond what is achieved through the offline part of TP3M at design-time. Meanwhile, when a task T$_i$ is executed successfully in the mandatory phase at runtime and is dropped its copies from the schedule of the conservative phase, the dynamic slack time is released that can be exploited by DVS to reduce the power consumption of the tasks in the mandatory phase at runtime. Fig. 6 shows the power consumption trace of the execution task sets that were deployed in Fig. 4 where some tasks may become faulty. Like the worst-case scenario, in this case, TP3M consumes less power than LE-NMR due to its different schedule and better peak power management scheme. Also, TP3M distributes power consumption over the whole execution frame and reduces peak power over time. It can be seen from Fig. 6 that both the schemes consume no power at the end of the execution frame. This is because $\lceil N/2 \rceil$ copies of each task may have already finished successfully (when no fault occurs) and $\lfloor N/2 \rfloor$ copies of the tasks are canceled. Therefore, considering that the fault rate is low, almost always at the end of each execution frame, there is no task to be executed and the underlying cores go to sleep mode and consume no power.

Fig. 7 shows the energy consumption of TP3M, LE-NMR, RAPM, CNMR and [5]-NMR schemes where the energy consumption has been normalized with respect to the energy consumption of TP3M. These observations can be made from Fig. 7:

- It can be seen from Fig. 7 that when the parallelism degree of task graphs increases, the energy consumption decreases. This is because the amount of dynamic slack times increases, and hence we can achieve significant energy savings. However, the energy consumption of TP3M is always less than the other three schemes (RAPM, CNMR, and [5]-NMR).

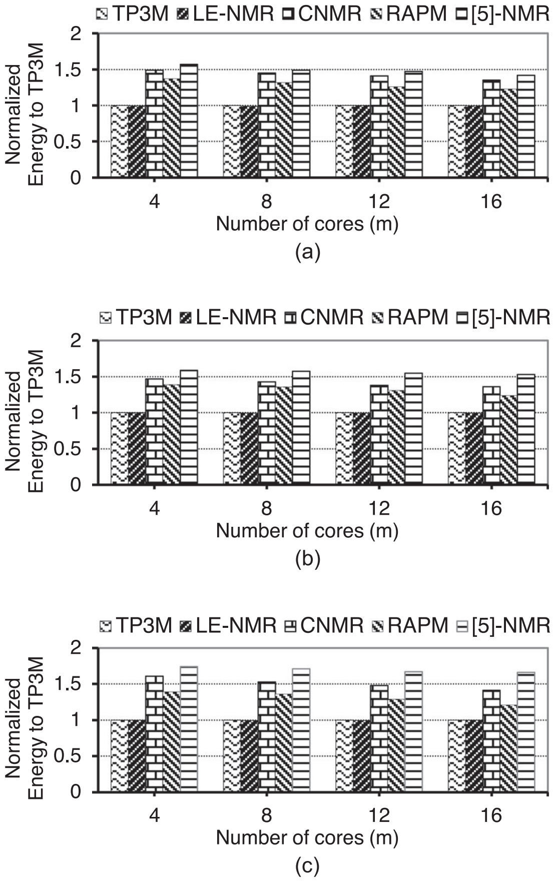


(a)

(b)

(c)

Fig. 6. Power consumption profile in the actual-case scenario on a 4-core system, a) Low parallelism degree, b) Medium parallelism degree, c) High parallelism degree.



(a)

(b)

(c)

Fig. 7. Normalized energy consumption to TP3M in the worst-case scenario. a) Low parallelism degree, b) Medium parallelism degree, c) High parallelism degree.

- When the task graph parallelism degree increases from low (Fig. 7a) to high (Fig. 7c), the energy consumption of TP3M is always less than or equal to the other schemes. TP3M provides in average respectively 30.7 percent (up to 37.8), 23.4 percent (up to 28 percent) and 36.5 percent (up to 42.5 percent) energy saving as compared to CNMR, RAPM and [5]-NMR.
- While TP3M provides almost the same energy consumption as LE-NMR (Fig. 6), but TP3M consumes much less energy than CNMR, RAPM and [5]-NMR (Fig. 7) and consumes much less peak power than four schemes mainly because of the more sophisticated power-management technique that TP3M uses.

We also compared TP3M with $N = 3$ and $N = 7$ with RAPM, CNMR, and [5]-NMR. The experiments show that TP3M completely outperforms the three schemes from both the energy and peak power consumption viewpoints. The TP3M scheme with $N = 3$ and $N = 7$ provides on average respectively 22.6 percent (up to 32.7 percent) and 26.9 percent (up to 44.3 percent) energy saving as compared to three mentioned schemes.

6 Conclusion

In this paper, we have presented a solution to reduce peak power consumption on multicore embedded systems, which uses fault-tolerance techniques to achieve high reliability. We have developed a new scheduling algorithm (TP3M) that avoids concurrent execution of tasks based on one policy called the peak-power-aware longest task first. The proposed method tries to remove overlaps of the peak power of concurrently executing tasks to keep the maximum power consumption below the chip TDP constraint. Meanwhile, the proposed scheme considers that the power consumption of the core must be less than the core TDP constraint. Also, we use the DVS technique to reduce the instantaneous power dissipation on each core. At runtime, we exploit a scheme that provides further power reduction in the realistic scenario. It cancels the execution of the $\lfloor N/2 \rfloor$ copies of those tasks that during the execution of their $\lceil N/2 \rceil$ copies no fault has occurred. The experimental results show that TP3M provides up to 50 percent peak power reduction and 44.3 percent energy saving as compared to state-of-the-art schemes.

Acknowledgments

Mohsen Ansari, Sepideh Safari, Amir Yeganeh-Khaksar and Alireza Ejlali acknowledge Research Vice-Presidency of Sharif University of Technology for funding this work under grant no. G930827.

References

[1] J. Lee, B. Yun and K. G. Shin, "Reducing peak power consumption in multi-core systems without violating real-time constraints," *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, no. 4, pp. 1024–1033, Apr. 2014.

[2] M. Salehi, A. Ejlali, and B. M. Al-Hashimi, "Two-phase low-energy N-modular redundancy for hard real-time multi-core systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, no. 4, pp. 1024–1033, Apr. 2015.

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

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

[5] W. Munawar, H. Khdr, S. Pagani, M. Shafique, J.-J. Chen, and J. Henkel, "Peak power management for scheduling real-time tasks on heterogeneous many-core systems," in *Proc. 20th IEEE Int. Conf. Parallel Distrib. Syst.*, Dec. 2014, pp. 200–209.

[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 *Proc. IEEE/ACM Int. Conf. Hardware/Softw. Codesign Syst. Synthesis*, Oct. 2014, pp. 1–10.

[7] M. I. Bandan, S. Pagliarini, J. Mathew and D. Pradhan, "Improved multiple faults-aware placement strategy: Reducing the overheads and error rates in digital circuits," *IEEE Trans. Reliab.*, vol. 66, no. 1, pp. 233–244, Mar. 2017.

[8] F. R. Poursafaei, S. Safari, M. Ansari, M. Salehi, and A. Ejlali, "Offline replication and online energy management for hard real-time multicore systems," in *Proc. 1th Int. CSI Symp. Real-Time Embedded Syst. Technol.*, Oct. 2015, pp. 1–7.

[9] Q. Han, M. Fan, and G. Quan, "Energy minimization for fault tolerant real-time applications on multiprocessor platforms using checkpointing," in *Proc. IEEE/ACM Int. Symp. Low Power Elect. Design*, Sep., 4-6 2013, pp. 76–81.

[10] A. Mirhoseini, E.M. Songhori, and F. Koushanfar, "Automated checkpointing for enabling intensive applications on energy harvesting devices," in *Proc. IEEE/ACM Int. Symp. Low Power Electron. Des.*, pp. 27–32, Sep., 4-6 2013.

[11] A. Ejlali, B. M. Al-Hashimi, M. T. Schmitz, P. Rosinger, S. G. Miremadi, "Combined time and information redundancy for SEU-tolerance in energy-efficient real-time systems," *IEEE Trans. Very Large Scale Integr Syst.*, vol. 14, no. 4, pp. 323–335, Apr. 2006.

[12] P. Shivakumar, M. Kistler, S. Keckler, D. Burger, and L. Alvisi, "Modeling the effect of technology trends on the soft error rate of combinational logic," in *Proc. Int. Conf. Dependable Syst. Netw.*, 2002, pp. 389–398.

[13] S. Reinhardt and S. Mukherjee, “Transient fault detection via simultaneous multithreading,” *ACM SIGARCH Comput. Archit. News*, vol. 28, no. 2, pp. 25–36, 2000.

[14] S. S. Mukherjee, M. Kontz and S. K. Reinhardt, “Detailed design and evaluation of redundant multi-threading alternatives,” in *Proc. 29th Annu. Int. Symp. Comput. Archit.*, 2002, pp. 99–110.

[15] A. Shye, T. Moseley, V. J. Reddi, J. Blomstedt, and D. A. Connors, “Using process-level redundancy to exploit multiple cores for transient fault tolerance,” in *Proc. 37th Annu. IEEE/IFIP Int. Conf. Dependable Syst. Netw.*, 2007, pp. 297–306.

[16] D. K. Pradhan, *Fault-Tolerant Computer System Design*. Upper Saddle River, NJ, USA, Prentice-Hall, 1996.

[17] I. Koren, and C. M. Krishna, *Fault-Tolerant Systems*. San Francisco, CA, USA: Morgan Kaufmann, Elsevier, 2007.

[18] A. Munir, S. Ranka, and A. Gordon-Ross, “High-performance energy-efficient multicore embedded computing,” *IEEE Trans. Parall. Distr. Syst.*, vol. 23, no. 4, pp. 684–700, Apr. 2012.

[19] Intel Corporation, “Dual-core intel xeon processor 5100 series datasheet, revision 003,” Aug. 2007, https://www.intel.com/Assets/PDF/datasheet/313355.pdf

[20] Intel Corporation, “Desktop 3rd generation intel core processor family thermal mechanical specifications and design guidelines,” Jan. 2013, https://www.intel.com/content/dam/www/.../3rd-gen-core-desktop-vol-1-datasheet.pdf

[21] A. Ejlali, B. M. Al-Hashimi, and P. Eles, “A standby-sparing technique with low energy-overhead for fault-tolerant hard real-time systems,” in *Proc. Int. Conf. Hardware-Softw. Codesign Syst. Synthesis*, Grenoble, Oct. 2009, pp. 193–202.

[22] M. A. Haque, H. Aydin, and D. Zhu, “Energy-aware standby-sparing technique for periodic real-time applications,” in *Proc. IEEE 29th Int. Conf. Comput. Des.*, Oct. 2011, pp. 190–197.

[23] M. A. Haque, H. Aydin, and D. Zhu, “Energy management of standby-sparing systems for fixed-priority real-time workloads,” in *Proc. Green Comput. Conf.*, Jun. 2013, pp. 1–10.

[24] P. Pillai, and K. G. Shin, “Real-time dynamic voltage scaling for low-power embedded operating systems,” in *Proc. SOSP ACM Symp. Operating Syst. Principles*, Dec. 2001, pp. 89–102.

[25] D. Zhu, R. Melhem, D. Mosse, and E. Elnozahy, “Analysis of an energy efficient optimistic TMR scheme,” in *Proc. Tenth Int. Conf. Parall. Distr. Syst.*, 2004, pp. 559–568.

[26] N. Fisher, J. Chen, S. Wang, and L. Thiele, “Thermal-aware global real-time scheduling and analysis on multicore systems,” *J. Syst. Archit.*, vol. 57, no. 5, pp. 547–560, 2011.

[27] R. Jejurikar, C. Pereira, and R. Gupta, “Leakage aware dynamic voltage scaling for real-time embedded systems,” in *Proc. 41st Des. Autom. Conf.*, 2004, pp. 275–280.

[28] B. Lee, J. Kim, Y. Jeung, and J. Chong, “Peak power reduction methodology for multi-core systems,” in *Proc. Int. SoC Des. Conf.*, 2010, pp. 233–235, Nov. 22-23, 2010.

[29] S. Pagani, J. J. Chen, and J. Henkel, “Energy and peak power efficiency analysis for the single voltage approximation (SVA) scheme,” *IEEE Trans. Comput.-Aided Des. Integrated Circuits Syst.*, vol. 34, no. 9, pp. 1415–1428, Sep. 2015.

[30] Y. Guo, D. Zhu, and H. Aydin, “Reliability-aware power management for parallel real-time applications with precedence constraints,” in *Proc. Int. Green Comput. Conf. Workshops*, Jul. 2011, pp. 1–8.

[31] A. Ejlali, B. M. Al-Hashimi, and P. Eles, “Low-energy standby-sparing for hard real-time systems,” *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 31, no. 3, pp. 329–342, Mar. 2012.

[32] M. K. Tavana, M. Salehi, and A. Ejlali, “Feedback-based energy management in a standby-sparing scheme for hard real-time systems,” in *Proc. IEEE 32nd Real-Time Syst. Symp.*, Nov./Dec. 2011, pp. 349–356.

[33] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown, “MiBench: A free, commercially representative embedded benchmark suite,” in *Proc. 4th IEEE Ann. Workshop Workload Characterization*, 2001, pp. 3–14.

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

[35] S. Li, J.-H. Ahn, R. Strong, J. Brockman, D. Tullsen, and N. Jouppi, “McPAT: An integrated power, area, and timing modeling framework for multicore and manycore architectures,” in *Proc. 42nd Annu. IEEE/ACM Int. Symp. Microarchitecture*, 2009, pp. 469–480.

[36] D. Rhodes and R. Dick, “TGFF: Task graphs for free,” *Proc. 6th Int. Workshop Hardware/Softw. Codesign*, Mar. 1998, pp. 97–101.

[37] T. Tobita and H. Kasahara, “A standard task graph set for fair evaluation of multiprocessor scheduling algorithms,” *J. Scheduling*, vol. 5, no. 5, pp. 379–394, Sep. 2002.

[38] 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

Mohsen Ansari received the MSc degree in computer engineering from the 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 and peak power management in embedded systems, and multi-/many-core systems with a focus on dependability/reliability.

Sepideh Safari received the MSc degree in computer engineering from the Sharif University of Technology, Tehran, Iran, in 2016. She is currently working toward the PhD degree in computer engineering at the 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.

Amir Yeganeh-Khaksar received the BSc degree in computer engineering from the Ferdowsi University of Mashhad. He is currently working toward the MSc degree in the Department of Computer Engineering at the Sharif University of Technology, Tehran, Iran. His research interest lies in computer architecture, especially in low power design and embedded systems.

Mohammad Salehi received the PhD degree in computer engineering from the Sharif University of Technology, Tehran, Iran, in 2016. He is currently an assistant professor of computer engineering at University of Guilan, Rasht, Iran. From 2014 to 2015, he was a visiting researcher in the Chair for Embedded Systems (CES), the Karlsruhe Institute of Technology (KIT), Germany. His research interests include design of low-power, reliable and real-time embedded systems with a focus on dependability and energy efficiency in cyber-physical systems and internet of things (IoT).

Alireza Ejlali received the PhD degree in computer engineering from the Sharif University of Technology in, Tehran, Iran, in 2006. He is currently an associate professor of computer engineering at the Sharif University of Technology. From 2005 to 2006, he was a visiting researcher in the Electronic Systems Design Group, University of Southampton, Southampton, United Kingdom. In 2006, he joined the 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.