Energy- and Reliability-Aware Task Replication in Safety-Critical Embedded Systems

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

***Abstract*—Safety-critical systems should satisfy a required level of reliability. To meet a desired reliability target, task replication can be realized with exploiting multicore platforms. However, inattentive task replication might lead to significant power, energy, and time overhead. In this paper, we demonstrate that when we use task replication technique, the required number of replicas for each task and the energy consumption of the system are significantly dependent on the accuracy of the fault detection. At design time, we propose a method that determines the level of replication along with the voltage and frequency setting for each task to satisfy a desired reliability target such that the energy consumption is minimized. At run time, the proposed method controls cancelling the task replicas in the fault-free scenarios. The proposed method can be applied on both dynamic- and static-priority applications. We evaluated the effectiveness of our method through extensive simulations. The evaluation results show that our proposed method provides up to 43.5% (on average 26.2%) energy saving without reliability degradation.**

***Index Terms*—Reliability, Task Replication, Energy, Safety-Critical Systems, Multicore Systems.**

I. Introduction

ENERGY consumption is one of the important factors of designing safety-critical embedded systems [1][2][3]. More consumed energy in these systems leads to reduced battery life time and higher temperature [3]. There are two well-studied energy management techniques including Dynamic Voltage and Frequency Scaling (DVFS) and Dynamic Power Management (DPM) [1][3][4]. DVFS prolongs the execution time of tasks and degrades the reliability of the tasks, hence, employing the DVFS technique in safety-critical embedded systems might lead to violating timing constraints and reliability targets [5].

Embedded systems are commonly exploited for safety-critical applications whenever high reliability is a very important objective [10][14]. Reducing energy consumption and improving reliability in these systems are in conflict with each other. Because high reliability mainly requires employing fault-tolerant techniques [1][2][3][13]. When DVFS is used for power/energy reduction, its negative impact on system-level reliability should be considered [11], because the fault rate increases exponentially whenever voltage and frequency are reduced for energy saving. Therefore, reliability degradation due to applying DVFS should be considered carefully in safety-critical applications.

The problem of high reliability demand of safety-critical systems is further exacerbated by the fact that the emerging multicore processors used for executing these applications, are fabricated with unreliable technologies [1][15]. These processors built from unreliable components are more susceptible to different kinds of faults. Therefore, another principal issue for coping with the different reliability threats is the adoption of appropriate fault detection (and even recovery) mechanisms [2]. The various fault detection mechanisms provide different fault coverage, i.e., the probability of successful fault detection which leads to different overheads in terms of extra hardware and time [6]. The accuracy of the fault detection mechanism makes an impact on the required number of replicas for a task to satisfy its reliability target and minimization of the energy consumption. Although some fault detection mechanisms significantly reduce the probability of undetected faults, no fault detection solution is perfect [1][2]. To the best of our knowledge, in the related studies, the effects of the fault detection mechanism on reliability and the total energy consumption of the system has been ignored.

In this paper, we focus on an energy- and reliability-aware task replication technique for multicore systems when running a variety of applications including dynamic- or static-priority task sets. Increasing the number of cores on the recently emerging multicore systems makes great opportunities to employ task replication techniques for reliability improvement. We propose a design-time-based replication technique and run-time-based energy management method for safety-critical multicore embedded systems. In the design time, the required number of replicas and the voltage and frequency level for each task by considering a desired reliability target and the fault coverage of the fault detection mechanism are determined. At run time, the run-time manager prevents the unnecessary execution of tasks' copies because we need at least one of its copies which finishes successfully. This run-time manager leads to save up to 23% more energy compared to the cases where just the offline phase has been applied.

The main contributions of this paper are:

- Providing an energy-efficient mapping and scheduling of the system along with finding the required number of replicas and the voltage and frequency level for all tasks to meet their reliability targets.
- Managing the cancelation of the tasks' replicas to prevent unnecessary executions.
- Executing a variety of applications consisting of dynamic- and static-priority task sets.
- Examining the impact of different fault detection mechanisms, with different fault coverage, on reliability and energy consumption of the tasks.

II. Models and Assumptions

In this section, we introduce the system, task, power consumption, and reliability models.

A. Task Model

We consider a set of N periodic hard real-time tasks $T=\{\tau_1, \tau_2, \ldots, \tau_N\}$. Each task set T has a priority type $Prio_T$ that determines whether the tasks in T are dynamic- or static-priority. Each task τ_i has a worst-case execution time WC_i under the maximum frequency level f_{max} and a period Π_i. The tasks are scheduled and executed on M homogeneous cores. Each core has F sets of voltage and frequency levels. We employ the normalized frequency f regarding the maximum frequency level f_{max} ($0.4 \leq f \leq f_{max}=1.0$). Note that the minimum normalized frequency level is equal to 0.4. Because, in reality, the voltage and frequency level of each task can be scaled to some extent and no further scaling of voltage and frequency level can be applied [1][4]. Moreover, each task τ_i has k_i replicas which are scheduled and executed on $k_i \leq M$ different cores.

B. Energy Consumption Model

We assume that the total energy consumption of the proposed system E_{total} has static and dynamic parts. The total energy consumption is calculated as [8]:

$$E_{total} = E_{static} + E_{dynamic} \quad (1)$$

where the static energy consumption is denoted by E_{static} which is overcome by the leakage current of the system and the dynamic energy consumption is denoted by $E_{dynamic}$.

Note that since the processing cores exploit the DVFS technique, the supply voltage V_i may be scaled and less than the maximum supply voltage V_{max} during the execution of each task. We consider the normalized supply voltage ρ_i as:

$$\rho_i = \frac{V_i}{V_{max}} \quad (2)$$

When the task τ_i ($E_{dynamic}(\tau_i)$) is executed at the scaled supply voltage V_i, the dynamic energy consumption of each core can be written as:

$$E_{dynamic}(\tau_i) = C_{eff} V_i^2 f_i \left(\frac{WC_i}{\rho_i}\right) \quad (3)$$

where C_{eff} is the effective switching capacitance, and V_i and f_i are the supply voltage and operational frequency, respectively. WC_i/ρ_i is the scaled worst-case execution time of τ_i due to applying DVFS. Note that since voltage has a linear correlation with frequency, we can have $\rho_i = V_i/V_{max} = f_i/f_{max}$ where V_{max} is the maximum supply voltage corresponding to the maximum frequency f_{max}. Hence, Eq. 3 can be rewritten as:

$$E_{dynamic}(\tau_i) = C_{eff} V_{max}^2 f_{max} \rho_i^2 WC_i \quad (4)$$

We exploit the normalized energy consumption by removing $C_{eff} V_{max}^2 f_{max}$. Therefore, the normalized dynamic energy consumption of each core $NE_{dynamic}(\tau_i)$ while executing the task τ_i can be written as:

$$NE_{dynamic}(\tau_i) = \rho_i^2 WC_i \quad (5)$$

To figure out the effect of the static energy consumption on the total energy of the system, we should first calculate the static power when the task τ_i is executing:

$$P_{static}(\tau_i) = I_{sub} V_i \quad (6)$$

where $P_{static}(\tau_i)$ is the static power consumption corresponding to the task τ_i, and I_{sub} is the sub-threshold leakage current. By means of Eq. 6, the static energy during the execution of the task τ_i (i.e., $E_{static}(\tau_i)$) is as follows:

$$E_{static}(\tau_i) = P_s(\tau_i)\left(\frac{WC_i}{\rho_i}\right) \quad (7)$$

C. Impact of the Fault Detection Mechanism

As mentioned before, the fault detection mechanism has overhead in terms of hardware and time which affects the energy consumption of the system. Therefore, if the total power consumption of the system is denoted by P_{total}, and the total time that the system spends to finish its duty is denoted by t_{total}, the total energy of the system, E_{total}, consisting of the useful energy (which is consumed for executing the system's duty) and the additional energy (due to the overheads of the fault detection mechanism) can be calculated as follows:

$$E_{total} = (P_{total} + \alpha P_{total})(t_{total} + \beta t_{total}) \quad (8)$$

$$E_{total} = P_{total} t_{total}(1 + \alpha + \beta + \alpha\beta) \quad (9)$$

where α is a coefficient specifying the hardware overhead of the fault detection mechanism, and β is a coefficient describing the performance overhead. With some simplifications, Eq. 9 can be derived from Eq. 8. It can be inferred from Eq. 9 that the energy overhead of the fault detection mechanism is equal to $(\alpha+\beta+\alpha\beta) \times E_{total}$ which, depending on the value of α and β, may have considerable impact on the total energy of the system.

According to what has been mentioned earlier, the total energy consumption of the system equipped with a fault detection mechanism includes three main parts: the useful energy that is consumed to execute a task, the extra energy that is imposed to the system due to the hardware overhead, and the excess energy due to the overhead of the fault detection mechanism

D. Fault Model and Reliability Analysis

The transient fault rate is modeled by an exponential distribution [3]. According to Eq. 10, the rate of transient faults increases when the frequency level decreases through DVFS [11]. If the fault rate at the maximum frequency is λ_0, the fault rate at the frequency f is [7][11][12][28]:

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

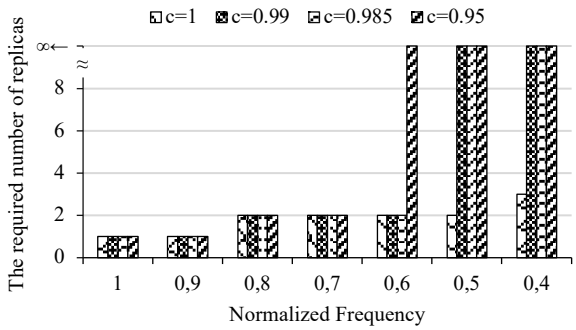
where d is the sensitivity factor to the technology and its typical value ranges from 2 to 6 [16]. We assume that at the maximum available frequency, the value of λ_0 is equal to 10^{-6} [1][2][3],.

The probability of successful execution of a task is called "reliability". The reliability of a task at frequency f_i is computed as [9][19][20][22][23][28]:

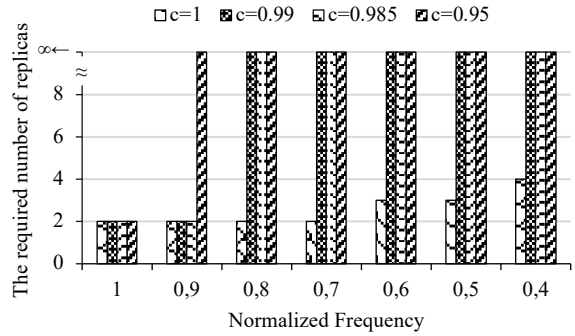
$$R_i(f_i) = e^{-\lambda(f_i)\frac{WC_i}{f_i}} \quad (11)$$

When we have several copies of a task on the system, it is enough to have at least one copy successfully executed. At the finish time of each task, the fault detection mechanism tries to detect a fault [3]. The fault coverage of the fault detection mechanism strongly affects the total system reliability which is the product of all tasks reliability executing on the system.

If we denote the reliability of each single copy of the task τ_i, at the frequency f by $R_i(f)$, the reliability of the same task with k different copies by $R_i^k(f)$, and the fault coverage of the fault detection mechanism by c, the reliability of the τ_i with k replicas is:



(a) R_{targ}=0.99999



(b) R_{targ}=0.9999999

Fig. 1. The required number of replicas for a task to satisfy its target reliability as a function of frequency and fault coverage, when (a) R_{targ}=0.99999, (b) R_{targ}=0.9999999.

$$R_i^k(f) = \sum_{l=1}^{k} R_i(f)(1 - R_i(f))^{l-1} c^{1-l} \quad (12)$$

As mentioned before, no fault detection method is perfect. In other words, the fault coverage is never exactly equal to 1. Originally, the fault coverage of a fault detection mechanism can be defined as the probability of detecting that a copy of a task has been finished unsuccessfully. Although the previous studies assumed the fault coverage is perfect, it is obvious from Eq. 12 that to meet a target reliability R_{targ}, both $R_i(f)$ and c plays almost similar role. Because there are some situations when imperfect fault coverage may impose a greater number of replicas for the task.

Therefore, given a certain reliability target (R_{targ}), for a task to meet its reliability target, the following inequality should be true:

$$R_{targ} \leq R_i^k(f) \quad (13)$$

If we constitute the $R_i^k(f)$ from Eq. 12, we have:

$$R_{targ} \leq \sum_{l=1}^{k} R_i(f)(1 - R_i(f))^{l-1} c^{1-l} \quad (14)$$

From the Eq. 14, we determine the minimum number of replicas in order to meet its reliability target. It confirms the previous intuitive conclusion that the required number of replicas for a task to satisfy its reliability target not only depends on the reliability target and the processing frequency, but also relates to the fault coverage of the fault detection mechanism.

Fig. 1 helps to gain a better insight into Eq. 14 that clarifies the impact of the fault coverage of the fault detection mechanism and the processing frequency on the required number of replicas for a task to meet its reliability target. Generally, as the frequency scales down which leads to the decline of the reliability, the task's required number of replicas to meet its reliability target increases. The key point about these two figures is that the fault coverage of the fault detection mechanism affects the required number of replicas significantly. For instance, when the fault coverage is perfect (i.e., c=1), the reliability target is met with minimum number of replicas in comparison to other cases where the fault coverage is imperfect (i.e., c<1). Another point that needs to be mentioned is that for cases with imperfect fault coverage, meeting the reliability target at all frequency levels may not be possible. In other words, when the fault coverage is imperfect, for some frequency levels, increasing the number of replicas may not results in meeting the reliability target. Therefore, the important point that can be deduced from Eq. 14, together with the Fig. 1, is that all the three factors including the frequency level, the number of replicas, and the fault coverage of the fault detection mechanism affect the ability of a system to make the reliability target of a task achievable.

It is concluded that for satisfying a desired reliability target, there are several sets of the number of replicas and voltage and frequency levels that can satisfy the reliability target. Both reliability and energy consumption of the system are under the effects of the fault detection mechanism. Although we mainly considered transient faults, our proposed method can be capable of handling the permanent faults in an efficient way as well. The way of achieving this goal is described as follows. It is remarkable that according to Eq. 14, for τ_i to be able to meet its reliability target, it may need k replicas that at least one of them should complete successfully. If we map more than one copy of τ_i to a single core, and the assigned core encounters a permanent fault, more than one copy of τ_i becomes faulty and τ_i may not be able to achieve its reliability target. Therefore, for τ_i to be able to accomplish its reliability target, even in the presence of probable permanent faults, a fundamental requirement is to map different replicas of a task to distinct cores. Moreover, apart from k copies that satisfy the reliability target in the presence of transient faults that has been computed without the consideration of the permanent faults, τ_i should have δ more copies, for the sake of encountering δ permanent faults. Therefore, τ_i should have $k+\delta$ copies to satisfy its reliability target in presence of transient as well as δ permanent faults, in total.

III. PROBLEM DEFINITION AND SOLUTION

Assume that we have a multicore system consisting of M homogeneous cores plus a predefined fault detection method, and a set of N periodic hard real-time tasks. The main problem is to determine the number of replicas to meet the reliability target and the voltage and frequency level for each task such that the total energy consumption of the system is reduced, considering the fault coverage of the fault detection mechanism. Mapping the tasks to the cores should guarantee that all deadlines are met. In addition, all copies of a task are mapped to different cores because we wish to tolerate any permanent

fault in one core from corrupting more than one copy of a task. The minor part of the problem is about to use the opportunities created in fault-free scenarios at run time to further reduce energy. When a copy of a task finishes successfully, the online manager stops the execution of the remaining parts of other copies and puts the system to a low energy state. An overview of our proposed method is shown in Fig. 2. The following subsections present the details of each phase.

A. *Offline Phase*

Finding an optimal solution for the energy-efficient replication problem is an NP-hard problem [1][17][21][22]. Therefore, we develop a heuristic-based algorithm. However, the original EER algorithm does not model our problem completely and efficiently. Therefore, we modify this algorithm in such a way that the obtainable solution has the necessary efficacy. In this phase, we walk through a set of steps that are discussed in the following paragraphs.

Initially, since each task set can have either dynamic- or static-priority, Algorithm 1 checks the priority type of the task set and marks the task set with the proper tag specifying the priority type. Then, the algorithm constructs a collection of configurations for each task. Generally, each of these configurations consists of some practical information of which the major ones include the *Reliability-Energy-Frequency (REF)* of the task corresponding to that configuration. Indeed, each configuration consists of a frequency level f from the set of all possible frequencies in the system F, the minimum number of copies k, the corresponding overall energy consumption of the configuration. Note that k is under the influence of the fault coverage of the selected fault detection mechanism as well as the reliability of the task at the frequency f, and is computed according to Eq. 14. We call this collection containing all the possible task's configurations as *REF collection.*

After constructing the REF collection for each task, the algorithm maps the tasks to the set of available M cores, with regard to the task set priority type. For mapping the tasks to the cores, a modified version of *First-Fit-Decreasing (FFD)* is developed. The notable modifications are: (1) the total utilization of the assigned tasks to each core should not be more than the utilization bound imposed by the scheduling policy, (2) for tolerating the permanent faults, the task to core allocation policy should map copies of the same task on different cores.

In the phase of mapping the tasks to the cores, the algorithm starts with the most energy-efficient configuration of the REF collection for each task at its minimum frequency level. If this process is successful, the optimal solution can be found and the algorithm can be terminated and return the whole system configuration. The whole system configuration consists of the selected features for each task. To distinguish the task's and the whole system's configurations, from now on, we call the former the *task configuration*, and the latter the *entire system configuration*. If the first process is not successful, the algorithm proceeds to the next step. In this step, firstly, the algorithm knows whether there is a feasible schedule or not. The algorithm checks another situation where each task is running at the maximum possible frequency. If the algorithm cannot find a solution, the algorithm returns an infeasible message. In this situation the execution of the workload on the system is not feasible at all. Then, there cannot exist any offline

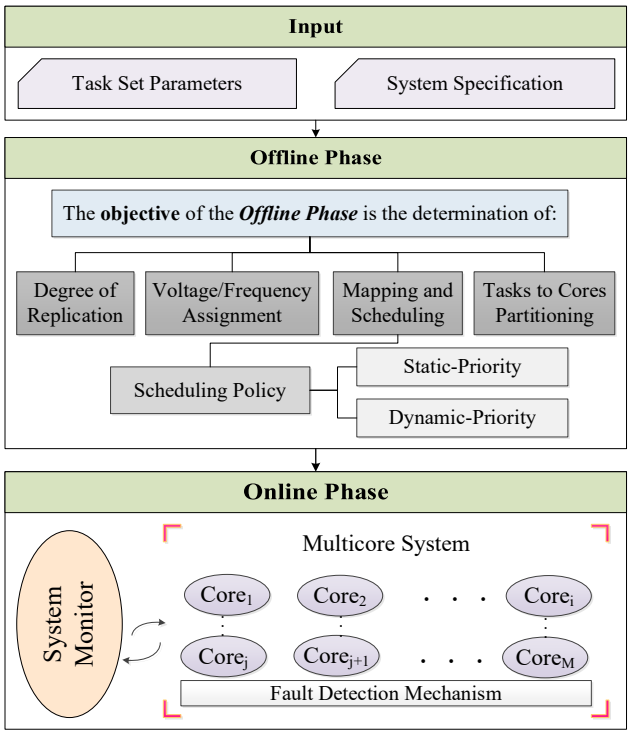


Fig. 2. The overview of our proposed mapping and scheduling method.

execution and the corresponding online phase will not even start. Otherwise, the algorithm considers the relaxation mode.

In the relaxation mode, the algorithm starts with a feasible entire system configuration. It should be noted that at the first run of the relaxation mode, each task has its configuration where each of its copy is executing at the maximum frequency. In each iteration of the relaxation mode, the algorithm selects the most eligible task to relax its configuration. By relaxing the configuration, we mean to promote the task's configuration to a new one which is a more efficient configuration. This promotion is mainly accompanied by the reduction of the frequency (and so the corresponding voltage) of the task, that means relaxing the processing frequency level. The selection of the most eligible task is according to the proposed policies in [18] (will be explained later). The algorithm considers another round of relaxation, and the new entire system configuration is committed such that the new entire system configuration is feasible. Otherwise, the selected task will be omitted from the set of eligible tasks for relaxation and will not be marked for the further iterations, and then, the system backtracks to its previous feasible entire system configuration. Moreover, when a task reaches a configuration in which the frequency is the minimum possible frequency, it will be deleted from the set of eligible tasks for the next iterations. This process continues until there is no task in the set of eligible tasks. Finally, the last feasible entire system configuration of the system is chosen, and will be passed to run time.

In the offline phase of our proposed method, some policies which are introduced in [18] are considered to choose the most eligible task in each iteration of the offline phase. These heuristics are:

- *Maximum-Energy-Saving (MES):* The task that results in the maximum energy saving by relaxing its configuration is chosen [18].

Algorithm 1: *Offline Phase Algorithm*

```
8.  Start the offline phase
9.  /* Design time algorithm: applying a modified version of the EER */
10.   Check the priorityType of the task set
11.       taskSetPriority ← priorityType
12.   Construct the REF collection for each task of the task set
13.   for each task τi in the task set do
        /*the most energy-efficient of the task τi is meeConfigi*/
14.     TasksConfig[τi] ← meeConfigi
15.   end for
16.   EntireSysConfig ← Partition the TasksConfig with modified_FFD
17.   if (taskSetPriority == dynamicPriority) then
18.     feasible(EntireSysConfig)←EDF_schedulable(EntireSysConfig)
19.   else if (taskSetPriority == StaticPriority) then
20.     feasible(EntireSysConfig)←RMS_schedulable(EntireSysConfig)
21.   else
22.       return error and exit
23.   end if
24.   if (feasible(EntireSysConfig)) then
25.     return the TasksConfig and the EntireSysConfig
26.     exit
27.   end if
      /*the relaxation mode starts*/
28.   for each task τi in the task set do
      /*the least energy-efficient of the task τi leeConfigi*/
29.     TasksConfig[τi] ← leeConfigi
30.     eligible[τi] ← true
31.   end for
32.   EntireSysConfig ← Partition the TasksConfig with modified_FFD
33.   if (! feasible(EntireSysConfig)) then
34.     return error /* No feasible solution exists */
35.     exit
36.   end if
37.   while (∃j (eligible[τj] is true)) do
38.     Select the most eligible task according to MES, MPS, or MU
        heuristic algorithms
        /*τi is chosen for relaxation mode*/
        /*nxConfigi: the next energy-efficient configuration of τi*/
39.     TasksConfig[τi] ← nxConfigi
40.     EntireSysConfig ← Partition the TasksConfig with modified FFD
41.   if (taskSetPriority == dynamicPriority) then
42.     feasible(EntireSysConfig)←EDF_schedulable(EntireSysConfig)
43.   else if (taskSetPriority == StaticPriority) then
44.     feasible(EntireSysConfig)←RMS_schedulable(EntireSysConfig)
45.   else
46.       return error and exit
47.   end if
48.     if(!feasible(EntireSysConfig)) then
        /*goes back to the last feasible-configuration*/
          TasksConfigl[τi] ← preConfigi
49.       eligible[τi] ← false;
50.     end if
51.   end while
52.   return the TasksConfig and the EntireSysConfig
53. End
```

- *Maximum-Power-Saving (MPS):* The task that has the largest energy savings by considering the additional processing time is chosen as the most eligible task [18].
- *Maximum-Utilization (MU):* In this heuristic, the most eligible task with the largest utilization value is chosen [18].

The pseudo-code of the procedures of the offline phase is presented in Algorithm 1. The offline phase starts at line 1. At line 3, the algorithm checks the priority type of the task set, and at line 4, it marks corresponding field of the task set with the appropriate tag demonstrating the task set's priority type. Then at line 5, the REF collection for each task is constructed. Line 6 to 8 are the first trial of the offline phase algorithm where the most energy-efficient configuration of each task is chosen. At line 9, the algorithm tries to partition the tasks with their selected configurations among the cores according to FFD. In lines 10-20, the algorithm checks whether the suggested mapping of tasks to cores are feasible according to the scheduling policy. The scheduling policy is specified according to the priority of the task set (EDF for dynamic-priority and RMS for static-priority task sets). If this mapping is feasible, the algorithm hands in the tasks configurations and the entire system configuration as output to be used in the online phase. Otherwise, the relaxation mode starts. Lines 30 to 45 show the relaxation mode of the algorithm. At line 31, the most eligible task is selected. At line 32, the algorithm promotes the selected task's configuration and reaches a new task set's configuration. Again, in lines 33 to 40 the algorithm maps the tasks to each core and checks the feasibility of the suggested task set. If the configuration is feasible, the relaxation mode continues until the eligible task set becomes empty. Otherwise, the algorithm backtracks to its last feasible task set configuration (lines 41-43). At line 45, the best choice of the tasks' configuration and the entire system configuration are passed as the offline phase output. The algorithm ends at line 46.

Algorithm 2: Online Phase Algorithm

```
    Inputs: TasksConfig and EntireSysConfig from the design time algorithm
1.  Start the online phase
      /*Run-time algorithm:  Applying DPM*/
2.    for each task τi in the TasksConfig do
3.      coreSet[τi] ← all cores that execute a copy of τi
4.      online_Manager controls running tasks on the cores
5.          detect the first copy of τi that is finished successfully on one of
            the cores in coreSeti and drop all the remaining parts of other
            copies of τi on other cores in coreSeti
6.    end for
7.  End
```

B. Online Phase

At run time, the objective of our method is to further reduce the energy consumption by considering the entire system configuration from the offline phase. The online controller cancels the unnecessary copies of the tasks to achieve further energy savings. Then, the online manager puts the system into a low-energy state.

Algorithm 2 shows the pseudo-code of the online phase. The algorithm starts at line 1. Line 2 to 7 consist of the main responsibility of the online phase. At line 3, the set of all cores where a copy of a task is executing is found. Then, at line 4, the online manager starts to control the system as the tasks are running. At line 5, the online manager detects the first copy which finishes, and at line 6, the manager drops the remaining parts of other copies on the other cores. The algorithm finishes at line 8.

IV. Results and Discussion

In this section, our simulation results are discussed. We consider an ARM processor, which is exploited in safety-critical embedded systems, based on simulations conducted on gem5 [25] and McPAT [26] with out-of-order Cortex-A7 cores. Our simulations execute various task sets, including real-life real-time embedded applications of MiBench Benchmark [27]

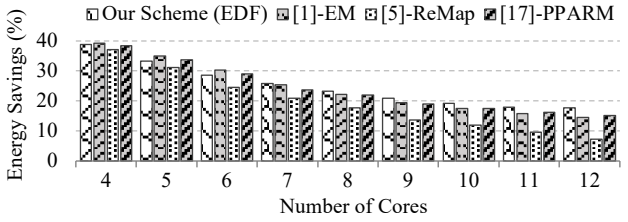


Fig. 3. Impact of increasing the number of cores for dynamic priority task set with U_{tot}=3 and R_{targ}=0.99999.

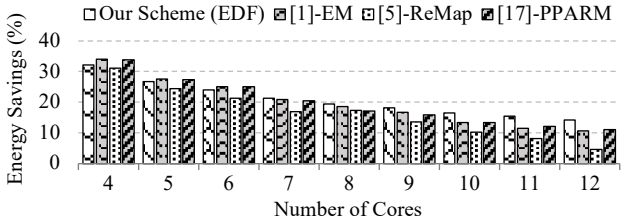


Fig. 4. Impact of increasing the number of cores for dynamic priority task set with U_{tot}=3 and R_{targ}=0.9999999.

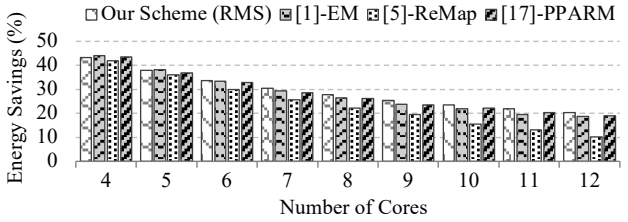


Fig. 5. Impact of increasing the number of cores for static priority task set with U_{tot}=3 and R_{targ}=0.99999.

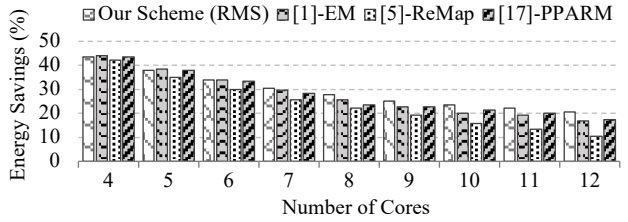


Fig. 6. Impact of increasing the number of cores for static priority task set with U_{tot}=3 and R_{targ}=0.9999999.

running on a homogeneous multicore platform with 16 cores. Then, we develop a system-level discrete event scheduler.

Our evaluations consist of three major sets of simulation. In the first set of evaluation, we analyze the impact of increasing the number of available cores in the system on the energy savings. In the second set, the effects of the system load on the energy savings are examined. Finally, in the third set of simulation, the impact of the fault coverage of different fault detection methods on the energy consumption of the system is evaluated.

The first and second sets of simulations include different cases. In these cases, we simulate different scenarios for dynamic- and static-priority task sets separately. The set of all available frequency contains seven different normalized frequency levels regarding f_{max}= 2GHz (F={0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1}). For each bar in the figures, the average results for 1000 simulations are reported. For generating task sets, we employ the method which produces tasks according to the *UUnifast* method [24].

We compare our proposed method with three state-of-the-art power and reliability management techniques, i.e., EM [1], PPARM [17], and ReMap [5].

A. Impact of Increasing the Number of Cores

In this set of simulations, we investigate the impact of the number of cores on energy savings. For this set of simulations, the software-based fault detection mechanism (for which the fault coverage is 98.5%, the performance overhead is almost 100%, and the hardware overhead is near zero) has been adopted. The results are shown in the Fig. 3 to Fig. 6. In the Fig. 3 and Fig. 4, the task sets have dynamic-priority, while in the Fig. 5 and Fig. 6, the priorities of the task sets are static. What makes the results in Fig. 3 distinguished from the results of Fig. 4, is that the tasks' reliability target in Fig. 4 are higher than the corresponding values in Fig. 3. This is the same as for the Fig. 5 and Fig. 6; i.e., Fig. 6 depicts the results of simulations where the tasks have higher reliability target in comparison to the tasks of Fig. 5.

In general, it can be observed that by increasing the number of cores in the system, the trend of energy savings is decreasing. The reason for this observation is that by increasing the number of cores, the capability of the offline phase to reach the most energy-efficient configuration of each tasks increases. In their most energy-efficient configuration, tasks consume their minimum dynamic energy. On the other hand, increasing the number of cores leads to the increase of the static energy. Therefore, for the systems with a greater number of cores, the main part of the energy consumption is due to the static energy. In other words, for high number of cores, the tasks have almost reached their most-energy efficient configuration and the most dynamic energy savings has been achieved, thus the remaining energy consumption is due to the static energy that increase as the number of cores increase.

The discriminating point between the results from the dynamic- and static-priority task sets is that the overall energy savings of the static priority task sets are more than the dynamic-priority task sets. This is the result of the different utilization bounds for dynamic- and static-priority task set on each core. In static-priority task sets, the less value for the total utilization leads to allocating a smaller number of tasks to each core; hence, on each core, more opportunity for scaling the voltage and frequency of the task is available and the probability of achieving the most energy-efficient configuration for each task is higher.

There exists one interesting point in the results of the static-priority task set: the energy saving of the method where RMS scheduling is applied at the online phase is more than the method where the EDF scheduling is applied. These results admit the general rule which states that RMS is the best scheduling policy for static-priority task sets.

B. Impact of the System Load (Task Set Total Utilization)

In the second set of the simulations, we evaluate the effects of system load, i.e., the task set total utilization, on the energy savings of the system. The results are shown in Fig. 7 to Fig. 10. The results of simulating dynamic-priority task sets are shown in Fig. 7 and Fig. 8, and the results of static-priority tasks set are shown in Fig. 9 and Fig. 10. For the tasks of Fig. 8 and Fig. 10, the reliability targets are higher than the Fig. 7 and Fig. 9, respectively.

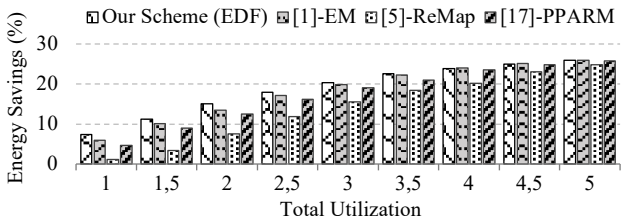


Fig. 7. Impact of increasing the system utilization for dynamic priority task set and R_{targ}=0.99999 on a system with 16 cores.

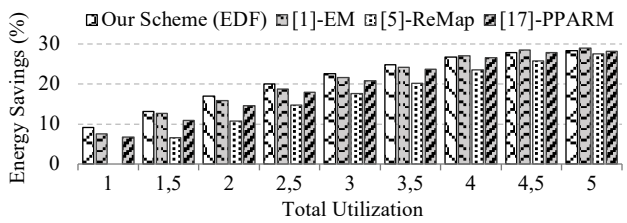


Fig. 8. Impact of increasing the system utilization for dynamic priority task set and R_{targ}=0.9999999 on a system with 16 cores.

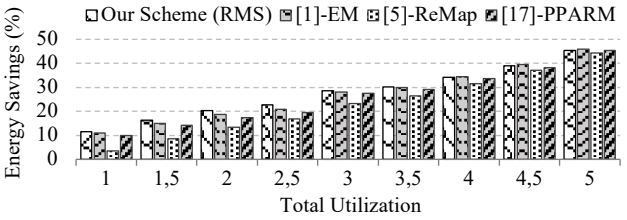


Fig. 9. Impact of increasing the system utilization on energy saving for static priority task set and R_{targ}=0.99999 on a system with 16 cores.

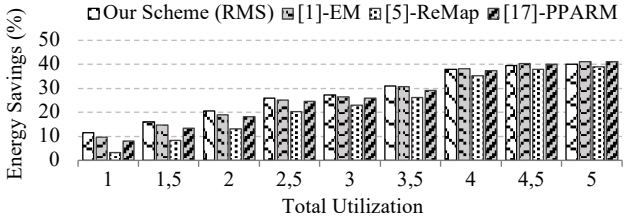


Fig. 10. Impact of increasing the system utilization on energy saving for static priority task set and R_{targ}=0.9999999 on a system with 16 cores.

Generally, in cases where both offline and online phases are applied, by increasing the system total utilization, more energy is saved, because when the total utilization is higher, there exists less opportunities for scaling the frequency and voltage of the tasks; thus, tasks should be executed in the configurations where they have higher voltage and frequency levels along with probable fewer number of copies. In these situations, by finishing a single copy of a task, the probability of executing the remaining parts of the other copies decreases (because the reliability of each copy is higher) which leads to more energy savings. On the other hand, in cases where just the offline phase has been applied, by increasing the system load, less energy is saved. This means that when the system load is higher, there is less chance for each task to reach its more energy-efficient configurations; so, when just the offline phase is applied, less energy savings can be obtained. Also, as it is expected, when the tasks' target reliabilities are higher the amounts of energy savings are less.

It should be noted that when the task sets have static-priorities, reaching the very high value of total utilization (like in the dynamic-priority task sets) is impossible. That is because of the fact that, for each core, the total admissible utilization is lower than the peer value for the dynamic-priority task sets; so, lower value of total utilization may make the system configuration feasible. Like for the previous set of simulations, here again it can be observed that for static-priority tasks sets, the cases in which the RMS scheduling is applied is more energy-efficient than the cases where EDF is applied, and the fact that RMS is the best-known scheduling policy for static-priority task sets is confirmed again.

C. *Impact of the Fault Detection Method*

In this set of simulations, we investigate the impact of fault detection method on energy consumption of the system. The results are shown in Fig. 11 and Fig. 12, for dynamic- and static-priority task sets respectively. As it was mentioned, no fault detection method is perfect and each one has some overhead. However, to the best of our knowledge, all the previous works have neglected this fact. In the previous works, not only has it been assumed that the fault coverage of the fault detection mechanism is 100%, but also the other implicit assumption is that the overheads of the fault detection mechanism is negligible. In our evaluation of the impacts of the fault detection mechanisms on the energy consumption of the system, we investigate different cases. As the baseline, we adopt an *optimistic* fault detection mechanism where, like the previous studies, the fault coverage of the fault detection is almost 100% and the overheads are near zero. In addition to the optimistic method, we consider two other methods; in one method, we assumed that the fault coverage is 98%, the hardware overhead is 17%, and the performance overhead is almost 3.5%; in another mechanism, we assumed that the fault coverage is 98.5%, the hardware overhead is negligible, and the performance overhead is nearly 100%. These two different mechanisms correspond to real fault detection mechanisms which are in use [16]. These two methods are specified by

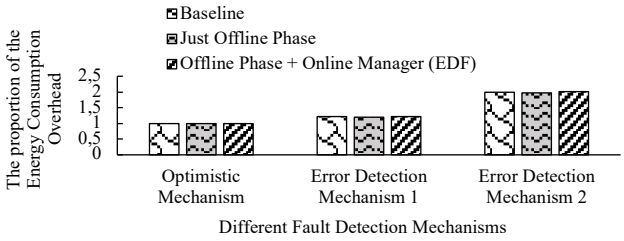


Fig. 11. The impact of the fault detection method on the energy consumption for dynamic priority task set.

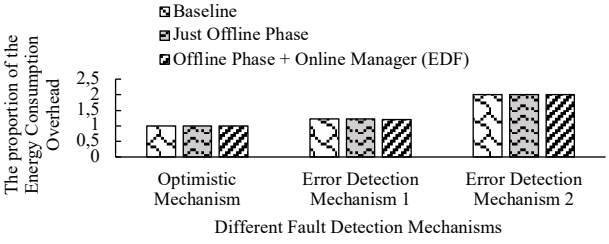


Fig. 12. The impact of the fault detection method on the energy consumption for static priority task set.

"Fault Detection Mechanism 1" and *"Fault Detection Mechanism 2"* in the figures.

As it can be observed, in both cases of dynamic- and static-priority task sets, the proportion of the energy consumption overheads of the Mechanism 2 is more than Mechanism 1, as the fault coverage and also the total overheads of Mechanism 2 is higher. The important point is that the proportion of the energy consumption overheads of both Mechanism 1 and Mechanism 2, in both dynamic- and static-priority task sets, is more than 1; it suggests a fundamental point that the overheads of the fault detection mechanisms should be considered, and considering the optimistic mechanism in which the fault coverage is 100%, and there is no hardware or performance overhead, is rather simplistic and not practical.

V. Conclusion

In this paper, we have proposed a joint energy and reliability management method for a set of periodic hard real-time tasks on a multicore safety-critical system. We also consider the impact of different fault detection mechanisms on the number of tasks' replicas to meet its reliability target and also on the energy consumptions of the system. For tackling with the aforementioned problem, we introduced an energy-reliability management method consisting of two phases. In the offline phase, after the determination of the task set priority type, we try to find the number of replicas and the proper frequency level for each task. At run time, when a copy of a task completes successfully, other copies of the same task are canceled, and then, the system goes to a low-energy state. Moreover, it has been observed that the effect of the fault detection mechanism on the configuration and energy consumption of the system is significant, and cannot be neglected.

References

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

[2] S. Safari, M. Ansari, G. Ershadi and S. Hessabi, "On the Scheduling of Energy-Aware Fault-Tolerant Mixed-Criticality Multicore Systems with Service Guarantee Exploration," in *IEEE Trans. on Par. and Dis. Sys.*, vol. 30, no. 10, pp. 2338-2354, 1 Oct. 2019.

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

[4] M. Salehi, and A. Ejlali, "A Hardware Platform for Evaluating Low-Energy Multiprocessor Embedded Systems Based on COTS Devices", *IEEE Trans. on Industrial Electronics, vo.* 62, no. 3, pp. 1262-1269, 2015.

[5] A. Yeganeh-Khaksar, M. Ansari, and A. Ejlali, "ReMap: Reliability Management of Peak-Power-Aware Real-Time Embedded Systems through Task Replication," in *IEEE Transaction on Emerging in Computing*, 2020.

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

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

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

[9] P. Pop, V. Izosimov, P. Eles and Z. Peng, "Design Optimization of Time- and Cost-Constrained Fault-Tolerant Embedded Systems with Checkpointing and Replication," in *IEEE Trans. on Very Large-Scale Integration (VLSI) Systems*, vol. 17, no. 3, pp. 389-402, March 2009.

[10] P. Marwedel, "Embedded System Design: Embedded Systems Foundations of Cyber-Physical Systems and the Internet of Things," in *Springer*, 2017.

[11] D. Zhu, R. Melhem and D. Mosse, "The effects of energy management on reliability in real-time embedded systems," *IEEE/ACM International Conference on Computer Aided Design*, San Jose, CA, USA, 2004, pp. 35-40.

[12] S. Ma. P. Dinakarrao, A. Joseph, A. Haridass, M. Shafique, J. Henkel, and H. Homayoun, "Application and Thermal-reliability-aware Reinforcement Learning Based Multi-core Power Management," in *J. Emerg. Technol. Comput. Syst.*, vol 15, no. 4, pp. 19, Oct 2019.

[13] S. Safari, H. Khdr, P. Gohari-Nazari, M. Ansari, S. Hessabi and J. Henkel, "TherMa-MiCs: Thermal-Aware Scheduling for Fault-Tolerant Mixed-Criticality Systems," in *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 7, pp. 1678-1694, 1 July 2022.

[14] H. Kopetz, "Real-Time Systems Design Principles for Distributed Embedded Applications," *Springer*, US, 2011.

[15] M. Ansari et al., "Thermal-Aware Standby-Sparing Technique on Heterogeneous Real-Time Embedded Systems," in *IEEE Transactions on Emerging Topics in Computing*, 2021.

[16] D. Gizopoulos et al., "Architectures for online error detection and recovery in multicore processors," 2011 Design, Automation & Test in Europe, Grenoble, 2011, pp. 1-6.

[17] M. Ansari, J. Saberlatibari, S. M. Pasandideh and A. Ejlali, "Simultaneous Management of Peak-Power and Reliability in Heterogeneous Multicore Embedded Systems," in *IEEE Trans. on Par. and Dis. Sys.*, vol. 31, no. 3, pp. 623-633, 1 March 2020.

[18] M. A. Haque, H. Aydin and D. Zhu, "Energy-aware task replication to manage reliability for periodic real-time applications on multicore platforms," *2013 International Green Computing Conference Proceedings*, Arlington, VA, 2013, pp. 1-11.

[19] V. Rathore, V. Chaturvedi, A. K. Singh, T. Srikanthan and M. Shafique, "Life Guard: A Reinforcement Learning-Based Task Mapping Strategy for Performance-Centric Aging Management," *ACM/IEEE Design Automation Conference (DAC)*, Las Vegas, NV, USA, 2019, pp. 1-6.

[20] 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 Trans. on Par. and Dis. Sys.*, vol. 30, no. 1, pp. 161-173, 1 Jan. 2019.

[21] F. R. Poursafaei, S. Safari, M. Ansari, M. Salehi and A. Ejlali, "Offline replication and online energy management for hard real-time multicore systems," *2015 CSI Symposium on Real-Time and Embedded Systems and Technologies (RTEST)*, Tehran, 2015, pp. 1-7.

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

[23] A. Yeganeh-Khaksar, M. Ansari, S. Safari, S. Yari-Karin and A. Ejlali, "Ring-DVFS: Reliability-Aware Reinforcement Learning-Based DVFS for Real-Time Embedded Systems," in *IEEE Embedded Systems Letters*, 2020.

[24] Bini, E. and Buttazo, G.C. "Measuring the Performance of Schedulability Tests," *Journal of Real-Time Systems,* vol. 30, pp. 129-154, 2005.

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

[26] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi, "McPAT: An integrated power, area, and timing modeling framework for multicore and manycore architectures," in *Proc. of the Annual Int'l Symposium on Microarchitecture*, 2009.

[27] 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.

[28] S. Safari, S. Hessabi and G. Ershadi, "LESS-MICS: A Low Energy Standby-Sparing Scheme for Mixed-Criticality Systems," in *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 12, pp. 4601-4610, Dec. 2020.