Improving Availability of Multicore Real-Time Systems Suffering Both Permanent and Transient Faults

Junlong Zhou, *Member, IEEE*, Xiaobo Sharon Hu, *Fellow, IEEE*, Yue Ma, *Student Member, IEEE*, Jin Sun, *Member, IEEE*, Tongquan Wei, *Senior Member, IEEE*, and Shiyan Hu, *Senior Member, IEEE*

Abstract—CMOS scaling has greatly increased concerns for both lifetime reliability due to permanent faults and soft-error reliability due to transient faults. Most existing works only focus on one of the two reliability concerns, but often times techniques used to increase one type of reliability may adversely impact the other type. A few efforts do consider both types of reliability together and use two different metrics to quantify the two types of reliability. However, for many systems, the user's concern is to maximize system availability by improving the mean time to failure (MTTF), regardless of whether the failure is caused by permanent or transient faults. Addressing this concern requires a uniform metric to measure the effect due to both types of faults. This paper introduces a novel analytical expression for calculating the MTTF due to transient faults. Using this new formula and an existing method to evaluate system MTTF, we tackle the problem of maximizing availability for multicore real-time systems with consideration of permanent and transient faults. A framework is proposed to solve the system availability maximization problem. Experimental results on a hardware board and simulation results of synthetic tasks show that our scheme significantly improves system MTTF (and hence availability) compared with existing techniques.

Index Terms—System availability, soft-error reliability, lifetime reliability, multicore real-time systems

1 Introduction

MULTICORE processors have become the mainstream for current and future embedded microprocessors in various real-time applications. However, the exponential increase in power density of multicore processors caused by aggressive technology scaling can lead to elevated operating temperature and frequent temperature variations, which accelerate chip wear-out due to electromigration (EM) [1], time-dependent dielectric breakdown (TDDB) [2], stress migration (SM) [3], and thermal cycling (TC) [4]. Such accelerated wear-outs eventually result in permanent faults occurring earlier and reduce lifetime. Furthermore, the decreasing feature size and operating voltage make the circuits more vulnerable to transient faults, thus degrade soft-error reliability. To reduce the cost of repairing/replacing an entire system and maintain quality of service, improving lifetime reliability (LTR) and soft-error reliability (SER) becomes an imperative design concern of multicore systems, especially for embedded systems deployed in critical applications and harsh environments.

As aforementioned, multicore systems are mainly susceptible to two faults: permanent fault resulting in faulty hardware and transient fault resulting in soft error. Permanent fault is a type of failure that continues to exist until the faulty hardware is repaired or replaced, and is caused by circuit wear-out [5], [6], [7]. Transient fault is a type of failure that appears for a short time and then disappears without damage to the device, and is caused by cosmic radiation [8], [9], [10]. Many safety-related embedded systems are required to have the capacity of providing a reliable execution in the presence of both faults.

Extensive investigations have been made in the design of reliability-aware real-time systems. However, most of them either focus on LTR [7], [11], [12], [13], [14] or SER [15], [16], [17], [18], [19]. Although the causes and repair techniques of transient and permanent faults are quite different, certain set ups of a chip in general impact both SER and LTR. For example, decreasing the core frequency would decrease SER but increase LTR. Therefore, it is important to consider the two faults at the same time when selecting the right set ups for the chip. Having a common metric would make it easier to balance the effects on SER and LTR. A few recent works

• J. Zhou and J. Sun are with the School of Computer Science and Engineering, Nanjing University of Science and Technology, Nanjing 210094, China. E-mail: {jlzhou, sunj}@njust.edu.cn.
• X. S. Hu and Y. Ma are with the Department of Computer Science and Engineering, University of Notre Dame, Notre Dame, IN 46656. E-mail: {shu, yma1}@nd.edu.
• T. Wei is with the School of Computer Science and Technology, East China Normal University, Shanghai 200062, China. E-mail: tqwei@cs.ecnu.edu.cn.
• S. Hu is with the Department of Electrical and Computer Engineering, Michigan Technological University, Houghton, MI 49931. E-mail: shiyan@mtu.edu.

*Manuscript received 15 May 2018; revised 25 June 2019; accepted 5 Aug. 2019. Date of publication 14 Aug. 2019; date of current version 5 Nov. 2019. (Corresponding author: Junlong Zhou.)
Recommended for acceptance by S. Ha.
Digital Object Identifier no. 10.1109/TC.2019.2935042*

[20], [21], [22], [23], [24] have examined both SER and LTR together. But all these works utilize separate metrics to evaluate reliability. Specifically, mean time to failure (MTTF) is used to measure LTR while the probability of successful execution is used to estimate SER. However, evaluating LTR and SER with different metrics presents two dilemmas. First, for system users, the concern is the mean time to first failure, regardless of whether the failure is caused by a permanent fault or transient fault. It is true that repairing the two different failures incurs different overheads, but at the end of the day, any failure would cause an interruption of normal execution. Second, certain design decisions (e.g., task mapping and voltage/frequency scaling) may increase LTR but decrease SER or vice versa. Without a common metric, it is difficult to gauge how tradeoffs should be made to achieve overall high system reliability. To this end, we use the MTTF as the common metric to evaluate LTR and SER.

In this paper, we first propose using MTTF to evaluate LTR and SER, and present a novel analytical method to calculate the MTTF due to transient faults. We then formulate the problem of maximizing the availability for real-time systems running on multicores in the presence of both permanent and transient faults. Finally, we design a hybrid framework to solve the system availability maximization problem. This paper makes the following contributions.

- We propose an analytical method to calculate the MTTF due to transient faults for a core executing a given workload. In addition, we show that our MTTF expression indeed correctly represents MTTF. Based on the expression, we formulate a max-min problem to optimize the availability of multicore real-time systems that suffer from both permanent and transient faults.
- We present a hybrid framework to solve the availability maximization problem, in which an offline and an online approach are alternately employed to improve the system availability based on the core states. The offline approach builds on reliability-aware methods for increasing system availability by balancing SER and LTR. The online approach improves the system availability by balancing the diverse availabilities of individual cores. MTTF-aware heuristics are used in the online approach to adjust the strategy generated from the offline approach.

The rest of this paper is organized as follows. Section 2 shows the background of this work. Section 3 introduces the analytical method to calculate the MTTF due to transient faults. Section 4 defines the optimization problem and presents an overview of our framework to solve the problem. Our framework consists of an offline approach and an online approach, which are described in Sections 5 and 6, respectively. Experimental results are discussed in Section 7 and concluding remarks are given in Section 8.

2 Background

2.1 Related Work

Considerable research efforts have been devoted to investigating permanent faults in the past decade. Huang et al. [11] established an analytical model to estimate the LTR of multicores and designed a simulated annealing based method to maximize system lifetime. Amrouch et al. [12] studied the impact of individual aging mechanisms on the probability of failures and the interdependencies of these mechanisms. Duque et al. [13] developed an LTR model that considers the variation of fault behaviors at runtime, and proposed an adaptive LTR-driven scheduling approach. Unlike the approaches in [11], [12], [13] that solve the LTR optimization problem statically, Chantem et al. [14] presented an online LTR-aware task scheduling which slows down core wear-out speed, and Ma et al. [7] designed an online framework which maximizes LTR through core utilization control. However, none of the above-mentioned work takes into account transient faults.

On the other hand, a lot of studies have focused on improving SER. Zhao et al. [15] explored the optimal frequency for each task to maximize system SER under the deadline and energy constraints. Rozo et al. [16] presented an adaptive fault-tolerant technique to improve SER by dynamically tuning resource allocation. Haque et al. [17] tackled the problem of minimizing the energy consumed by real-time tasks under a given SER constraint. Although these proposed techniques can either lower fault rate or tolerate occurred transient faults, they cannot handle the uncertainty in transient fault occurrences. Zhou et al. [18] introduced a fault adaptation variable to model the uncertainty, and proposed a stochastic fault-tolerant task scheduling algorithm. However, all the aforementioned approaches do not deal with permanent faults. Axer et al. [19] presented an SER analysis approach to detect and recover transient faults while keeping time predictability for periodic task sets executing in mixed-critical systems. Though [19] has made impressive contributions in fault detection and recovery, it does not consider two specific aspects related to system SER improvements: 1) some design decisions such as task mapping and task operating frequency selection have great impacts on reliability, and 2) lifetime is also an important concern in embedded systems. Our work attempts to address these missing aspects.

A few recent papers have focused on handling permanent and transient faults simultaneously. An efficient fault-aware resource management method is designed for network-on-chip systems [20]. The authors proposed placing spare cores to improve system SER and LTR. Huang et al. [21] developed a software/hardware recovery based scheduling algorithm to deal with both permanent and transient faults. A genetic algorithm based approach is introduced to jointly improve SER and LTR by determining the mapping and frequency for each task [22]. Kim et al. [23] presented energy and lifetime optimization techniques that use DVFS-aware reliability model and Q-learning-based method for multicore systems considering permanent and transient faults. Ma et al. [24] established an online framework for enhancing SER and LTR of real-time systems running on Big-Little type MPSoCs. However, all these works lack a uniform metric to measure the effect due to both faults, thus are not suitable for addressing the user's concern of maximizing system availability regardless of whether the failure is caused by permanent or transient faults.

MTTF is used as the common metric to evaluate system reliability. Aliee et al. [25] presented a success tree based scheme to carry out reliability analysis for embedded systems suffering both permanent and transient faults. The approach can model more general component dependencies instead of the serial failure model used in our paper. However, it uses a Monte-Carlo (MC) analysis to compute the overall system MTTF. The MC analysis requires designers to specify a time step for discretizing the continuous-time reliability functions for both permanent and transient fault processes and thus is time-consuming. Unlike [25], we analytically derive the MTTF due to transient faults and use a simplified version of the LTR modeling tool [26] to obtain the MTTF due to permanent faults. The analytical method avoids the use of MC analysis and the simplified tool significantly reduces the number of MC trials with an acceptable accuracy degradation. Therefore, our approach is more computational-efficient. Using the common metric MTTF, we formulate and solve the problem of maximizing availability for general real-time applications running on multicore systems with consideration of permanent and transient faults.[1]

2.2 Architecture and Application Model

We consider a multicore system $\mathcal{C}$ that consists of M homogeneous cores $\{\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_M\}$. $\mathcal{C}$ executes a task set $\mathcal{T}$ that consists of N independent periodic tasks $\{\tau_1, \tau_2, \ldots, \tau_N\}$ with real-time constraints. The characteristics of a task τ_i $(1 \leq i \leq N)$ is described by a quadruplet $\tau_i : \{p_i, d_i, wt_i, \rho_i\}$, where p_i and d_i are the period and relative deadline of task τ_i, respectively. wt_i is the worst-case execution time of task τ_i at the core's maximum frequency, and ρ_i is the task vulnerability factor indicating the probability that a transient fault at the hardware level ultimately leads to a program failure at the task level [28]. The period of a task is assumed equal to its deadline [8], i.e., $p_i = d_i$. The hyper-period of task set $\mathcal{T}$, denoted by $\mathcal{H}_{\mathcal{T}}$, is the least common multiple of all task periods $\{p_1, p_2, \ldots, p_N\}$.

The task set is periodically executed on the multicore system, and each core is dynamic voltage and frequency scaling (DVFS) enabled and supports a discrete set of frequencies varying from the minimum voltage/frequency to the maximum voltage/frequency. For simplicity, we will use the term frequency change to stand for both supply voltage and frequency adjustments in what follows. Denoting the minimum and maximum frequency supported by a core as $f_{\min}$ and $f_{\max}$, the operating frequency f_i of task τ_i running on the multicore system, normalized with respect to $f_{\max}$, satisfies $0 < f_{\min} \leq f_i \leq f_{\max} = 1.0$. The execution time t_i of task τ_i at frequency f_i is then given by wt_i/f_i.

2.3 Soft-Error Reliability Model

Transient faults are in general modeled using an exponential distribution with an average arrival rate λ, which represents the expected number of failures occurring per second and increases as the voltage/frequency decreases [9]. Let $\lambda(f_i)$ denote the raw fault rate of a core running at frequency f_i, it then can be calculated by

$$\lambda(f_i) = \lambda_0 \cdot 10^{\frac{\alpha(1-f_i)}{1-f_{\min}}}, \tag{1}$$

where λ_0 is the average fault rate at the maximum frequency $f_{\max}$, and α is a hardware specific constant that indicates the sensitivity of fault rates to frequency scaling. Considering the task vulnerability, the actual fault rate of executing task τ_i on a core at frequency f_i is then $\lambda(f_i) \cdot \rho_i$.

The SER of a task is defined as the probability of its successful execution without the occurrence of any transient faults, and can be determined by the exponential failure law. Using the exponential distribution assumption, the SER of a task instance (job) of τ_i is expressed as [10]

$$R_i = e^{-\lambda(f_i)\cdot\rho_i\cdot\frac{wt_i}{f_i}}. \tag{2}$$

Replication is widely used to improve SER. In this paper, we consider multicore systems that use replication to tolerate up to one transient fault for each task since single-fault-tolerance is a common assumption [10]. Given task τ_i executing at frequency f_i with a recovery task running at the same frequency, the SER of the task is calculated as

$$R_i^{\text{rep}} = 1 - \left(1 - e^{-\lambda(f_i)\cdot\rho_i\cdot\frac{wt_i}{f_i}}\right)^2. \tag{3}$$

2.4 Lifetime Reliability Model

We consider four IC-dominant failure mechanisms: EM, TDDB, SM, and TC. Other failure mechanisms such as negative/positive bias temperature instability can be incorporated using the sum-of-fault rate model [3], [22].

EM refers to dislocation of metal atoms caused by momentum imparted by electrical current in wires and vias [1]. The MTTF due to EM is

$$MTTF_{\text{EM}} = \frac{A_{\text{EM}}}{G^{\alpha}} e^{\frac{E_{\text{act}_{\text{EM}}}}{\delta T}}, \tag{4}$$

where A_{EM} is a constant determined by the physical characteristics of the metal interconnect, G is the current density, $E_{\text{act}_{\text{EM}}}$ is the active energy for electromigration, α is an empirically determined constant, δ is the Boltzmann constant, and T is the runtime temperature.

TDDB refers to deterioration of the gate oxide layer [2]. The MTTF due to time-dependent dielectric breakdown is

$$MTTF_{\text{TDDB}} = A_{\text{TDDB}} \left(\frac{1}{V}\right)^{(\vartheta_1 - \vartheta_2 T)} e^{\frac{A+B/T+CT}{\delta T}}, \tag{5}$$

where A_{TDDB} is a fitting constant, V is the supply voltage, and ϑ_1, ϑ_1, A, B, and C are empirical fitting parameters.

SM is caused by the directionally biased motion of atoms in metal wires due to mechanical stress caused by thermal mismatch between metal and dielectric materials [3]. The MTTF resulting from stress migration is

$$MTTF_{\text{SM}} = A_{\text{SM}} |T_0 - T|^{-\alpha} e^{\frac{E_{\text{act}_{\text{SM}}}}{\delta T}}, \tag{6}$$

where A_{SM} is a fitting constant, T_0 is the mental deposition temperature during fabrication, and $E_{\text{act}_{\text{SM}}}$ is the activation energy for stress migration.

TC refers to wear due to thermal stress induced by mismatched coefficients of thermal expansion for adjacent material layers [4]. The number of cycles to failure (N_{TC}) can be calculated as

$$N_{\text{TC}} = A_{\text{TC}} (\Delta T - T_{\text{th}})^q e^{\frac{E_{\text{act}_{\text{TC}}}}{\delta T_{\max}}}, \tag{7}$$

1. The preliminary version of this manuscript appears in [27].

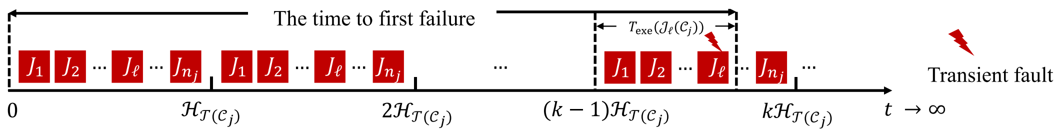

The time to first failure
$T_{\text{exe}}(\mathcal{J}_\ell(\mathcal{C}_j))$
J_1 J_2 ⋯ J_ℓ ⋯ J_{n_j}
Transient fault
0
$\mathcal{H}_{\mathcal{T}(\mathcal{C}_j)}$
$2\mathcal{H}_{\mathcal{T}(\mathcal{C}_j)}$
$(k-1)\mathcal{H}_{\mathcal{T}(\mathcal{C}_j)}$
$k\mathcal{H}_{\mathcal{T}(\mathcal{C}_j)}$
$t \to \infty$

Fig. 1. The time to first failure due to a transient fault of job J_ℓ in the kth run of $\mathcal{J}(\mathcal{C}_j)$. $\mathcal{J}(\mathcal{C}_j) = \{J_1, \ldots, J_{n_j}\}$ represents all the jobs of set $\mathcal{T}(\mathcal{C}_j)$ in a hyper-period $\mathcal{H}_{\mathcal{T}(\mathcal{C}_j)}$ and $T_{\text{exe}}(\mathcal{J}_\ell(\mathcal{C}_j))$ is the total execution time of jobs J_1 to J_ℓ on core $\mathcal{C}_j$.

where A_{TC} is an empirically determined constant, ΔT is the thermal cycle amplitude and can change between cycles, Tth is the temperature at which inelastic deformation begins, q is the Coffin-Manson exponent constant, $E_{\text{act}_{\text{TC}}}$ is the activation energy for thermal cycling, and $T_{\max}$ is the maximal temperature during the cycle.

Although the scaling parameters of the above four failure mechanisms can be quite different, the aging effects caused by these failure mechanisms can be dealt with simultaneously as done by existing work, e.g., in [3], [26]. We leverage a hierarchical LTR modeling tool [26] to estimate the system-level MTTF due to permanent faults when considering the four failure mechanisms. The tool models wear due to the above four mechanisms at the device level. The tool accounts for the effect of using multiple devices in a component upon fault distributions. Based on the device-level reliability models and temporal failure distributions, component-level MTTF is calculated [26]. Then, using the component-level reliability as input, the system-level MTTF is obtained by MC simulation. The efficacy of this hierarchical modeling tool has been validated in [26]. The tool is shown to be accurate and efficient in estimating system-level LTR and thus it has been widely adopted.

3 MTTF Due to Transient Faults

A simple way to estimate the MTTF due to transient faults ($MTTF_T$) is calculating the reciprocal of average failure rate. However, this only holds if SER follows an exponential distribution. Though the simplest model for SER can be described by an exponential distribution as shown in Eq. (2), after considering task recovery techniques, the reliability model no longer follows an exponential distribution as shown in Eq. (3). Since we are interested in system MTTF when executing a given workload, unfortunately we cannot simply use $1/\lambda$ to estimate the $MTTF_T$. In this paper, we introduce an analytical approach for calculating the $MTTF_T$ by using task-level SERs, which are obtained based on the failure rate model. Obtaining $MTTF_T$ enables the designers to utilize a common metric to evaluate LTR and SER, which allows to make decisions for achieving overall high system reliability. This section first introduces an analytical method to calculate the $MTTF_T$ for one core and gives a simple example to illustrate the calculation, then proves some properties of the MTTF expression, and finally shows that the expression indeed correctly represents MTTF.

We represent the set of tasks allocated to core $\mathcal{C}_j$ $(1 \le j \le M)$ by $\mathcal{T}(\mathcal{C}_j)$, which satisfies $\mathcal{T}(\mathcal{C}_j) \subset \mathcal{T}$ and consists of r_j tasks $\{\tau_1, \tau_2, \ldots, \tau_{r_j}\}$. The hyper-period of task set $\mathcal{T}(\mathcal{C}_j)$ is represented by $\mathcal{H}_{\mathcal{T}(\mathcal{C}_j)}$. Since each task in set $\mathcal{T}(\mathcal{C}_j)$ generates a sequence of jobs within its period, we use $\mathcal{J}(\mathcal{C}_j) = \{J_1, J_2, \ldots, J_{n_j}\}$ to represent all the jobs of $\mathcal{T}(\mathcal{C}_j) = \{\tau_1, \tau_2, \ldots, \tau_{r_j}\}$ in $\mathcal{H}_{\mathcal{T}(\mathcal{C}_j)}$, where $n_j = \sum_{i=1}^{r_j} \mathcal{H}_{\mathcal{T}(\mathcal{C}_j)}/p_i$ is the number of jobs on core $\mathcal{C}_j$ during a hyper-period $\mathcal{H}_{\mathcal{T}(\mathcal{C}_j)}$.

We denote the mean time to first failure of core $\mathcal{C}_j$ due to transient faults by $MTTF_T(\mathcal{C}_j)$. To derive $MTTF_T(\mathcal{C}_j)$, the difficulty is not in computing the SER of tasks or multiplying the SER of individual tasks, but is in modeling the time to first failure due to transient faults and the corresponding probability of the first failure, as well as doing the integration based on them. Fig. 1 illustrates the time to first failure due to a transient fault occurring during the execution of job J_ℓ $(1 \le \ell \le n_j)$ in the kth run of set $\mathcal{J}(\mathcal{C}_j)$. Here $T_{\text{exe}}(\mathcal{J}(\mathcal{C}_j)) = \sum_{\ell=1}^{n_j} t_\ell$ is the total execution time of jobs in set $\mathcal{J}(\mathcal{C}_j)$, $T_{\text{exe}}(\mathcal{J}_\ell(\mathcal{C}_j)) = \sum_{i=1}^{\ell} t_i$ is the total execution time of jobs J_1 to J_ℓ on core $\mathcal{C}_j$, and $\mathcal{J}_\ell(\mathcal{C}_j)$ is the set of jobs $J_1, J_2, \ldots, J_\ell$. As shown in Fig. 1, the time to first failure is equal to $(k-1)\cdot \mathcal{H}_{\mathcal{T}(\mathcal{C}_j)} + T_{\text{exe}}(\mathcal{J}_\ell(\mathcal{C}_j))$, where $\mathcal{H}_{\mathcal{T}(\mathcal{C}_j)}$ is assumed equal to $T_{\text{exe}}(\mathcal{J}(\mathcal{C}_j))$ considering that the idle time without task execution in $\mathcal{H}_{\mathcal{T}(\mathcal{C}_j)}$ is deemed reliable and thus is not included in the calculation of the time to first failure. Now, let $P_{\text{succ}}(\mathcal{J}(\mathcal{C}_j), k-1)$ be the probability that the first $k-1$ runs of $\mathcal{J}(\mathcal{C}_j)$ are all successful, and $P_{\text{fail}}(J_\ell)$ be the probability that J_ℓ is erroneous but J_1 to $J_{\ell-1}$ in the same run of $\mathcal{J}(\mathcal{C}_j)$ are successful. Then $MTTF_T(\mathcal{C}_j)$ is

$$MTTF_T(\mathcal{C}_j) = \sum_{k=1}^{\infty} \sum_{\ell=1}^{n_j} \{(k-1)\cdot \mathcal{H}_{\mathcal{T}(\mathcal{C}_j)} + T_{\text{exe}}(\mathcal{J}_\ell(\mathcal{C}_j))\} \cdot P_{\text{succ}}(\mathcal{J}(\mathcal{C}_j), k-1) \cdot P_{\text{fail}}(J_\ell). \tag{8}$$

Using Eq. (8) to compute MTTF directly is challenging due to the infinite number of summation terms. However, by applying a series of algebraic transformations, we can remove these terms and derive a simple expression to calculate MTTF. Below, we show key steps of the transformation.

Based on the definition of $P_{\text{succ}}(\mathcal{J}(\mathcal{C}_j), k-1)$ and $P_{\text{fail}}(J_\ell)$, we have

$$P_{\text{succ}}(\mathcal{J}(\mathcal{C}_j), k-1) = \left(\prod_{\ell=1}^{n_j} R_\ell\right)^{k-1} \tag{9}$$

$$P_{\text{fail}}(J_\ell) = R_1 R_2 \cdots R_{\ell-1}(1-R_\ell), \tag{10}$$

where R_ℓ is the reliability of job J_ℓ and can be obtained using Eq. (2). Let $P_{\text{fail}}(\mathcal{J}(\mathcal{C}_j))$ be the probability that $\mathcal{J}(\mathcal{C}_j)$ sees a failure in a run, it can be then expressed as

$$P_{\text{fail}}(\mathcal{J}(\mathcal{C}_j)) = 1 - \prod_{\ell=1}^{n_j} R_\ell. \tag{11}$$

Thus the probability $P_{\text{succ}}(\mathcal{J}(\mathcal{C}_j), k-1)$ can be written as

$$P_{\text{succ}}(\mathcal{J}(\mathcal{C}_j), k-1) = (1 - P_{\text{fail}}(\mathcal{J}(\mathcal{C}_j)))^{k-1}. \tag{12}$$

According to Eqs. (9), (10), (11), and (12), we can rewrite Eq. (8) as

TABLE 1
Task Execution Time at the Maximum Frequency, Period, Vulnerability to Soft Error, and Deadline

Task	Execution Time (ms)	Period (ms)	Vulnerability to Soft-Error	Deadline (ms)
τ_1	100	300	0.5	300
τ_2	140	400	0.6	400
τ_3	190	600	0.7	600

$$\begin{aligned} MTTF_T(\mathcal{C}_j) = \sum\nolimits_{k=1}^{\infty}(k-1)(1-P_{\text{fail}}(\mathcal{J}(\mathcal{C}_j)))^{k-1} \cdot \mathcal{H}_{\mathcal{T}(\mathcal{C}_j)} \\ \cdot P_{\text{fail}}(\mathcal{J}(\mathcal{C}_j)) + \sum\nolimits_{k=1}^{\infty}(1-P_{\text{fail}}(\mathcal{J}(\mathcal{C}_j)))^{k-1} \\ \cdot \sum\nolimits_{\ell=1}^{n_j} T_{\text{exe}}(\mathcal{J}_\ell(\mathcal{C}_j)) \cdot P_{\text{fail}}(J_\ell). \end{aligned} \tag{13}$$

Since $\sum_{k=1}^{\infty}(k-1)a^{k-1} = \frac{a}{(1-a)^2}$ and $\sum_{k=1}^{\infty} a^{k-1} = \frac{1}{1-a}$, we can derive that

$$\sum\nolimits_{k=1}^{\infty}(k-1)(1-P_{\text{fail}}(\mathcal{J}(\mathcal{C}_j)))^{k-1} = \frac{1-P_{\text{fail}}(\mathcal{J}(\mathcal{C}_j))}{(P_{\text{fail}}(\mathcal{J}(\mathcal{C}_j)))^2}, \tag{14}$$

$$\sum\nolimits_{k=1}^{\infty}(1-P_{\text{fail}}(\mathcal{J}(\mathcal{C}_j)))^{k-1} = \frac{1}{P_{\text{fail}}(\mathcal{J}(\mathcal{C}_j))}. \tag{15}$$

Finally, $MTTF_T(\mathcal{C}_j)$ can be calculated as

$$\begin{aligned} MTTF_T(\mathcal{C}_j) &= \frac{\mathcal{H}_{\mathcal{T}(\mathcal{C}_j)} + T_{\text{exp}}(\mathcal{J}(\mathcal{C}_j))}{P_{\text{fail}}(\mathcal{J}(\mathcal{C}_j))} - \mathcal{H}_{\mathcal{T}(\mathcal{C}_j)} \\ &= \frac{T_{\text{exe}}(\mathcal{J}(\mathcal{C}_j)) + T_{\text{exp}}(\mathcal{J}(\mathcal{C}_j))}{P_{\text{fail}}(\mathcal{J}(\mathcal{C}_j))} - T_{\text{exe}}(\mathcal{J}(\mathcal{C}_j)). \end{aligned} \tag{16}$$

$T_{\text{exp}}(\mathcal{J}(\mathcal{C}_j))$ in Eq. (16) is the expected time to failure when the fault occurs in the first run, and it is expressed as

$$T_{\text{exp}}(\mathcal{J}(\mathcal{C}_j)) = \sum\nolimits_{\ell=1}^{n_j} T_{\text{exe}}(\mathcal{J}_\ell(\mathcal{C}_j)) \cdot P_{\text{fail}}(J_\ell). \tag{17}$$

According to the above transformation, given the execution time and reliability of each job, the $MTTF_T$ given in Eq. (16) can be readily evaluated, where the reliability of jobs (i.e., $R_1, R_2, \ldots, R_\ell$) are derived by Eqs. (1) and (2). Note that we cannot ignore $T_{\text{exe}}(\mathcal{J}(\mathcal{C}_j))$ and $T_{\text{exp}}(\mathcal{J}(\mathcal{C}_j))$ in Eq. (16) to further simplify $MTTF_T$ since doing this would result in non-negligible error.

In Eq. (16), $\mathcal{H}_{\mathcal{T}(\mathcal{C}_j)}$ is assumed equal to $T_{\text{exe}}(\mathcal{J}(\mathcal{C}_j))$. This assumption works well for heavily loaded systems (i.e., systems that are busy) but results in larger approximation error for lightly loaded systems. The reasons why it is acceptable to assume that the system under consideration is heavily loaded (i.e., busy) are given as follows. First, in a heavily loaded system, accelerated aging of cores caused by heavy workloads makes the multicore processor prone to suffer permanent faults. Second, long task execution times in a heavily loaded system expose the processor to more transient faults. Thus, it is more critical to consider permanent and transient faults simultaneously and making tradeoff between LTR and SER to achieve overall high system reliability in heavily loaded systems than lightly loaded systems. We believe that our analytical model can be improved to reduce approximation error for lightly loaded systems by judiciously calculating the slack time. We leave the detailed discussion of this aspect to future work.

For better understanding, we provide a simple example to illustrate the calculation of $MTTF_T$. This example considers three tasks τ_1, τ_2, τ_3 on a core running at the maximum frequency. The task-related parameters are presented in Table 1. From the table, we can easily derive the execution time of these tasks during a hyper-period. That is, $T_{\text{exe}} = 0.1 \times 4 + 0.14 \times 3 + 0.19 \times 2 = 1.2$s. Using the earliest-deadline-first policy [29], we can determine the scheduling order of tasks during a hyper-period. Assuming that the fault rate λ_0 at the maximum frequency is 1.0×10^{-7} [15], the reliability of $\tau_1 - \tau_3$ can be obtained using Eq. (2) as $R_1 = 0.999999995$, $R_2 = 0.9999999916$, $R_3 = 0.9999999867$. Given the task reliabilities, we can use Eqs. (11) and (17) to calculate P_{fail} and T_{exp} of the task set, respectively. That is, $P_{\text{fail}} = 7.180 \times 10^{-8}$ and $T_{\text{exp}} = 1.058 \times 10^{-8}$s. Finally, substituting T_{exe}, T_{exp}, and P_{fail} into Eq. (16), we have $MTTF_T = \frac{1.2+1.058\times10^{-8}}{7.180\times10^{-8}} - 1.2 = 1.671 \times 10^7$s $= 0.53$ year. We also investigate the value of $MTTF_T$ in Section 7 using other fault rate values (i.e., $\lambda_0 = 1.0 \times 10^{-6}$ and $\lambda_0 = 1.0 \times 10^{-8}$) suggested from [15].

The $MTTF_T(\mathcal{J}(\mathcal{C}_j))$ expression in Eq. (16) is derived for a workload $\mathcal{J}(\mathcal{C}_j)$ being periodically executed. One immediate question is what the $MTTF_T(\mathcal{J}(\mathcal{C}_j))$ would be if we treat two or more runs of $\mathcal{J}(\mathcal{C}_j)$ as the given workload being periodically repeated. Clearly this new $MTTF_T(\mathcal{J}(\mathcal{C}_j))$ based on multiple runs of $\mathcal{J}(\mathcal{C}_j)$ should be exactly the same as the one in Eq. (16). We introduce a theorem below to show that computing $MTTF_T(\mathcal{J}(\mathcal{C}_j))$ using Eq. (16) indeed draws the desired conclusion.

Theorem 1. *For any given job set $\mathcal{J}(\mathcal{C}_j)$ and any integer m (≥ 2), let $\mathcal{J}^m(\mathcal{C}_j)$ denote the set containing m runs of $\mathcal{J}(\mathcal{C}_j)$, i.e., $\mathcal{J}^m(\mathcal{C}_j) = \{\underbrace{J_1, J_2, \ldots, J_{n_j}, J_1, J_2, \ldots, J_{n_j}, \cdots, J_1, J_2, \ldots, J_{n_j}}_{n_j \times m}\}$. Then $MTTF_T(\mathcal{J}^m(\mathcal{C}_j)) = MTTF_T(\mathcal{J}(\mathcal{C}_j))$ holds if both are evaluated following the expression in Eq. (16).*

Theorem 1 can be proved by a series of inductions. Its proof is given in the Appendix, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TC.2019.2935042. The theorem demonstrates that the calculation of $MTTF_T$ given in Eq. (16) indeed satisfies the basic property that $MTTF_T$ should be independent of the number of runs of a given periodical task set used to calculate $MTTF_T$.

4 Overall Framework

As pointed out earlier, the user's goal is to maximize system availability, no matter whether the system failure is caused by permanent or transient faults. In this section, we first describe the studied problem and then provide a high-level overview of our approach to solve this problem.

4.1 Problem Definition

System availability is defined as the ratio of system uptime to the sum of system uptime and downtime [30]. Since uptime is quantified by $MTTF$ and downtime is equal to $MTTR$, system availability is written as

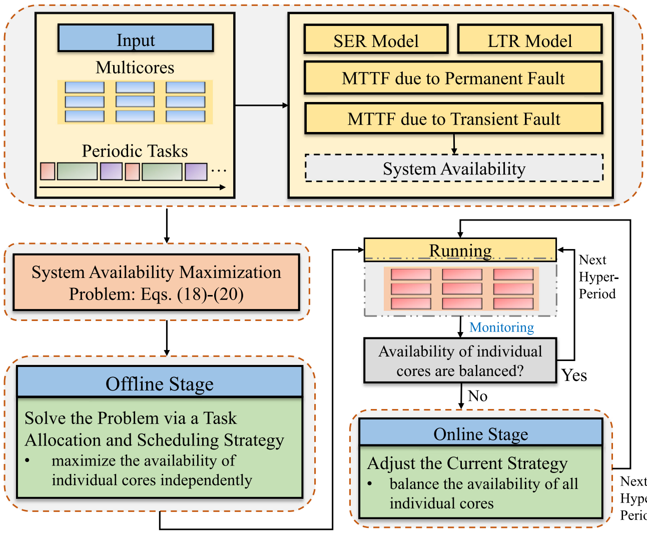

Input
Multicores
Periodic Tasks
SER Model
LTR Model
MTTF due to Permanent Fault
MTTF due to Transient Fault
System Availability
System Availability Maximization Problem: Eqs. (18)-(20)
Offline Stage
Solve the Problem via a Task Allocation and Scheduling Strategy
• maximize the availability of individual cores independently
Running
Monitoring
Availability of individual cores are balanced?
Yes
No
Next Hyper-Period
Online Stage
Adjust the Current Strategy
• balance the availability of all individual cores
Next Hyper-Period

Fig. 2. Overview of HAOF.

$$\mathcal{A} = \frac{MTTF}{MTTF + MTTR}, \tag{18}$$

where $MTTR$ is the mean time to repair [30]. For each core that may suffer from both transient and permanent faults, the availability of core $\mathcal{C}_j$ is computed as

$$\mathcal{A}(\mathcal{C}_j) = Min : \left\{ \frac{MTTF_T(\mathcal{C}_j)}{MTTF_T(\mathcal{C}_j) + MTTR_T(\mathcal{C}_j)}, \frac{MTTF_P(\mathcal{C}_j)}{MTTF_P(\mathcal{C}_j) + MTTR_P(\mathcal{C}_j)} \right\}, \tag{19}$$

where $MTTF_T(\mathcal{C}_j)$ is calculated using our formula given in Eq. (16) and $MTTF_P(\mathcal{C}_j)$ can be obtained using the LTR modeling tool [26]. $MTTR_T(\mathcal{C}_j)$ and $MTTR_P(\mathcal{C}_j)$ are constants once recovery techniques are determined. Maximizing the availability of core $\mathcal{C}_j$ then becomes

$$Max\ \mathcal{A}(\mathcal{C}_j) = Max\ Min : \left\{ \frac{MTTF_T(\mathcal{C}_j)}{MTTF_T(\mathcal{C}_j) + MTTR_T(\mathcal{C}_j)}, \frac{MTTF_P(\mathcal{C}_j)}{MTTF_P(\mathcal{C}_j) + MTTR_P(\mathcal{C}_j)} \right\}. \tag{20}$$

Through a series of transformations (see the Appendix, available in the online supplemental material), maximizing the core availability $\mathcal{A}(\mathcal{C}_j)$ given in Eq. (19) is equivalent to maximizing the core MTTF in the presence of permanent and transient faults. The core MTTF is given by

$$MTTF(\mathcal{C}_j) = Min : \{\Upsilon_j \cdot MTTF_T(\mathcal{C}_j), MTTF_P(\mathcal{C}_j)\}, \tag{21}$$

where $\Upsilon_j = MTTR_P(\mathcal{C}_j)/MTTR_T(\mathcal{C}_j)$. Accordingly, the objective of maximizing the system availability is equivalent to maximizing the system MTTF (denoted by $MTTF_{\text{sys}}$) in the presence of two faults, which is expressed as

$$Max\ MTTF_{\text{sys}} = Max\ Min_j MTTF(\mathcal{C}_j),\ \forall j = 1, 2, \ldots, M. \tag{22}$$

Clearly, we formulate the problem of maximizing MTTF (and hence availability) for a multicore as a max-min optimization problem, i.e., maximizing the minimum MTTF of all cores. The max-min formulation is an approximation to more precise but very complex ways to compute the MTTF of a multicore system using the serial failure model (i.e., the system fails if any one core fails), thus has been used frequently in the literature. The error resulted by this approximation becomes larger if the MTTF of each core is closer to one another and if the number of cores is larger. Otherwise, the error would be small. Since we aim to improve overall system MTTF especially when loads are quite different, using the max-min approximation is acceptable.

In the above formulation (Eq. (22)) we adopt the serial failure model. It is possible the system can tolerate some core failures, e.g., through re-mapping tasks from the faulty core to a spare core if the system has some spare cores (e.g., cores not turned on due to the dark-silicon concern [31]). Our optimization framework may be extended to this type of setups, but needs to significantly revise the assumptions, architecture and failure models as well as consider the timing overheads of task communication and migration. As for how to revise the models and assumptions as well as how to use spare cores for enhancing system reliability, it is not the focus of this paper and is a different problem [20]. We leave the detailed discussion of this aspect to future work.

Since the system may suffer from both transient and permanent faults, we focus on handling the two faults simultaneously and solving the problem of maximizing system availability by improving SER and LTR. Specifically, the problem is described as follows. Given a task set $\mathcal{T}$ to be executed periodically on the multicore system $\mathcal{C}$, design a strategy including (i) the allocation of N tasks to M cores (ii) determining whether any tasks should be replicated and (iii) the frequency that each task should be executed at in each hyper-period, in order to maximize system MTTF (and hence system availability) while satisfying design constraints. Formally, we intend to solve the following

$$\begin{aligned} &Maximize\ MTTF_{\text{sys}} = Min_j\ MTTF(\mathcal{C}_j),\ \forall j = 1, 2, \ldots, M, \\ &Subject\ to\ f_{\min} \le f_i \le f_{\max},\ \forall i = 1, 2, \ldots, N, \end{aligned} \tag{23}$$

$$wt_i/f_i \le d_i,\ \forall i = 1, 2, \ldots, N. \tag{24}$$

Eq. (23) is introduced to bound the operating frequency of tasks and Eq. (24) captures the real-time requirement that each task needs to be finished before its deadline.

4.2 Overview of HAOF

We propose a hybrid availability optimization framework (HAOF) that consists of an offline approach and an online approach to improve the availability for real-time systems running on multicores in the presence of both permanent and transient faults. The offline approach solves the max-min problem defined in Eqs. (22), (23), and (24) statically, and the online approach dynamically tunes the task allocation and scheduling strategy obtained from the offline approach to further increase system availability.

A high-level depiction of HAOF is given in Fig. 2. HAOF operates as follows. In the first hyper-period, HAOF adopts the task allocation and scheduling strategy generated by the offline approach to execute tasks. The offline approach improves the system availability by maximizing the

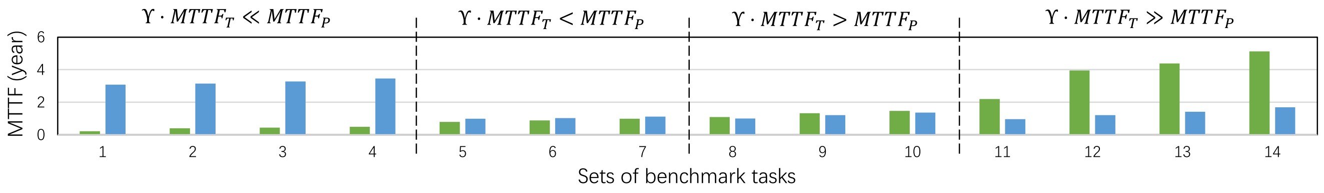


Fig. 3. $\Upsilon \cdot MTTF_T$ and $MTTF_P$ for 14 different sets of benchmark tasks.

availability of individual cores separately. After executing tasks on cores, the states (including wear, temperatures, and frequency setups) of all cores can be derived. At the end of the first as well as each subsequent hyper-period, HAOF checks whether the availability of individual cores are balanced. If the availability of all cores are balanced, indicating that the current strategy can achieve the maximum system availability since no one core would fail much earlier than others, no adjustment is needed for the next hyper-period. Otherwise, the current strategy needs an adjustment. HAOF then utilizes the online approach, which improves the system availability by balancing the availability of all cores, to decide the strategy for the next hyper-period.

The two main components of HAOF, offline and online approaches, are introduced in the following two sections.

5 Offline Stage

The offline stage aims to improve the availability of the entire multicore system by maximizing the availability of individual cores independently. The approach builds on four reliability-aware methods, which are designed for increasing the availability of cores for different states. Since three of the four reliability-aware methods are either simple to realize or have been widely explored, most of the discussion will be on the fourth method. This section introduces the details of our offline approach and our concerned method.

5.1 Maximize the Availability of A Core

The key issue in our offline approach is how to maximize the availability of a core. To solve the problem of maximizing the MTTF (and hence availability) of a core (e.g., $\mathcal{C}_j$), we need to first determine $MTTF_T(\mathcal{C}_j)$ and $MTTF_P(\mathcal{C}_j)$. Using our proposed analytical method to calculate $MTTF_T(\mathcal{C}_j)$ and the system-level LTR modeling tool [26] to estimate $MTTF_P(\mathcal{C}_j)$, this can be readily achieved. Depending on which reliability dominates, different methods may be adopted accordingly to maximize system availability. We group the relationship between $\Upsilon_j \cdot MTTF_T(\mathcal{C}_j)$ and $MTTF_P(\mathcal{C}_j)$ into four scenarios when running task set $\mathcal{T}(\mathcal{C}_j)$ under various core states (i.e., wear, temperature profiles, and frequency setups):

Scenario 1. $\Upsilon_j \cdot MTTF_T(\mathcal{C}_j) \ll MTTF_P(\mathcal{C}_j)$;
Scenario 2. $\Upsilon_j \cdot MTTF_T(\mathcal{C}_j) < MTTF_P(\mathcal{C}_j)$;
Scenario 3. $\Upsilon_j \cdot MTTF_T(\mathcal{C}_j) > MTTF_P(\mathcal{C}_j)$;
Scenario 4. $\Upsilon_j \cdot MTTF_T \gg MTTF_P(\mathcal{C}_j)$.

Before discussing how to handle each of the four scenarios, we first justify that all four scenarios indeed exist. We have carried out several simulations to evaluate $\Upsilon_j \cdot MTTF_T(\mathcal{C}_j)$ and $MTTF_P(\mathcal{C}_j)$ for different benchmark programs, core wear states, temperature profiles, and frequencies. Benchmark tasks (representing automotive, network, office, security, telecommunication, and consumer applications from MiBench [32]) and ALPHA 21264 microprocessor [33] are used in the simulations[2]. The results of $\Upsilon_j \cdot MTTF_T(\mathcal{C}_j)$ and $MTTF_P(\mathcal{C}_j)$ are summarized in Fig. 3, which clearly show that all four scenarios exist depending on the actual benchmarks. From the above, it is clear that we compare the frequencies of permanent faults and transient faults by including the factor Υ. For better understanding, we give a simple example to sketch out how the frequency of permanent faults can be comparable to that of transient ones. If transient fault occurs twice a day while permanent failure occurs once in two years, assuming Υ is 1500 (within the range of $[1000, 10000]$), then $MTTF_P = 2$ year and $\Upsilon \cdot MTTF_T = 1500 \times 0.5$ day $= 2.05$ year are comparable. Besides, there are actual systems (e.g., the brake-by-wire system [35]) in which the rates of permanent and transient faults are indeed comparable.

The offline approach (OFA) is summarized in Algorithm 1, which is a combination of four reliability-aware methods for handling the four different scenarios listed above. OFA first generates an initial assignment to cores and the operating frequencies of tasks using an input heuristic such as those in [7], [17], and adopts the EDF scheduling policy [29] to schedule the tasks (line 1). Using the EDF scheduling policy, an incoming job with the earliest deadline is executed first. Note that in the following algorithms, checking the task schedulabiliity under EDF is well known [36], [37], [38] and hence omitted. Given the initial workloads and frequency settings of cores, OFA then evaluates the MTTF of cores (line 3) and invokes different reliability-aware methods to maximize the availability of cores based on the different scenarios (lines 4-13).

Specifically, for each core $\mathcal{C}_j$, if $MTTF_P(\mathcal{C}_j)$ is far greater than $\Upsilon_j \cdot MTTF_T(\mathcal{C}_j)$ (scenario 1), the core's availability is dominated by $\Upsilon_j \cdot MTTF_T(\mathcal{C}_j)$ and Full Replication and Speedup can be safely used to increase $MTTF_T(\mathcal{C}_j)$ (lines 5-6). Full Replication and Speedup refers to the strategy that each original task has a recovery task, and both original and recovery task are executed at the maximum frequency. For this scenario, other existing techniques such as [9], [39], [40] can be used to increase $MTTF_T(\mathcal{C}_j)$. On the other hand, if $\Upsilon_j \cdot MTTF_T(\mathcal{C}_j)$ is far greater than $MTTF_P(\mathcal{C}_j)$ (scenario 4), LTR-aware Strategy as in [11], [12], [13], [14] can be used to maximize $MTTF_P(\mathcal{C}_j)$ and hence improve core availability (lines 10-11).

If $MTTF_P(\mathcal{C}_j)$ is greater than $\Upsilon_j \cdot MTTF_T(\mathcal{C}_j)$ but not by too much (scenario 2), when maximizing core availability, the negative effect on LTR due to executing an increased

2. The simulation results here are not meant to be comprehensive but just to demonstrate that all four scenarios identified can occur. In the simulation, $MTTF_T$ is calculated using Eq. (16), same as the example given in Section 3. $MTTF_P$ is derived using the LTR modeling tool [26]. The detailed settings of this tool for ALPHA 21264 microprocessor can be found in [34]. The value of Υ is set to 10.

workload (caused by replication) at the maximum frequency should be taken into account. Thus, we adopt Partial Replication and Speedup to select a subset of tasks to have recovery tasks and execute them at the maximum frequency (lines 7-8). On the other hand, if $\Upsilon_j \cdot MTTF_T(\mathcal{C}_j)$ is greater than $MTTF_P(\mathcal{C}_j)$ but not by too much (scenario 3), DVFS-based Strategy as in [22], [23] that reduce the operating temperature for increasing $MTTF_P(\mathcal{C}_j)$ and consider the negative effect of DVFS on SER can be applied to handle this scenario (lines 12-13).

Algorithm 1. Offline Approach to Improve System Availability

```
1  generate an initial task and frequency assignment by an
   input heuristic (e.g., SER/LTR-aware schemes [7], [17])
   and determine the task sequence using the EDF [29];
2  for $j = 1$ to $M$ do
3    calculate $MTTF_T(\mathcal{C}_j)$ using Eq. (16) and derive $MTTF_P(\mathcal{C}_j)$
     using the tool [26] based on workload $\mathcal{T}(\mathcal{C}_j)$ and frequency
     setup $\mathcal{F}(\mathcal{C}_j)$;
4    if $\Upsilon_j \cdot MTTF_T(\mathcal{C}_j) < MTTF_P(\mathcal{C}_j)$ then
5      if $\Upsilon_j \cdot MTTF_T(\mathcal{C}_j) \ll MTTF_P(\mathcal{C}_j)$ then
6        Full Replication and Speedup (scenario 1);
7      else
8        Partial Replication and Speedup (scenario 2);
9    else
10     if $\Upsilon_j \cdot MTTF_T(\mathcal{C}_j) \gg MTTF_P(\mathcal{C}_j)$ then
11       LTR-aware Strategy (scenario 4);
12     else
13       DVFS-based Strategy (scenario 3);
```

From Algorithm 1, readers can easily find that our OFA, which considers availability, actually compares $MTTF_P$ and $\Upsilon \cdot MTTF_T$. Thus the relative magnitude of $MTTF_P$ and $MTTF_T$ are more important in determining which reliability improvement approach should be used. Our OFA has considered all four possible scenarios which may occur in different application systems. Among the four approaches used in OFA, Full Replication and Speedup is simple to implement, DVFS-based Strategy and LTR-Aware Strategy have been explored in the literature. Therefore, we focus on discussing the Partial Replication and Speedup strategy for scenario 2 in the rest of this section. Note that our proposed analytical method for calculating the MTTF due to transient faults is still applicable to the approaches designed for other scenarios.

5.2 Partial Replication and Speedup

Replication and speed selection can be used to maximize core MTTF (and hence availability) for the scenario $\Upsilon_j \cdot MTTF_T(\mathcal{C}_j) < MTTF_P(\mathcal{C}_j)$ since it improves the SER due to transient faults, and simultaneously limits the negative effect on hardware aging due to the execution of an increased number of tasks. Some researchers [22], [41] have investigated the impact of selective replication and partial speedup on SER and LTR, and observed that the technique is effective in balancing the two reliabilities. However, the existing work does not have a specific task replication and frequency selection strategy to maximize system availability when considering both transient and permanent faults. In this paper, we propose a Partial Replication and Speedup (PRS) approach to maximize core availability for the scenario $\Upsilon_j \cdot MTTF_T(\mathcal{C}_j) < MTTF_P(\mathcal{C}_j)$.

Note that PRS only allows a task to have at most one recovery or raise the task's frequency to the maximum frequency, which means that the solution space considered in PRS is a subset of and smaller than the entire solution space of the studied problem. It is possible to use existing optimization methods (e.g., genetic algorithm) to search the entire solution space, similar to the approaches in [22], [42]. However, an offline information often deviates from actual information such that spending a lot of time optimizing may not be as beneficial. In addition, offline approach is generally used to produce conservative solutions which could be further improved online. In PRS, rather than slightly raise frequency, we increase a task's operating frequency to a core's maximum frequency (leading to the fastest aging speed). By doing this, conservative solutions can be ensured by PRS. As for why PRS does not replicate a task and raise its operating frequency simultaneously to increase SER, it's because either of the two operations by itself is already capable of greatly improving the SER of the task.

5.2.1 Replication versus Speedup for a Single Task

Since the MTTF of core $\mathcal{C}_j$ is the minimum of $\Upsilon_j \cdot MTTF_T(\mathcal{C}_j)$ and $MTTF_P(\mathcal{C}_j)$, the main idea of PRS is to maximize $MTTF_T(\mathcal{C}_j)$ when $\Upsilon_j \cdot MTTF_T(\mathcal{C}_j)$ is below $MTTF_P(\mathcal{C}_j)$. We achieve this by iteratively making the best choice (replication or speedup) for each task in terms of improving SER. For this approach to be effective, the key is to select the "right" tasks to have a recovery task and the "right" tasks to execute at the maximum frequency.

Let RS_i be defined as

$$RS_i = \begin{cases} 1, & \text{if } \tau_i \text{ is selected to have a recovery task} \\ 0, & \text{if } \tau_i \text{ is selected to execute at frequency } f_{\max} \end{cases}.$$

As described in Section 2.3, replicating a task can tolerate one transient fault and hence improve SER, and increasing task operating frequency leads to exponentially decreasing transient fault rate and hence also improve SER. According to Eqs. (1), (2), and (3), the SER increment of task τ_i achieved by replication and speedup are given by $\Delta R_i^r = R_i^{\text{rep}} - R_i$ and $\Delta R_i^s = R_{i|f_i=f_{\max}} - R_i$, respectively. Since PRS needs to decide the selection of tasks for replication or speedup, we introduce a metric ΔR_i^{r-s} to compare replication with speedup with respect to improve SER, i.e.,

$$\Delta R_i^{r-s} = \Delta R_i^r - \Delta R_i^s = 2R_i - R_i^2 - R_{i|f_i=f_{\max}}. \quad (25)$$

Clearly, $\Delta R_i^{r-s} > 0$ indicates replication is better for task τ_i and $\Delta R_i^{r-s} < 0$ indicates speedup is better. Referring to Eq. (2), we have that ΔR_i^{r-s} is a function of task frequency f_i. Furthermore, it is a monotonically increasing function of f_i as stated in the following theorem. The proof of Theorem 2 can be found in the Appendix, available in the online supplemental material.

Theorem 2. *ΔR_i^{r-s} as defined in Eq. (25) increases monotonically as f_i increases.*

According to Theorem 2, given a specific frequency f_i for task τ_i, ΔR_i^{r-s} must belong to one of the cases illustrated in

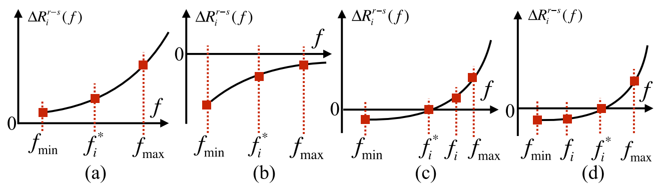


Fig. 4. An illustration of four cases of ΔR_i^{r-s} values, where $\Delta R_i^{r-s} > 0$ for (a) and (c) and $\Delta R_i^{r-s} < 0$ for (b) and (d).

Fig. 4. Note that $f_{\min}$ and $f_{\max}$ are given, f_i is the operating frequency of task τ_i, and f_i^* corresponds to the frequency where $\Delta R_i^{r-s}(f_i^*) = 0$ and can be obtained by Eqs. (2) and (25). The significance of Theorem 2 is that, after obtaining ΔR_i^{r-s} values for the four points, one can immediately decide whether replication ($RS_i = 1$) or speedup ($RS_i = 0$) should be chosen to maximize SER. We summarize the basic steps needed to determine RS_i for task τ_i in Algorithm 2. The algorithm takes as inputs f_i, $f_{\min}$, $f_{\max}$, and f_i^*. It first initializes the value of f_i, then calculates the values of $\Delta R_i^{r-s}(f_{\min})$ and $\Delta R_i^{r-s}(f_{\max})$ and compares the values of f_i and f_i^*, and finally decides the value of RS_i according to the four cases illustrated in Fig. 4.

5.2.2 PRS Heuristic

Algorithm 2 decides whether a task should be replicated or sped up in order to maximize SER. However, if each task is replicated or sped up, LTR may be degraded. We now present a heuristic to decide which task should be replicated/sped up and which should be left alone so as to maximize system availability when the system is in the concerned scenario ($\Upsilon MTTF_T < MTTF_P$). At a high level, the heuristic works as follows. It first uses Algorithm 2 to determine whether replication or speedup should be applied to each task. Based on the replication/speedup choice for each task given by Algorithm 2, it iteratively decides whether the replication/speedup choice for a task should actually be adopted. Some tasks may end up neither replicated nor sped up since doing so would lead to degraded system availability.

Algorithm 2. $RS(f_i, f_{\min}, f_{\max}, f_i^*)$

```
1  $f_i = f_i^{\text{init}}$;// $f_i^{\text{init}}$ is derived by line 1 of Algorithm 1
   if $\Delta R_i^{r-s}(f_{\min}) > 0$ then
2  $RS_i = 1$; // $\Delta R_i^{r-s}(f_i) > \Delta R_i^{r-s}(f_{\min}) \Rightarrow \Delta R_i^r > \Delta R_i^s$
3  if $\Delta R_i^{r-s}(f_{\max}) < 0$ then
4     $RS_i = 0$; // $\Delta R_i^{r-s}(f_i) < \Delta R_i^{r-s}(f_{\max}) \Rightarrow \Delta R_i^r < \Delta R_i^s$
5  if $f_i > f_i^*$ && $R_i^{r-s}(f_i^*) = 0$ then
6     $RS_i = 1$; // $\Delta R_i^{r-s}(f_i) > \Delta R_i^{r-s}(f_i^*) \Rightarrow \Delta R_i^r > \Delta R_i^s$
7  if $f_i < f_i^*$ && $R_i^{r-s}(f_i^*) = 0$ then
8     $RS_i = 0$; // $\Delta R_i^{r-s}(f_i) < \Delta R_i^{r-s}(f_i^*) \Rightarrow \Delta R_i^r < \Delta R_i^s$
```

Rather than blindly determining the increase/decrease in $MTTF_T/MTTF_P$, our PRS judiciously controls the increase/decrease in $MTTF_T/MTTF_P$ to achieve a state in which $\Upsilon \cdot MTTF_T$ and $MTTF_P$ are equal and the resultant core MTTF is higher. The details of our heuristic PRS are given in Algorithm 3. The algorithm takes as inputs the core workload $\mathcal{T}(\mathcal{C}_j)$ and frequency setup $\mathcal{F}(\mathcal{C}_j)$, the minimum/maximum frequency $f_{\min}/f_{\max}$, and a given constant Υ_j capturing the ratio of $MTTR_P$ to $MTTR_T$ of core $\mathcal{C}_j$. The algorithm starts by determining the replication/speedup choice RS_i for each task τ_i using Algorithm 2 (lines 1-2). Temporary task set $\mathcal{T}_{\text{temp}}$ is used to store candidate tasks when deciding which task's replication and speedup choice should actually be adopted. It keeps a copy of $\mathcal{T}(\mathcal{C}_j)$ at first and will be updated as each decision is made (line 3). The system's current workload $\mathcal{T}_{\text{cur}}$ is initialized to $\mathcal{T}(\mathcal{C}_j)$ (i.e., no replication and using initial frequency) (line 3). The algorithm then iteratively compares the value of $\Upsilon_j \cdot MTTF_T$ and $MTTF_P$ for the core's current workload. When the core is in the concerned scenario (line 4), the algorithm first computes the increase in MTTF for each candidate task $\tau_i \in \mathcal{T}_{\text{temp}}$ if τ_i is replicated (lines 6-7) or sped up (lines 8-9). The task that could lead to the maximum increase in MTTF is found (lines 10-13) and is allowed to use its replication (lines 14-15) or speedup choice (lines 16-17). The system's current workload is then updated, and the task is removed from $\mathcal{T}_{\text{temp}}$ (line 18). Finally, when the core is no longer in the concerned scenario, the algorithm returns the achieved core MTTF (line 19).

6 Online Stage

Intuitively, the MTTF of a multicore system is maximized when the MTTF of individual cores are balanced, so no one core fails much earlier than others. To this end, our online stage aims to improve the system MTTF by balancing the MTTF of all cores. The balanced MTTF is achieved by iteratively exploiting the core with the maximum MTTF, $\mathcal{C}_{\max}$, to increase the MTTF of the core with the minimum MTTF, $\mathcal{C}_{\min}$. This is because the system MTTF is determined by $\mathcal{C}_{\min}$ and $\mathcal{C}_{\max}$ is most capable of increasing the MTTF of $\mathcal{C}_{\min}$. The improvement in MTTF of $\mathcal{C}_{\min}$ achieved by sacrificing the MTTF of $\mathcal{C}_{\max}$ is realized by MTTF-aware task reassignment or replication for the two cores. This section introduces our online approach and MTTF-aware task reassignment/replication heuristics in detail.

Algorithm 3. $PRS(\mathcal{T}(\mathcal{C}_j), \mathcal{F}(\mathcal{C}_j), f_{\min}, f_{\max}, \Upsilon_j)$

```
1  for $i = 1$ to $r_j$ do
2     determine $RS_i$ by Algorithm 2;
3  $\mathcal{T}_{\text{temp}} = \mathcal{T}(\mathcal{C}_j)$ and $\mathcal{T}_{\text{cur}} = \mathcal{T}(\mathcal{C}_j)$;
4  while $\Upsilon_j \cdot MTTF_T(\mathcal{T}_{\text{cur}}) < MTTF_P(\mathcal{T}_{\text{cur}})$ do
5     for $i = 1$ to $sizeof(\mathcal{T}_{\text{temp}})$ do
6        if $RS_i = 1$ then
7           $\Delta\Upsilon_j \cdot MTTF_{T,i} = \Upsilon_j \cdot MTTF_T(\mathcal{T}_{\text{cur}} + \tau_i) - \Upsilon_j \cdot MTTF_T(\mathcal{T}_{\text{cur}})$;
8        else
9           $\Delta\Upsilon_j \cdot MTTF_{T,i} = \Upsilon_j \cdot MTTF_T(\mathcal{T}_{\text{cur}} - \tau_i + \tau_{i|f_i=f_{\max}}) - \Upsilon_j \cdot$
           $MTTF_T(\mathcal{T}_{\text{cur}})$;
10       if $i = 1$ then
11          $\Delta\Upsilon_j \cdot MTTF_T^{\max} = \Delta\Upsilon_j \cdot MTTF_{T,i}$ and $\ell = 1$;
12       if $\Delta\Upsilon_j \cdot MTTF_{T,i} > \Delta\Upsilon_j \cdot MTTF_T^{\max}$ then
13          $\Delta\Upsilon_j \cdot MTTF_T^{\max} = \Delta\Upsilon_j \cdot MTTF_{T,i}$ and $\ell = i$;
14    if Task deadline constraint holds for $\forall\tau_j \in \mathcal{T}_{\text{cur}} + \tau_\ell$ and
      $RS_\ell = 1$ then
15       $\mathcal{T}_{\text{cur}} = \mathcal{T}_{\text{cur}} + \tau_\ell$;
16    else
17       $\mathcal{T}_{\text{cur}} = \mathcal{T}_{\text{cur}} - \tau_\ell + \tau_{\ell|f_\ell=f_{\max}}$;
18    $\mathcal{T}_{\text{temp}} = \mathcal{T}_{\text{temp}} - \tau_\ell$;
19 return $MTTF(\mathcal{T}_{\text{cur}})$ (i.e., $MTTF(\mathcal{C}_j)$) using Eq. (21);
```

6.1 Balance the Availability of Cores

The key challenge in our online approach is how to balance the MTTF of M cores. We tackle this challenge by iteratively using the best core $\mathcal{C}_{\max}$ to improve the worst core $\mathcal{C}_{\min}$ in terms of MTTF. To accomplish this MTTF tradeoff, we need to first find the two cores that are respectively associated with the maximum MTTF (represented by $MTTF^{\max}_{\text{core}}$) and the minimum MTTF (represented by $MTTF^{\min}_{\text{core}}$). The calculation of $MTTF^{\min}_{\text{core}}$ and $MTTF^{\max}_{\text{core}}$ are

$$\begin{cases} MTTF^{\min}_{\text{core}} = Min: \ \{MTTF(\mathcal{C}_1), \ldots, MTTF(\mathcal{C}_M)\}, \\ MTTF^{\max}_{\text{core}} = Max: \ \{MTTF(\mathcal{C}_1), \ldots, MTTF(\mathcal{C}_M)\}. \end{cases} \tag{26}$$

After identifying cores $\mathcal{C}_{\min}$ and $\mathcal{C}_{\max}$, the online approach performs MTTF-aware task reassignment or replication operation for the two cores depending on which type of reliability dominates the availability of $\mathcal{C}_{\min}$. The pseudocode of the online approach is described in Algorithm 4.

Algorithm 4. Online Approach to Improve System Availability

```
1  for $j = 1$ to $M$ do
2      calculate core MTTF, $MTTF(\mathcal{C}_j)$, using Eq. (21);
3  derive the minimum and maximum MTTF (i.e., $MTTF^{\min}_{\text{core}}$
   and $MTTF^{\max}_{\text{core}}$) of $M$ cores using Eq. (26);
4  $Pre = 0$ and $Cur = MTTF^{\min}_{\text{core}}$;
5  while $MTTF^{\min}_{\text{core}} < MTTF^{\max}_{\text{core}}$ and $Pre \neq Cur$ do
6      $Pre = MTTF^{\min}_{\text{core}}$;
7      if $MTTF^{\min}_{\text{core}} = MTTF_P(\mathcal{C}_{\min})$ then
8          call $LMF\_Reassign$, as given in Algorithm 5;
           // Largest-$\Delta MTTF$-First (LMF) Reassignment
9      else
10         call $LMF\_Replicate$, as given in Algorithm 6;
           // Largest-$\Delta MTTF$-First (LMF) Replication
11     compute $MTTF(\mathcal{C}_{\min})$ and $MTTF(\mathcal{C}_{\max})$ using Eq. (21);
12     obtain $MTTF^{\min}_{\text{core}}$ and $MTTF^{\max}_{\text{core}}$ using Eq. (26);
13     update $\mathcal{C}_{\min}$, $\mathcal{C}_{\max}$, $\mathcal{T}(\mathcal{C}_{\min})$, $\mathcal{T}(\mathcal{C}_{\max})$, $\mathcal{F}(\mathcal{C}_{\min})$, $\mathcal{F}(\mathcal{C}_{\max})$;
14     $Cur = MTTF^{\min}_{\text{core}}$;
15 return $MTTF_{\text{sys}} = Min: \ \{MTTF(\mathcal{C}_1), \ldots, MTTF(\mathcal{C}_M)\}$;
```

Algorithm 4 first computes the MTTF of M cores using Eq. (21) (lines 1-2). Then, it obtains the minimum MTTF $MTTF^{\min}_{\text{core}}$ and maximum MTTF $MTTF^{\max}_{\text{core}}$ of M cores using Eq. (26) (line 3). Variables Pre and Cur are used to test if the MTTF of M cores are balanced, and are initialized to 0 and $\mathcal{A}^{\min}_{\text{core}}$, respectively (line 4). If the MTTF of cores are not balanced, the algorithm iteratively uses core $\mathcal{C}_{\max}$ to help increase the MTTF of core $\mathcal{C}_{\min}$ (lines 5-14). In each iteration, the algorithm checks which MTTF (i.e., $MTTF_P$ or $MTTF_T$) dominates the MTTF of core $\mathcal{C}_{\min}$. If the MTTF of $\mathcal{C}_{\min}$ is determined by $MTTF_P(\mathcal{C}_{\min})$, indicating that LTR (measured by $MTTF_P$) of the core needs to be improved for maximizing MTTF, the assignments of tasks to $\mathcal{C}_{\min}$ and $\mathcal{C}_{\max}$ are adjusted using $LMF\_Reassign$ (Algorithm 5), to be discussed in Section 6.2, which judiciously selects a task and moves the selected task from core $\mathcal{C}_{\min}$ to core $\mathcal{C}_{\max}$ (lines 7-8). Among all the tasks originally assigned to $\mathcal{C}_{\min}$, the reassignment of the selected task leads to the largest $\Delta MTTF_P$. If the MTTF of $\mathcal{C}_{\min}$ is determined by $MTTF_T(\mathcal{C}_{\min})$, indicating that SER (measured by $MTTF_T$) of the core needs to be improved for maximizing MTTF, a task is selected from core $\mathcal{C}_{\min}$ and replicated on core $\mathcal{C}_{\max}$ using $LMF\_Replicate$ (Algorithm 6), to be discussed in Section 6.2. Among all the tasks originally assigned to core $\mathcal{C}_{\min}$, the replication of the selected task leads to the largest $\Delta MTTF_T$ (lines 9-10).

Note that task reassignment and replication are both performed under the deadline constraint. After task reassignment or replication, the $MTTF(\mathcal{C}_{\min})$ and $MTTF(\mathcal{C}_{\max})$ are calculated using Eq. (21). Then $MTTF^{\min}_{\text{core}}$ and $MTTF^{\max}_{\text{core}}$ can be readily derived using Eq. (26) (lines 11-12). At the end of each iteration, the algorithm updates $\mathcal{C}_{\min}$, $\mathcal{C}_{\max}$, $\mathcal{T}(\mathcal{C}_{\min})$, $\mathcal{T}(\mathcal{C}_{\max})$, $\mathcal{F}(\mathcal{C}_{\min})$, $\mathcal{F}(\mathcal{C}_{\max})$ accordingly, and sets Cur to $MTTF^{\min}_{\text{core}}$ (lines 13-14). This process repeats until core $\mathcal{C}_{\max}$ cannot be used to increase MTTF of core $\mathcal{C}_{\min}$. When the terminating condition is met, the algorithm returns the improved system availability (line 15).

6.2 MTTF-Aware Task Reassignment and Replication

We have designed the MTTF-aware task reassignment and replication (i.e., $LMF\_Reassign$ and $LMF\_Replicate$) in the online approach to maximize the MTTF of core $\mathcal{C}_{\min}$, depending on which reliability dominates the availability of the core. Specifically, if the MTTF of core $\mathcal{C}_{\min}$ is decided by $MTTF_P(\mathcal{C}_{\min})$, $LMF\_Reassign$ is activated to improve the $MTTF_P$ of core $\mathcal{C}_{\min}$; otherwise, $LMF\_Replicate$ is activated to improve the $MTTF_T$ of core $\mathcal{C}_{\min}$. Both operations are performed in the same manner, that is, iteratively selecting the best task on $\mathcal{C}_{\min}$ in terms of increasing MTTF to reassign to $\mathcal{C}_{\max}$ or replicate on $\mathcal{C}_{\max}$. The pseudocodes of these two procedures are described in Algorithms. 5 and 6, respectively.

Algorithm 5 uses temporary task set $\mathcal{T}_{\text{temp}}$ to store candidate tasks when deciding which task should be selected for reassignment to improve the LTR of core $\mathcal{C}_{\min}$. $\mathcal{T}_{\text{temp}}$ keeps a copy of $\mathcal{T}_{\min}$ at first (line 1) where $\mathcal{T}_{\min}$ is the set of tasks on core $\mathcal{C}_{\min}$, and will be updated as each selection is made (line 12). The algorithm then iteratively compares the value of $MTTF(\mathcal{C}_{\min})$ and $MTTF(\mathcal{C}_{\max})$. If core $\mathcal{C}_{\max}$ can be used to help core $\mathcal{C}_{\min}$, i.e., $MTTF(\mathcal{C}_{\min}) < MTTF(\mathcal{C}_{\max})$, the algorithm first computes the increase in $MTTF_P$ for each candidate task $\tau_i \in \mathcal{T}_{\text{temp}}$ if τ_i is moved from core $\mathcal{C}_{\min}$ to core $\mathcal{C}_{\max}$ (lines 2-4). The task that could lead to the maximum increase in $MTTF_P$ is identified (lines 5-8) and allowed to be reassigned if the deadlines of the tasks on core $\mathcal{C}_{\max}$ can all be met (line 9). Subsequently, the task set of cores $\mathcal{C}_{\min}$ and $\mathcal{C}_{\max}$ and their frequency setups $\mathcal{F}_{\min}$ and $\mathcal{F}_{\max}$ are updated (line 10). The new MTTF of both cores after task reassignment is calculated using Eq. (21) (line 11). Finally, if no cores can be used to help core $\mathcal{C}_{\min}$, i.e., the set of tasks on core $\mathcal{C}_{\min}$ is empty or the availability of all cores are balanced, the algorithm exits (lines 13-14).

Algorithm 6 has the similar workflow as Algorithm 5. A temporary set $\mathcal{T}_{\text{temp}}$ is also used to store candidate tasks when deciding which task should be selected for replication to improve the SER of core $\mathcal{C}_{\min}$. It is initialized to $\mathcal{T}_{\min}$ (line 1) and will be updated as each selection is made (line 13). Algorithm 6 also iteratively compares the value of $MTTF(\mathcal{C}_{\min})$ and $MTTF(\mathcal{C}_{\max})$. If core $\mathcal{C}_{\max}$ can be used to help core $\mathcal{C}_{\min}$, i.e., $MTTF(\mathcal{C}_{\min}) < MTTF(\mathcal{C}_{\max})$, the algorithm first computes the increase in $MTTF_T$ for each candidate task $\tau_i \in \mathcal{T}_{\text{temp}}$ if τ_i is replicated on core $\mathcal{C}_{\max}$ (lines 2-5). The task leading to the maximum increase in $MTTF_T$ is identified (lines 6-9) and is allowed to be replicated if deadlines are satisfied (line 10). The task set of cores $\mathcal{C}_{\min}$ and

$\mathcal{C}_{\max}$ and their frequency setups $\mathcal{F}_{\min}$ and $\mathcal{F}_{\max}$ are then updated (line 11). The new MTTF of both cores after task replication can be derived using Eq. (21) (line 12). Finally, the algorithm exits when no cores are available to help core $\mathcal{C}_{\min}$ (lines 14-15).

Algorithm 5. $LMF\_Reassign(\mathcal{T}, \mathcal{C}, \mathcal{F})$

```
1  $\mathcal{T}_{\text{temp}} = \mathcal{T}_{\min}$;
2  while $MTTF(\mathcal{C}_{\min}) < MTTF(\mathcal{C}_{\max})$ do
3    for $i = 1$ to $size(\mathcal{T}_{\text{temp}})$ do
4      $\Delta MTTF_P(\tau_i) = MTTF_P(\mathcal{T}_{\min} - \tau_i, \mathcal{C}_{\min}, \mathcal{F}_{\min} - f_i) - MTTF_P$
       $(\mathcal{T}_{\min}, \mathcal{C}_{\min}, \mathcal{F}_{\min})$;
5      if $i = 1$ then
6        $\Delta MTTF_P^{\max} = \Delta MTTF_P(\tau_i)$, $\iota = 1$;
7      if $\Delta MTTF_P(\tau_i) > \Delta MTTF_P^{\max}$ then
8        $\Delta MTTF_P^{\max} = \Delta MTTF_P(\tau_i)$, $\iota = i$;
9    if Task deadline constraint holds for $\forall \tau_j \in \mathcal{T}_{\max} + \tau_\iota$ then
10     $\mathcal{T}_{\min} = \mathcal{T}_{\min} - \tau_\iota$, $\mathcal{T}_{\max} = \mathcal{T}_{\max} + \tau_\iota$, $\mathcal{F}_{\min} = \mathcal{F}_{\min} - f_\iota$,
       $\mathcal{F}_{\max} = \mathcal{F}_{\max} + f_\iota$;
11     calculate $MTTF(\mathcal{C}_{\min})$ and $MTTF(\mathcal{C}_{\max})$ by Eq. (21);
12   $\mathcal{T}_{\text{temp}} = \mathcal{T}_{\text{temp}} - \tau_\iota$;
13   if $\mathcal{T}_{\text{temp}} = \emptyset$ then
14     break;
```

Algorithm 6. $LMF\_Replicate(\mathcal{T}, \mathcal{C}, \mathcal{F})$

```
1  $\mathcal{T}_{\text{temp}} = \mathcal{T}_{\min}$;
2  while $MTTF(\mathcal{C}_{\min}) < MTTF(\mathcal{C}_{\max})$ do
3    for $i = 1$ to $size(\mathcal{T}_{\text{temp}})$ do
4      calculate $R_i$ and $R_i^{\text{rep}}$ using Eqs. (2) and (3), respectively;
5      $\Delta MTTF_T(\tau_i) = MTTF_T(\mathcal{T}_{\min}, \mathcal{C}_{\min}, \mathcal{F}_{\min})_{|R(\tau_i)=R_i^{\text{rep}}} - MTTF_T$
       $(\mathcal{T}_{\min}, \mathcal{C}_{\min}, \mathcal{F}_{\min})_{|R(\tau_i)=R_i}$;
6      if $i = 1$ then
7        $\Delta MTTF_T^{\max} = \Delta MTTF_T(\tau_i)$, $\iota = 1$;
8      if $\Delta MTTF_T(\tau_i) > \Delta MTTF_T^{\max}$ then
9        $\Delta MTTF_T^{\max} = \Delta MTTF_T(\tau_i)$, $\iota = i$;
10   if Task deadline constraint holds for $\forall \tau_j \in \mathcal{T}_{\max} + \tau_\iota$ then
11     $R_\iota = R_\iota^{\text{rep}}$, $\mathcal{T}_{\max} = \mathcal{T}_{\max} + \tau_\iota$, $\mathcal{F}_{\max} = \mathcal{F}_{\max} + f_\iota$;
12     calculate $MTTF(\mathcal{C}_{\min})$ and $MTTF(\mathcal{C}_{\max})$ using Eq. (21);
13   $\mathcal{T}_{\text{temp}} = \mathcal{T}_{\text{temp}} - \tau_\iota$;
14   if $\mathcal{T}_{\text{temp}} = \emptyset$ then
15     break;
```

7 Evaluation

We validate the efficacy of HAOF through two sets of experiments. The first set of experiments is implemented on a real-world hardware platform using benchmark tasks while the second set of experiments is conducted in a simulation environment using synthetic tasks. In the two sets of experiments, we use the MTTF improvements to demonstrate the effectiveness of HAOF in increasing availability. This is because that maximizing system availability is equivalent to maximizing system MTTF in the presence of both transient and permanent faults, as discussed in Section 4.1.

In the first set of experiments, we use a heterogenous multicore, Nvidia's Jetson Tegra X2 (TX2) board [43], as the hardware platform. The TX2 board belongs to the so-called performance-heterogeneous MPSoC where cores have the same functionality (i.e., same instruction set architecture (ISA)) but different power-performance characteristics. Thus the MPSoC of the TX2 board actually can meet the homogeneity requirement considered in our framework as tasks do have the same number of execution cycles on both types of cores. The major difference between a strictly homogeneous MPSoC and a performance-heterogeneous MPSoC is the power-performance characteristics. For example, increasing the supply voltage by the same amount on different cores results in the same rise of power consumption, but results in different power consumption changes for performance-heterogeneous cores.

In HAOF, core power consumption impacts $MTTF_P$ since the LTR modeling tool [26] needs the temperature profiles of cores, which are calculated by HotSpot [44] based on the core power consumption. The power consumption of a core is calculated as a function of core frequency and utilization. Thus, the calculation of power consumptions of cores on the board are independent and the heterogeneous power-performance characteristics of TX2 has no impact on the deployment of HAOF in the experiments. Note that we do not use the GPU but only use the CPUs of the board in the experiment. As discussed above, we can conclude that our validation of HAOF on TX2 is sound. To further validate the performance of HAOF, we also use simulated homogeneous MPSoCs in the second set of experiments. The details of the two sets of experiments are presented in Sections 7.1 and 7.2, respectively.

7.1 Experiments on the Tegra Chip

This section first describes the hardware, benchmark tasks, parameter settings, and the existing approaches used for comparison in our experiments on the Tegra chip, then analyzes the experimental results in detail.

7.1.1 *Experimental Setup*

We select the Nvidia's Jetson TX2 board [43] that targets at edge computing such as robotics and medical devices as the hardware platform. This TX2 board contains two Denver [45] cores and four ARM Cortex-A57 cores. All these cores can be active simultaneously, but all Denver (ARM) cores must work at the same voltage and frequency. Except for the primary ARM core, core 0, all cores can be powered on and off dynamically. TX2 is shipped with an operating system based on Ubuntu 16.04 LTS and is capable of executing some widely used benchmarks. However, the operating system must be executed at core 0. We also implement our framework and execute it on core 0. Since all Denver (ARM) cores must work at the same voltage and frequency, we only activate one Denver core and one ARM core to execute tasks and power off other cores, and the activated Denver core and ARM core can run at their own core frequencies. As a low-power chip, both Denver cores and ARM cores support multiple frequencies. In our experiments, we set the regular core frequency to 1.11 GHz and the core frequency for full-speed execution is 2.04 GHz. Thermal sensors are deployed to sample temperatures of the CPU, GPU, and other components. Since the default interface only provides one CPU temperature for all CPU cores, we assume Denver cores and ARM cores have the same temperature.

For the operating system configuration, we use the "userspace" governor mode instead of the "tegraquiet" governor mode provided by Nvidia to dynamically power on and off cores and scale core frequencies based on their

TABLE 2
Execution Time (ms) of Ten Benchmark Tasks

Benchmark Task	Execution Time on ARM Core	Execution Time on Denver Core	Mapping to
crc	75	30	ARM
dijkstra	64	47	Denver
jpeg	33	24	ARM
qsort	69	49	Denver
stringsearch	3	2	Denver
susan	78	52	ARM
blowfish	55	26	Denver
patricia	16	12	Denver
bitcount	210	124	Denver
sha	72	40	ARM

workloads. With this implementation, the operating system does not automatically migrate tasks between cores, scale core frequencies, and power on and off cores. Therefore, dynamic resource management is achieved with our heuristic algorithms and would not be impacted by default strategies in the operating system. Note that we do not inject software errors in TX2 and we do not measure $MTTF_T$ on the board directly. Instead, we use our new analytical method to estimate $MTTF_T$ based on the execution of tasks on the board. This approach to estimating $MTTF_T$ is acceptable since (i) the validity of the expression has been justified in Theorem 1 and (ii) all different resource management methods use the same $MTTF_T$ evaluation formula.

We choose ten representative tasks from MiBench benchmark suites [32] (see Table 2). We measure the execution times of these tasks on TX2's Denver and ARM cores at 2.04 GHz. Based on the execution times, we map the tasks to ARM core or Denver core to balance the workload between cores. For different real-time requirements, we assume tasks' deadlines and periods are in the ranges of $500\ \mathrm{ms}$–$600\ \mathrm{ms}$, $600\ \mathrm{ms}$–$700\ \mathrm{ms}$, $700\ \mathrm{ms}$–$800\ \mathrm{ms}$, and $800\ \mathrm{ms}$–$900\ \mathrm{ms}$. Moreover, to consider the characteristic of task periods, facilitate the implementation of fault tolerance mechanisms, and control the runtime of the proposed scheme and comparative schemes, we jealously choose the periods of tasks under these limited time bounds.

Taking the temperatures (simulated offline by the thermal modeling tool HotSpot [44] and measured online by the thermal sensor on the TX2 board) as input and considering the four failure mechanisms (i.e., EM, TDDB, SM, and TC), the system-level LTR modeling tool [26] is used to derive the MTTF due to permanent faults, $MTTF_P$. During the calculation of $MTTF_P$, default settings are used for the tool [26]. The MTTF due to transient faults $MTTF_T$ and the core MTTF in the presence of permanent and transient faults $MTTF(\mathcal{C}_j)$ are calculated using formulas Eqs. (16) and (21), respectively. For the calculation of $MTTF_T$, we use the parameter values $\lambda_0 = 1.0 \times 10^{-6}$ and $\alpha = 3$ [15].

If a permanent failure is repaired by replacing the faulty chip, $MTTR_P$ could be minutes or hours if all the maintenance management systems are in place [46], [47]. If a chip has some spare cores (e.g., cores not turned on due to the dark-silicon concern [31]), a permanent fault can be repaired by re-mapping tasks from the faulty core to the spare cores. Compared to the time cost of chip replacement, the time cost of task re-mapping would be much lower and hence $MTTR_P$ could be on the order of micro seconds [48]. Transient faults are tolerated by executing the replication of faulty tasks. Thus, $MTTR_T$ is the same as the task execution time which is typically in the order of micro seconds (see Table 2). Clearly, $MTTR_P$ could be several orders of magnitudes larger than $MTTR_T$ in the case of replacing the faulty chip. To include all the above mentioned cases, we empirically set the MTTR ratio $\Upsilon = MTTR_P/MTTR_T$ to 10, 100, 1000, and 10000. Note that we do not focus on developing recovery methods and we only mention these recovery methods for the purpose of quantifying MTTR.

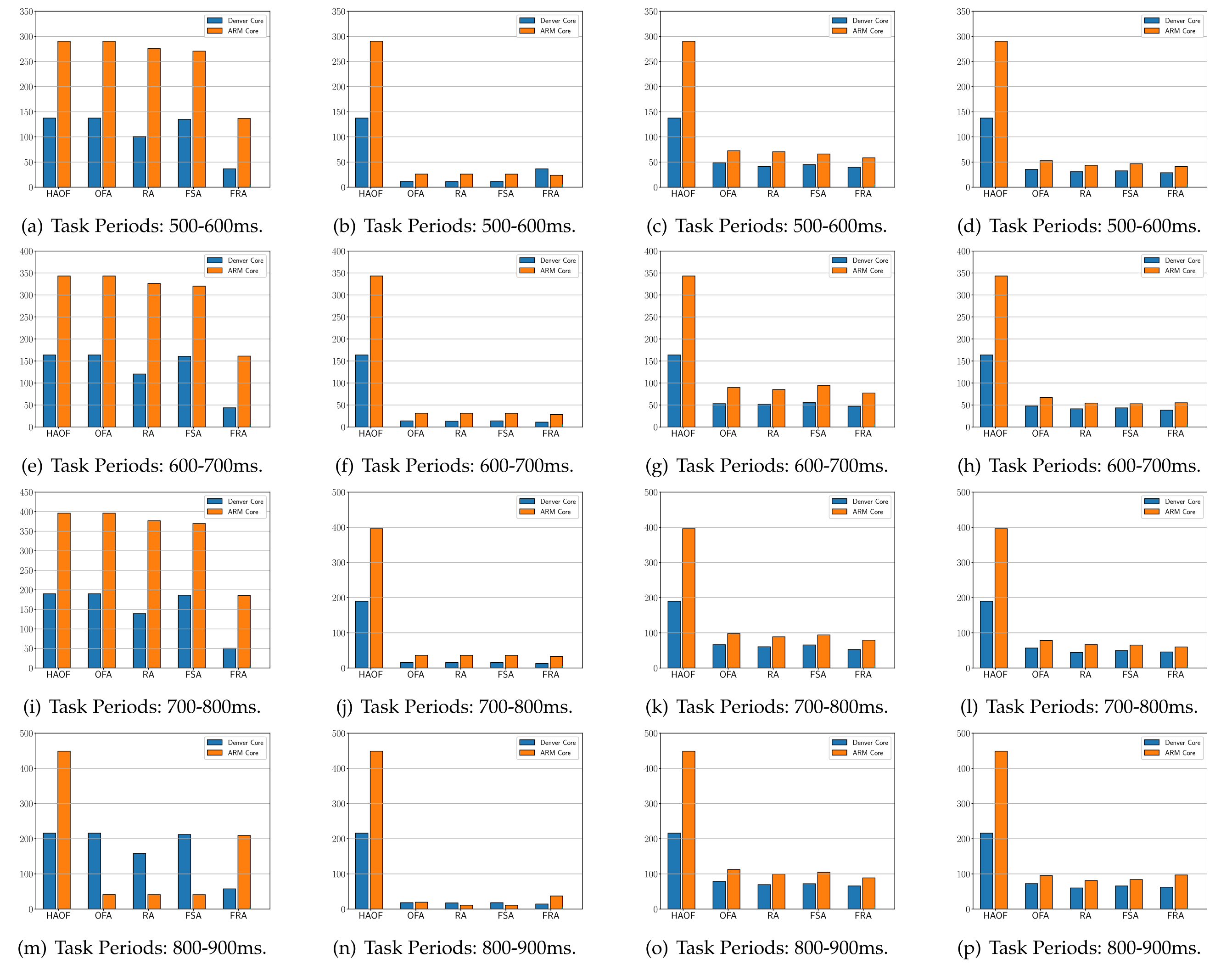
To examine the effectiveness of HAOF in increasing MTTF (and hence availability), we compare HAOF with the following algorithms under a given initial task and frequency assignment [7]: our offline approach (denoted by OFA), Random Algorithm (RA) where replication or speedup is randomly assigned to tasks, Full Speed Algorithm (FSA) where every task is executed at the maximum frequency, and Full Replication Algorithm (FRA) where every task has a recovery task. For fair comparison, the same settings are adopted for all the algorithms. In addition to the comparison, we also measure the runtime overhead of HAOF executing benchmark tasks.

7.1.2 Experimental Results

Fig. 5 shows the MTTF (day) of Denver and ARM cores on the Jetson TX2 board achieved by algorithms HAOF, OFA, RA, FSA, and FRA when executing benchmark tasks with different periods under a given initial task and frequency assignment [7]. In this set of experiments, we randomly set the periods of tasks in the range of $[500\ \mathrm{ms}, 600\ \mathrm{ms}]$, $[600\ \mathrm{ms}, 700\ \mathrm{ms}]$, $[700\ \mathrm{ms}, 800\ \mathrm{ms}]$, or $[800\ \mathrm{ms}, 900\ \mathrm{ms}]$. We set the MTTR ratio Υ to 10, 100, 1000, and 10000. The experimental results clearly show that our approach HAOF always results in larger MTTF improvements of Denver and ARM cores compared to RA, FSA, and FRA.

When the task periods are in the range of $[500\ \mathrm{ms}, 600\ \mathrm{ms}]$, the core MTTFs achieved by HAOF are 4.15 times, 4.54 times, 4.38 times, and 4.27 times higher than that of OFA, RA, FSA, and FRA on average, respectively. When the task periods are in the range of $[600\ \mathrm{ms}, 700\ \mathrm{ms}]$, the core MTTFs achieved by HAOF are 4.04 times, 4.39 times, 4.23 times, and 5.40 times higher than that of OFA, RA, FSA, and FRA on average, respectively. When the task periods are in the range of $[700\ \mathrm{ms}, 800\ \mathrm{ms}]$, the core MTTFs achieved by HAOF are 4.02 times, 4.43 times, 4.25 times, 5.51 times higher than that of OFA, RA, FSA, and FRA on average, respectively. When the task periods are in the range of $[800\ \mathrm{ms}, 900\ \mathrm{ms}]$, the core MTTFs achieved by HAOF are 6.64 times, 9.17 times, 9.01 times, and 5.16 times higher than that of OFA, RA, FSA, and FRA on average, respectively.

As compared to OFA, RA, FSA, and FRA, the higher MTTF of cores achieved by HAOF comes from HAOF's better tradeoff between SER and LTR and hence leads to overall high system reliability. High MTTF of individual cores does not necessarily guarantee high system MTTF since system MTTF is determined by the minimum MTTF of all cores. Unlike OFA, RA, FSA, and FRA, HAOF uses an online approach to balance the MTTF of Denver and ARM cores, thereby achieving a higher system MTTF. Let $MTTF_{\mathrm{sys},1}$, $MTTF_{\mathrm{sys},2}$, $MTTF_{\mathrm{sys},3}$, $MTTF_{\mathrm{sys},4}$, and $MTTF_{\mathrm{sys},5}$



(a) Task Periods: 500-600ms. (b) Task Periods: 500-600ms. (c) Task Periods: 500-600ms. (d) Task Periods: 500-600ms.

(e) Task Periods: 600-700ms. (f) Task Periods: 600-700ms. (g) Task Periods: 600-700ms. (h) Task Periods: 600-700ms.

(i) Task Periods: 700-800ms. (j) Task Periods: 700-800ms. (k) Task Periods: 700-800ms. (l) Task Periods: 700-800ms.

(m) Task Periods: 800-900ms. (n) Task Periods: 800-900ms. (o) Task Periods: 800-900ms. (p) Task Periods: 800-900ms.

Fig. 5. The MTTF (day) of Denver and ARM cores on the Jetson TX2 board when executing benchmark tasks with different periods using schemes HAOF, OFA, RA, FSA, and FRA, where $\Upsilon = 10$ for (a), (e), (i), (m), $\Upsilon = 100$ for (b), (f), (j), (n), $\Upsilon = 1000$ for (c), (g), (k), (o), and $\Upsilon = 10000$ for (d), (h), (l), (p).

denote the system MTTF of Denver and ARM cores achieved by HAOF, OFA, RA, FSA, and FRA, respectively. Then, $\mathcal{I}_{1,2} = \frac{MTTF_{\text{sys},1} - MTTF_{\text{sys},2}}{MTTF_{\text{sys},2}}$, $\mathcal{I}_{1,3} = \frac{MTTF_{\text{sys},1} - MTTF_{\text{sys},3}}{MTTF_{\text{sys},3}}$, $\mathcal{I}_{1,4} = \frac{MTTF_{\text{sys},1} - MTTF_{\text{sys},4}}{MTTF_{\text{sys},4}}$, and $\mathcal{I}_{1,5} = \frac{MTTF_{\text{sys},1} - MTTF_{\text{sys},5}}{MTTF_{\text{sys},5}}$ denote the system MTTF improvement achieved by HAOF over OFA, RA, FSA, and FRA, respectively.

Table 3 lists the average system MTTF improvements ($\times$) achieved by HAOF over OFA, RA, FSA, and FRA. The system MTTF is calculated as the minimum of the MTTFs of Denver and ARM cores. For each task period range, the improvements are averaged over four MTTR ratios (i.e., $\Upsilon = 10, 100, 1000, 10000$). As demonstrated in the table, the improvements are all positive and hence the system MTTF achieved by HAOF is the highest among the five methods, regardless of which task period range and MTTR ratio are adopted. For example, in the case that task periods are in the range of [800 ms, 900 ms], the average system MTTF improvements achieved by HAOF over OFA, RA, FSA, and FRA are about 4.72 times, 6.80 times, 6.73 times, and 5.35 times, respectively.

To investigate the impact of hyper-period length on the time overhead of HAOF, we measure the CPU runtime of HAOF executing benchmark tasks with varying hyper-periods on the Jetson TX2 board. Table 4 summarizes the collected data. In this experiment, the MTTR ratio Υ is set to 10. As can be seen in the table, the runtime overhead of HAOF is negligible (i.e., 0.001s) when the hyper-period length is small (i.e., 1s). However, with the increase of hyper-period length, the runtime overhead of HAOF becomes non-negligible and even unacceptable as compared to the time duration between

TABLE 3
Average System MTTF Improvements ($\times$) Achieved by HAOF over OFA, RA, FSA, and FRA

Benchmarks	$\mathcal{I}_{1,2}$ ($\times$)	$\mathcal{I}_{1,3}$ ($\times$)	$\mathcal{I}_{1,4}$ ($\times$)	$\mathcal{I}_{1,5}$ ($\times$)
Task Periods: 500-600 ms	3.90	4.34	4.05	3.45
Task Periods: 600-700 ms	3.85	4.18	3.90	5.56
Task Periods: 700-800 ms	3.77	4.26	3.91	5.58
Task Periods: 800-900 ms	4.72	6.80	6.73	5.35

TABLE 4
The Runtime Overhead of HAOF Executing Benchmark Tasks with Varying Hyper-Periods

Length of Hyper-period (s)	Runtime Overhead (s)
10000	5.152
1000	1.160
100	0.026
10	0.005
1	0.001

two consecutive invocation of HAOF. For example, when the hyper-period length is 10000s, the runtime overhead of HAOF can be up to 5.152s.

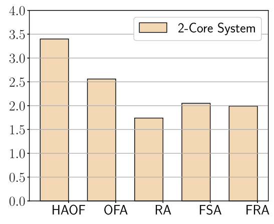
7.2 Simulation-Based Experiments

Readers can observe the restrictions of conducting experiments on the TX2 board that (i) all Denver (ARM) cores must work at the same voltage and frequency, thus we can only select two cores (one Denver core and one ARM core) to let them run at their own frequencies, and (ii) the default interface of the board only provides one CPU temperature for all CPU cores, thus we need to assume Denver and ARM cores have the same temperature. To fully validate our framework, we also conduct extensive simulations that remove the aforementioned restrictions. The setups and results of simulation-based experiments are described below.

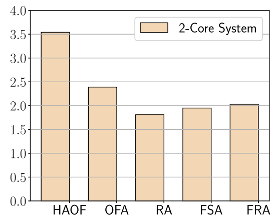
7.2.1 Simulation Setup

The simulations are implemented in C++ and run on a desktop PC equipped with a 3.3 GHz Intel Core i5 CPU and 16 GB RAM. Similar to [49], we perform the simulations using 3 MPSoC systems which consist of 2, 4, and 8 homogeneous cores, respectively. The model of these homogeneous cores is built by extracting the parameters from the ARM Cortex A8 core [50]. That is, the base supply voltage/frequency of each core is assumed to be 1.2V/1.5 GHz. For dynamic voltage/frequency scaling, five different frequency values between 0.8 GHz to 1.5 GHz are used. We develop a task generator that produces synthetic tasks according to the characteristics of benchmarks from the Embedded System Synthesis Benchmark Suite [51]. Three synthetic task sets, represented by $\mathcal{T}_1$, $\mathcal{T}_2$, $\mathcal{T}_3$, are constructed in this way. The three task sets consist of 10, 20, 40 tasks and are executed on the three simulated MPSoC systems, respectively.

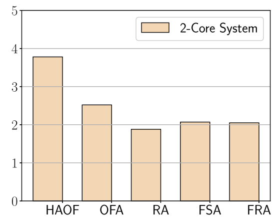
In the simulation based experiments, we obtain the temperature profile using HotSpot [44] and derive the MTTF due to permanent faults, $MTTF_P$, using the LTR modeling tool [26]. HotSpot takes the processor floorplan and power traces as inputs. We adopt the default floorplan of the ARM Cortex A8 core with 1×2, 2×2, and 2×4 core arrangements. The thermal resistance and capacitance of Cortex A8 core are set to $1.83^{\circ}\mathrm{C/W}$ and $112.2\mathrm{mJ}/^{\circ}\mathrm{C}$ [52], respectively. The core power traces are derived using the calculation method proposed in [7]. The method models the power of a core as a function of the core's frequency and utilization, i.e., $Pow(f,U) = Pow_{\mathrm{act}}(f) \times U + Pow_{\mathrm{oth}}(f)$, where $Pow_{\mathrm{act}}(f)$ is the core's active power at frequency f and $Pow_{\mathrm{oth}}(f)$ is the core's power that is independent of core utilization U but related to core frequency f [7]. The efficacy of this power model has been validated in [7]. Simulation results show that the power model leads to only about 1 percent



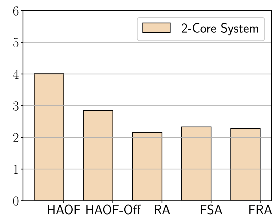
(a) Task Periods: 500-600ms.



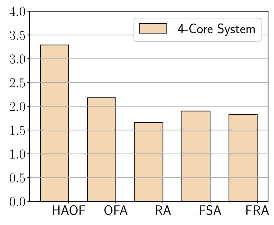
(b) Task Periods: 600-700ms.



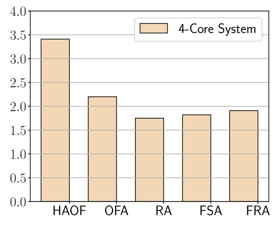
(c) Task Periods: 700-800ms.



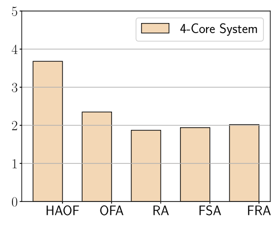
(d) Task Periods: 800-900ms.



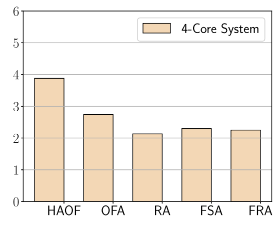
(e) Task Periods: 500-600ms.



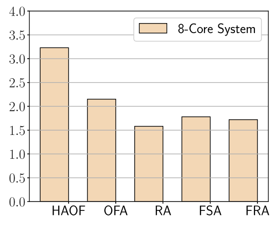
(f) Task Periods: 600-700ms.



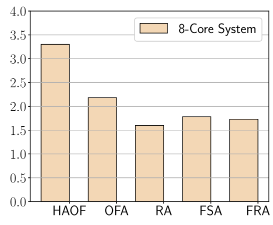
(g) Task Periods: 700-800ms.



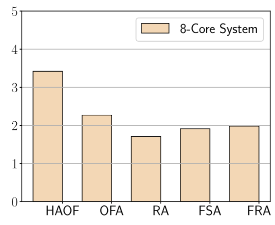
(h) Task Periods: 800-900ms.



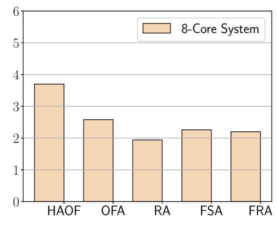
(i) Task Periods: 500-600ms.



(j) Task Periods: 600-700ms.



(k) Task Periods: 700-800ms.



(l) Task Periods: 800-900ms.

Fig. 6. The system MTTF (year) of simulated MPSoCs when executing synthetic tasks with varying periods using schemes HAOF, OFA, RA, FSA, and FRA.

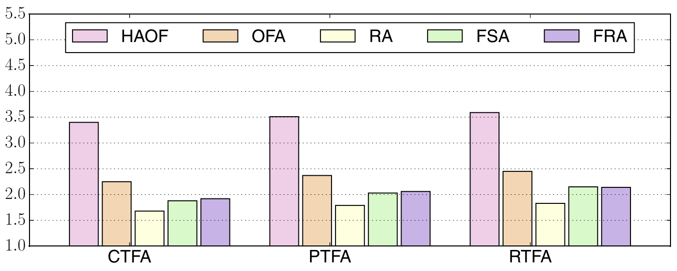


Fig. 7. The MTTF (year) of 4-core system achieved by HAOF, OFA, RA, FSA, and FRA using various initial task and frequency assignments CTFA [54], PTFA [57], and RTFA [22].

average error and less than 2 percent maximum error in $MTTF_P$ [7]. Using the floorplan and power consumption profiles as inputs, HotSpot computes the core temperatures based on the lumped RC thermal model. The RC model is a practical model to estimate the temperature of processors. In this model, the temperature is formulated as a function of core's power consumption, thermal resistance and capacitance, and ambient temperature [53]. The RC model and the tool HotSpot have been widely adopted in literature (e.g., [5], [18], [52], [54], [55], [56]). For more details of the RC model and HotSpot, readers can refer to [44], [53]. Taking the core temperature profiles generated by HotSpot as input and using the default parameters, we obtain the $MTTF_P$ results by the LTR modeling tool [26]. We adopt the parameter values $\lambda_0 = 1.0 \times 10^{-8}$ and $\alpha = 3$ [15] for the calculation of $MTTF_T$ and set $\Upsilon = MTTR_P/MTTR_T$ to 10 for the calculation of core MTTF. The task vulnerability ρ takes the value in the range of (0,1] at random as in [42].

Two simulation based experiments are carried out to validate our framework. First, to verify the effectiveness of HAOF in improving system MTTF (and hence availability), we compare HAOF with our offline approach OFA and peer algorithms RA, FSA, FRA under a given initial task and frequency assignment [7] and a given MTTR ratio (i.e., $\Upsilon = MTTR_P/MTTR_T = 10$). To explore the impact of initial task and frequency assignment on HAOF, we compare HAOF with OFA, RA, FSA, and FRA under different initial assignments: Computing-Performance-Aware Task and Frequency Assignment (CTFA) [54], Power-Aware Task and Frequency Assignment (PTFA) [57], and Reliability-Driven Task and Frequency Assignment (RTFA) [22].

7.2.2 Simulation Results

Fig. 6 shows the MTTF of three simulated MPSoC systems when executing synthetic tasks with varying periods using schemes HAOF, OFA, RA, FSA, and FRA. Clearly, the results indicate that no matter which MPSoC and task period range are adopted, HAOF always outperform OFA, RA, FSA, and FRA in terms of improving system MTTF. For example, in the case of tasks with periods in the range of [600 ms, 700 ms] executing on the 8-core system, the system MTTF achieved by HAOF is 51.4, 106.3, 85.4, and 90.8 percent higher that that of OFA, RA, FSA, and FRA, respectively.

Fig. 7 presents the MTTF of 4-core system achieved by HAOF, OFA, RA, FSA, and FRA when executing tasks with a fixed period range [600 ms, 700 ms] but with different initial task and frequency assignments including CTFA [54], PTFA [57], and RTFA [22]. It is easy to observe from the figure that regardless of which initial task and frequency assignment is adopted, HAOF always achieves the highest system MTTF. For example, in the case of adopting CTFA [57], HAOF outperforms OFA, RA, FSA, and FRA by 51.1, 102.4, 80.9, and 77.1 percent, respectively. The results demonstrate that the efficacy of HAOF in improving system MTTF (and hence availability) is independent of the initial task and frequency assignment. Similar results and the same conclusion can be derived for other cases of task periods and simulated multi-cores, and thus are omitted.

8 Conclusion

This paper addresses the reliability concern of real-time embedded systems considering both permanent and transient faults. We introduce a novel analytical approach to calculate the MTTF due to transient faults, and formulate a max-min problem to optimize the availability of multicore real-time systems. We propose a hybrid framework that consists of an offline stage and an online stage to solve the problem. A task replication and frequency selection strategy is designed for offline use and two task reassignment/replication strategies are developed for online use. These strategies are used to increase the mean time to first failure and hence prolong system availability. We conduct extensive experiments with benchmarks on a hardware platform and synthetic tasks in a simulation environment. The results demonstrate that, with a variety of setups, our framework is effective in improving the MTTF (and hence availability) of multicore systems compared to representative methods.

Acknowledgments

This work was partially supported by National Natural Science Foundation of China (Grant Nos. 61802185 and 61872185), Natural Science Foundation of Jiangsu Province (Grant No. BK20180470), the Fundamental Research Funds for the Central Universities (Grant No. 30919011233), Shanghai Municipal Natural Science Foundation (Grant No. 16ZR1409000), and U.S. NSF (Grant No. CNS-1319904).

References

[1] J. Srinivasan, et al., "The impact of technology scaling on lifetime reliability," in *Proc. Int. Conf. Depend. Syst. Netw.*, 2004, pp. 177–186.
[2] J. Srinivasan, S. V. Adve, P. Bose, and J. A. Rivers, "Exploiting structural duplication for lifetime reliability enhancement," in *Proc. 32nd Int. Symp. Comput. Architecture*, 2005, pp. 520–531.
[3] "Failure mechanisms and models for semiconductor devices," Joint Electron Device Engineering Council, Tech. Rep. JEP 122-B, 2003.
[4] M. Ciappa, et al., "Lifetime prediction and design of reliability tests for high-power devices in automotive applications," *IEEE Trans. Device Mater. Rel.*, vol. 3, no. 4, pp. 191–196, Dec. 2003.
[5] T. Chantem, X. S. Hu, and R. P. Dick, "Temperature-aware scheduling and assignment for hard real-time applications on MPSoCs," *IEEE Trans. VLSI Syst.*, vol. 19, no. 10, pp. 1884–1897, Oct. 2011.
[6] H. Zhang, L. Bauer, M. A. Kochte, E. Schneider, H. Wunderlich, and J. Henkel, "Aging resilience and fault tolerance in runtime reconfigurable architectures," *IEEE Trans. Comput.*, vol. 66, no. 6, pp. 957–970, Jun. 2017.
[7] Y. Ma, T. Chantem, R. P. Dick, and X. S. Hu, "Improving system-level lifetime reliability of multicore soft real-time systems," *IEEE Trans. VLSI Syst.*, vol. 25, no. 6, pp. 1895–1905, Jun. 2017.
[8] T. Wei, P. Mishra, K. Wu, and H. Liang, "Fixed-priority allocation and scheduling for energy-efficient fault tolerance in hard real-time multiprocessor systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 19, no. 11, pp. 1511–1526, Nov. 2008.

[9] D. Zhu, R. Melhem, and D. Mosse, "The effects of energy management on reliability in real-time embedded systems," in *Proc. IEEE/ACM Int. Conf. Comput. Aided Des.*, 2004, pp. 35–40.

[10] S. Aminzadeh and A. Ejlali, "A comparative study of system-level energy management methods for fault-tolerant hard real-time systems," *IEEE Trans. Comput.*, vol. 60, no. 9, pp. 1288–1299, Sep. 2011.

[11] L. Huang, F. Yuan, and Q. Xu, "On task allocation and scheduling for lifetime extension of platform-based MPSoC designs," *IEEE Trans. Parallel Distrib. Syst.*, vol. 22, no. 12, pp. 2088–2099, Dec. 2011.

[12] H. Amrouch, et al., "Towards interdependencies of aging mechanisms," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Des.*, 2014, pp. 478–485.

[13] L. A. R. Duque, J. M. M. Diaz, and C. Yang, "Improving MPSoC reliability through adapting runtime task schedule based on time-correlated fault behavior," in *Proc. Design Autom. Test Europe Conf. Exhib.*, 2015, pp. 818–823.

[14] T. Chantem, Y. Xiang, X. S. Hu, and R. P. Dick, "Enhancing multicore reliability through wear compensation in online assignment and scheduling," in *Proc. Des. Autom. Test Europe Conf. Exhibit.*, 2013, pp. 1373–1378.

[15] B. Zhao, H. Aydin, and D. Zhu, "On maximizing reliability of real-time embedded applications under hard energy constraint," *IEEE Trans. Industrial Informat.*, vol. 6, no. 3, pp. 316–328, Aug. 2010.

[16] L. Rozo and C. Yang, "Improving MPSoC reliability through adapting runtime application schedule based on time-correlated fault behavior," in *Proc. Des. Autom. Test Europe Conf. Exhibit.*, 2015, pp. 818–823.

[17] 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.

[18] J. Zhou and T. Wei, "Stochastic thermal-aware real-time task scheduling with considerations of soft errors," *J. Syst. Softw.*, vol. 102, pp. 123–133, 2015.

[19] P. Axer, M. Sebastian, and R. Ernst, "Reliability analysis for MPSoCs with mixed-critical, hard real-time constraints," in *Proc. 9th IEEE/ACM/IFIP Int. Conf. Hardware/Softw. Codes. Syst. Synthesis*, 2011, pp. 149–158.

[20] C. Chou and R. Marculescu, "FARM fault-aware resource management in NoC," in *Proc. Des. Autom. Test Europe Conf.*, 2011, pp. 1–6.

[21] J. Huang, J. O. Blech, A. Raabe, C. Buckl, and A. Knoll, "Analysis and optimization of fault-tolerant task scheduling on multiprocessor embedded systems," in *Proc. 9th IEEE/ACM/IFIP Int. Conf. Hardware/Softw. Codes. Syst. Synthesis*, 2011, pp. 247–256.

[22] A. Das, A. Kumar, B. Veeravalli, C. Bolchini, and A. Miele, "Combined DVFS and mapping exploration for lifetime and soft-error susceptibility improvement in MPSoCs," in *Proc. Des. Autom. Test Europe Conf. Exhibit.*, 2014, pp. 1–6.

[23] T. Kim, Z. Sun, H. Chen, H. Wang, and S. X. D. Tan, "Energy and lifetime optimizations for dark silicon many core microprocessor considering both hard and soft errors," *IEEE Trans. Very Large Scale Integration Syst.*, vol. 25, no. 9, pp. 2561–2574, Sep. 2017.

[24] Y. Ma, T. Chantem, R. P. Dick, S. Wang, and X. S. Hu, "An on-line framework for improving reliability of real-time systems on Big-Little type MPSoCs," in *Proc. Des. Autom. Test Europe Conf. Exhibit.*, 2017, pp. 446–451.

[25] H. Aliee, M. Glaß, F. Reimann, and J. Teich, "Automatic success tree-based reliability analysis for the consideration of transient and permanent faults," in *Proc. Des. Autom. Test Europe Conf. Exhibit.*, 2013, pp. 1621–1626.

[26] Y. Xiang, T. Chantem, R. P. Dick, X. S. Hu, and S. Li, "System-level reliability modeling for MPSoCs," in *Proc. IEEE/ACM/IFIP Int. Conf. Hardware/Softw. Codes. Syst. Synthesis*, 2010, pp. 297–306.

[27] J. Zhou, X. S. Hu, Y. Ma, and T. Wei, "Balancing lifetime and soft-error reliability to improve system availability," in *Proc. 21st Asia South Pacific Des. Autom. Conf.*, 2016, pp. 685–690.

[28] M. Salehi, et al., "DRVS: Power-efficient reliability management through dynamic redundancy and voltage scaling under variations," in *Proc. IEEE/ACM Int. Symp. Low Power Electron. Des.*, 2015, pp. 225–230.

[29] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," *J. ACM*, vol. 20, no. 1, pp. 46–61, 1973.

[30] H. Pham, *System Software Reliability*, London, U.K.: Springer, 2007.

[31] F. Kriebel, M. Shafique, S. Rehman, S. Garg, and J. Henkel, "Variability and reliability awareness in the age of dark silicon," *IEEE Des. Test*, vol. 33, no. 2, pp. 59–67, Apr. 2016.

[32] Electrical Engineering and Computer Science Department, University of Michigan, Mibench, 2016. [Online]. Available: http://vhosts.eecs.umich.edu/mibench, Accessed on: Jun. 2016

[33] Alpha 21264, "Alpha 21264 microprocessor," 2010. [Online]. Available: https://en.wikipedia.org/wiki/Alpha_21264

[34] Y. Ma, T. Chantem, X. S. Hu, and R. P. Dick, "Improving lifetime of multicore soft real-time systems through global utilization control," in *Proc. 25th Edition Great Lakes Symp. VLSI*, 2015, pp. 79–82.

[35] H. Aliee, "Reliability analysis and optimization of embedded systems using stochastic logic and importance measures," *Friedrich-Alexander-Universitt Erlangen-Nrnberg*, 2017. [Online]. Available: https://opus4.kobv.de/opus4-fau/frontdoor/index/index/docId/8718

[36] F. Zhang and A. Burns, "Schedulability analysis for real-time systems with EDF scheduling," *IEEE Trans. Comput.*, vol. 58, no. 9, pp. 1250–1258, Sep. 2009.

[37] N. Guan and W. Yi, "General and efficient response time analysis for EDF scheduling," in *Proc. Des. Autom. Test Europe Conf. Exhibit.*, pp. 1–6, 2014.

[38] J. A. Stankovic, et al., *Deadline Scheduling for Real-Time Systems: 1514 EDF and Related Algorithms*. Berlin, Germany: Springer, 2012.

[39] X. Li, S. V. Adve, P. Bose, and J. A. Rivers, "Online estimation of architectural vulnerability factor for soft errors," in *Proc. Int. Symp. Comput. Archit.*, 2008, pp. 341–352.

[40] V. Sridharan and D. R. Kaeli, "Using hardware vulnerability factors to enhance AVF analysis," in *Proc. 37th Annu. Int. Symp. Comput. Archit.*, 2010, pp. 461–472.

[41] T. Siddiqua and S. Gurumurthi, "Balancing soft error coverage with lifetime reliability in redundantly multithreaded processors," in *Proc. IEEE Int. Symp. Modeling Anal. Simul. Comput. Telecommun. Syst.*, 2009, pp. 1–12.

[42] J. Zhou, X. Zhou, J. Sun, T. Wei, M. Chen, S. Hu, and X. S. Hu, "Resource management for improving soft-error and lifetime reliability of real-time MPSoCs," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, 2019, in press, doi 10.1109/TCAD.2018.2883993.

[43] Nvidia, "Jetson tegra X2," [Online]. Available: https://developer.nvidia.com/embedded/buy/jetson-tx2

[44] HotSpot 5.0. University of Virginia, 2009. [Online]. Available: http://lava.cs.virginia.edu/HotSpot

[45] Nvidia, "Nvidia Jetson TX2 delivers twice the intelligence to the edge," 2017. [Online]. Available: https://devblogs.nvidia.com/parallelforall/jetson-tx2-delivers-twice-intelligence-edge/

[46] KPILibrary, "Mean time to repair," [Online]. Available: http://kpilibrary.com/kpis/mean-time-to-repair-mttr

[47] Reliability Analytics Blog, "Maintainability theory," [Online]. Available: http://www.reliabilityanalytics.com/blog/2011/09/03/maintainability-theory/

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

[49] A. Das, A. Kumar, and B. Veeravalli, "Reliability-driven task mapping for lifetime extension of networks-on-chip based multiprocessor systems," in *Proc. Des. Autom. Test Europe Conf. Exhibit.*, 2013, pp. 689–694.

[50] ARM, "Cortex A8 processor," [Online]. Available: https://developer.arm.com/products/processors/cortex-a/cortex-a8

[51] E3S, "Embedded system synthesis benchmark suite," 2009. [Online]. Available: http://ziyang.eecs.umich.edu/dickrp/e3s/

[52] R. Jayaseelan and T. Mitra, "Temperature aware task sequencing and voltage scaling," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Des.*, 2008, pp. 618–623.

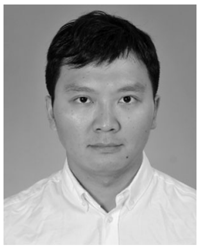
[53] K. Skadron, M. Stan, K. Sankaranarayanan, W. Huang, S. Velusamy, and D. Tarjan, "Temperature-aware microarchitecture: Modeling and implementation," *ACM Trans. Archit. Code Optimization*, vol. 1, no. 1, pp. 94–125, 2004.

[54] S. Sha, W. Wen, M. Fan, S. Ren, and G. Quan, "Performance maximization via frequency oscillation on temperature constrained multi-core processors," in *Proc. 45th Int. Conf. Parallel Process.*, 2016, pp. 526–535.

[55] S. Wang and J. Chen, "Thermal-aware lifetime reliability in multicore systems," in *Proc. 11th Int. Symp. Quality Electron. Des.*, 2010, pp. 399–405.

[56] J. Zhou, T. Wei, M. Chen, J. Yan, and X. Hu, "Thermal-aware task scheduling for energy minimization in heterogeneous real-time MPSoC systems," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 35, no. 8, pp. 1269–1282, Aug. 2016.

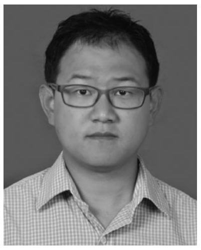
[57] Y. Ge and Q. Qiu, "Task allocation for minimum system power in a homogenous multicore processor," in *Proc. Int. Conf. Green Comput.*, 2010, pp. 299–306.



Junlong Zhou (S'15-M'17) received the PhD degree in computer science from East China Normal University, Shanghai, China, in 2017. He was a visiting scholar with the University of Notre Dame, Notre Dame, IN, during 2014-2015. He is currently an assistant professor with the Nanjing University of Science and Technology, Nanjing, China. His research interests include embedded systems and cyber-physical systems. He is a member of the IEEE.

Xiaobo Sharon Hu (S'85-M'89-SM'02-F'16) received the BS degree from Tianjin University, Tianjin, China, the MS degree from the Polytechnic Institute of New York, Brooklyn, NY, and the PhD degree from Purdue University, West Lafayette, IN. She is currently a professor with the Department of Computer Science and Engineering, University of Notre Dame, Notre Dame, IN. She has authored or co-authored more than 300 papers. Her current research interests include real-time embedded systems, low-power system design, and computing with emerging technologies. She is a fellow of the IEEE.

Yue Ma (S'16) received the bachelor's degree from the Chengdu University of Technology, Chengdu, China, in 2010, and the master's degree from the University of Electronic Science and Technology of China, Chengdu, China, in 2013. He is currently working toward the PhD degree with the University of Notre Dame, Notre Dame, IN. His current research interests include the areas of high performance computing, and multiprocessor systems-on-chip. He is a student member of the IEEE.



Jin Sun (M'17) received the BS and MS degrees in computer science from the Nanjing University of Science and Technology, Nanjing, China, in 2004 and 2006, respectively, and the PhD degree in electrical and computer engineering from the University of Arizona, in 2011. He is currently an associate professor with the Nanjing University of Science and Technology, Nanjing, China. His research interests include integrated circuit modeling and analysis and computer-aided design. He is a member of the IEEE.

Tongquan Wei (M'11-SM'19) received the PhD degree in electrical engineering from Michigan Technological University, in 2009. He is currently an associate professor with the Department of Computer Science and Technology, East China Normal University. His research interests include the areas of internet of things, edge computing, and design automation of intelligent systems and cyber physical systems (CPS). He is a senior member of the IEEE.

Shiyan Hu (SM'10) received the PhD degree in computer engineering from Texas A&M University, in 2008. He is an associate professor at Michigan Technological University. He has been a visiting professor at IBM Research (Austin) in 2010, and a visiting professor at Stanford University from 2015 to 2016. His research interests include Cyber-Physical Systems, Computer-Aided Design of VLSI Circuits, and Embedded Systems. He is a fellow of the IET. He is a senior member of the IEEE.