Balancing Lifetime and Soft-Error Reliability to Improve System Availability

Junlong Zhou[†‡], X. Sharon Hu[‡], Yue Ma[‡], Tongquan Wei[†§]
[†]Department of CST, East China Normal University, Shanghai 200241, China
[‡]Department of CSE, University of Notre Dame, Notre Dame, IN 46656, USA
Email: {jzhou4,shu,yma1}@nd.edu, tqwei@cs.ecnu.edu.cn

***Abstract*—CMOS scaling has greatly increased concerns for 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 concern of the user is to maximize system availability by improving the mean time to failure (MTTF), regardless of whether the failure is caused by permanent faults or transient faults. Addressing this concern requires a uniform metric to measure the effect due to both types of faults. In this paper, we derive 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 formulate and solve the problem of maximizing system availability with consideration of permanent faults, transient faults, and throughput constraint. Extensive simulations of synthetic task sets and benchmarks based on real-world applications were performed to validate our algorithm.**

I. Introduction

Relentless scaling of transistors and the consequent increase in the number of transistors on a chip have greatly improved processor performance during the past three decades. However, the ensuring increase in power density leads to elevated operating temperature and frequent temperature variations, which accelerates 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 processor lifetime. Furthermore, decreased feature size and operating voltage make the circuit 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 and soft-error reliability becomes a major design concern in current computer systems.

Considerable research efforts have been devoted to investigating permanent faults in the past decade [5]–[8]. Amrouch et al. studied the impact of individual aging mechanisms (e.g., EM, TDDB, SM, TC) on the probability of failures as well as the interdependencies of these mechanisms [5]. The authors of [6] established an analytical model to estimate the lifetime reliability of multiprocessor systems and designed a simulated annealing based method to maximize system lifetime. Chantem et al. presented an online reliability-aware task allocation and scheduling algorithm which slows down core wear-out speed to maximize system lifetime [7]. In [8], the authors developed a reliability model that considers the variation of fault behaviors at runtime, and proposed an adaptive reliability-driven scheduling approach which allocates critical and vulnerable tasks to reliable cores. However, none of the above work considers transient faults. On the other hand, a lot of studies have focused on improving soft-error reliability, e.g., [9]–[11]. They utilize fault tolerance techniques based on either software/hardware recovery or time/space redundancy. However, these approaches do not deal with permanent faults.

A few recent papers have focused on handling permanent and transient faults simultaneously [12]–[14]. An efficient fault-aware resource management method is proposed in [12] for network-on-chip systems. The authors proposed placing spare cores to improve system reliability in the presence of permanent and transient faults. Huang et al. [13] developed a software/hardware recovery based scheduling algorithm to tolerate both permanent and transient faults. A genetic algorithm is proposed in [14] to jointly improve system lifetime and soft error susceptibility by determining the mapping and frequency of each task. All the above works treat lifetime and soft-error reliability both as optimization objectives, and utilize separate metrics to perform reliability evaluation. Specifically, mean time to failure is used to measure lifetime reliability while the probability of successful execution is used to estimate soft-error reliability. However, evaluating lifetime and soft-error reliability with different metrics presents two dilemmas.

First, for system users, the concern is the mean time to first failure (MTTF), regardless of whether the failure is caused by 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 scheduling) may increase lifetime reliability but decrease soft-error reliability or vice versa. Without a uniform metric, it is difficult to gauge how tradeoffs should be made to achieve overall high system reliability. In this paper, we propose to use MTTF to evaluate lifetime reliability and soft-error reliability, and make the following main contributions.

We first present an analytical approach to calculate the MTTF due to transient fault for a processor executing a given workload, and show that the MTTF expression indeed correctly represents MTTF. We then formulate a single-objective optimization problem to maximize the system availability for a system that must satisfy a given throughput constraint and suffer from both permanent and transient faults. Finally, we design a framework to solve the optimization problem. The framework builds on a heuristic algorithm that utilizes selective replication and frequency selection.

This work was supported in part by National Natural Science Foundation of China under the grant 91418203, ECNU Outstanding Doctoral Dissertation Cultivation Plan of Action under the grant PY2015047, and U.S. NSF under award CNS-1319904. § Corresponding author.

978-1-4673-9569-4/16/$31.00 ©2016 IEEE

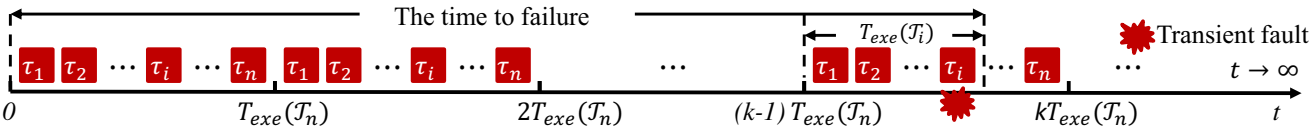


Fig. 1. The time to failure due to a transient fault of task τ_i in the kth run of $\mathcal{T}_n$.

II. Preliminary

A. Workload and Processor Model

The workload considered in this paper consists of n independent tasks, denoted by $\mathcal{T}_n = \{\tau_1, \tau_2, \cdots, \tau_i, \cdots, \tau_n\}$. Tasks in set $\mathcal{T}_n$ are executed consecutively and the entire task set is executed repeatedly while satisfying a given throughput constraint. Such workloads can be found in many applications such as streaming data processing and specified services implemented on severs. The workload is executed on a dynamic voltage/frequency scaling (DVFS) enabled uniprocessor system that supports a discrete set of frequencies varying from the minimum frequency f_{min} to the maximum frequency f_{max}, and all frequency values are normalized with respect to f_{max} (i.e., $f_{max} = 1.0$). Let f_i ($f_{min} \leq f_i \leq f_{max}$)[1] denote the operating frequency of task τ_i ($1 \leq i \leq n$), and c_i denote the execution time of task τ_i measured at the maximum frequency f_{max}. Then the execution time t_i of task τ_i at frequency f_i is given by c_i/f_i and is assumed to be constant from run to run.

Note that the actual execution time of a task may vary from one run to another, and the processor may have some idle time following task executions. Our work can be extended to consider task execution time of each run as a probabilistic value under a specific distribution, and handle idle time by simply treating it as a task without fault occurrence. We leave the study of these extensions to our future work.

B. Transient Fault and Recovery Model

Transient faults are in general modeled using an exponential distribution with an average arrival rate λ, which represents the expected number of failures that occurring per second [9]. It has been shown in [9] that the average rate λ highly depends on the processor frequency, and can be modeled as

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

where λ_0 is the average fault rate at f_{max}, and d (> 0) is a hardware specific constant that indicates the sensitivity of fault rates to frequency scaling. It is clear from (1) that reducing frequency leads to exponentially increasing fault rates.

The soft-error reliability of a task is defined as the probability of its successful execution without the occurrence of transient faults, and can be determined by the exponential failure law. Specifically, given the fault arrival rate $\lambda(f_i)$, the reliability of task τ_i running at frequency f_i is expressed as

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

Redundancy techniques are widely used in improving system reliability due to transient faults. In this paper we consider systems that use replication to tolerate up to one transient fault since single-fault-tolerance is a common assumption. Given task τ_i executing at frequency f_i with a recovery task working at the same frequency, the reliability due to transient fault is

$$R_i^{rep} = 1 - (1 - e^{-\lambda(f_i)\frac{c_i}{f_i}})^2. \tag{3}$$

C. Lifetime Related Failure Mechanisms

We focus on four main IC-dominant failure mechanisms: EM, TDDB, SM, and TC. Other failure mechanisms such as negative/positive bias temperature instability can be readily included as long as their MTTF can be modeled properly.

EM refers to dislocation of metal atoms caused by momentum imparted by electrical current in wires and vias [1]. TDDB refers to deterioration of the gate oxide layer. Gate current due to hot electrons causes defects in the oxide, which eventually form a low-impedance path and cause the transistor to permanently fail. This effect is strongly influenced by temperature, and increases with the reduction of gate oxide dielectric thickness and non-ideal supply voltage reduction [2]. SM is caused by the directionally biased motion of atoms in metal wires due to mechanical stress caused by thermal mismatch among metal and dielectric materials [3]. MTTF of these three failure mechanisms are exponential functions of temperature. TC refers to wear due to thermal stress induced by mismatched coefficients of thermal expansion for adjacent material layers; run-time temperature variation results in inelastic deformation, eventually leading to failure [4]. The MTTF due to TC can be readily calculated as the number of cycles multiplied by the period of the cycles [15].

In this paper, we leverage the system-level MTTF modeling tool [16] to estimate lifetime reliability when considering the above four failure mechanisms. The tool integrates three levels of models, i.e., device-, component-, and system-level models. At the device level, the reliability models due to the above four mechanisms are established. At the component level, each failure mechanism is assumed to obey a specific distribution. Based on the device-level reliability models and assumed distributions, component-level MTTF is calculated. Then using the component-level reliability as input, the system-level MTTF is obtained by Monte Carlo simulation.

III. MTTF due to Transient Fault

Obtaining MTTF due to transient fault enables the designers to use a uniform metric to evaluate lifetime and soft-error reliability, which allows to make decisions for achieving overall high system reliability. In this section, we first introduce an analytical formulation to calculate MTTF due to transient fault, then prove some properties of the MTTF expression and show the expression indeed correctly represents MTTF.

We denote MTTF due to transient fault by $MTTF_T$. As specified in Section II-A, our task model assumes the given workload repeatedly runs forever. Fig. 1 illustrates the time

[1] f_i and f_j represent the operating frequency of task τ_i and τ_j, respectively, and they can be the same frequency in discrete frequency set.

to failure (TTF) due to a transient fault occurring during the execution of task τ_i in the kth run of task set $\mathcal{T}_n$. Here $T_{exe}(\mathcal{T}_n) = \sum_{i=1}^{n} t_i$ is the total execution time of $\mathcal{T}_n$ and $T_{exe}(\mathcal{T}_i) = \sum_{j=1}^{i} t_j$ is the total execution time of τ_1 to τ_i. As shown in the figure, TTF is equal to $(k-1)T_{exe}(\mathcal{T}_n)+T_{exe}(\mathcal{T}_i)$. Now, let $P_{succ}(\mathcal{T}_{n,k-1})$ be the probability that the first $k-1$ runs of $\mathcal{T}_n$ are all successful, and $P_{fail}(\tau_i)$ be the probability that τ_i is erroneous but τ_1 to τ_{i-1} in the same run of $\mathcal{T}_n$ are successful. Then $MTTF_T$ can be calculated as

$$MTTF_T = \sum_{k=1}^{\infty} \sum_{i=1}^{n} \{(k-1)T_{exe}(\mathcal{T}_n) + T_{exe}(\mathcal{T}_i)\} \cdot P_{succ}(\mathcal{T}_{n,k-1}) \cdot P_{fail}(\tau_i). \quad (4)$$

Using (4) to compute MTTF directly is challenging due to the infinite summation terms. However, by applying a series of algebraic transformations, we can remove the infinite summation terms and derive a simple expression to calculate MTTF. Below, we show several key steps of the transformation.

Based on definition of $P_{succ}(\mathcal{T}_{n,k-1})$ and $P_{fail}(\tau_i)$, we have

$$\begin{cases} P_{succ}(\mathcal{T}_{n,k-1}) = (\prod_{i=1}^{n} R_i)^{k-1} \\ P_{fail}(\tau_i) = R_1 R_2 \cdots R_{i-1}(1-R_i) \end{cases}, \quad (5)$$

where R_i is the reliability of task τ_i at frequency f_i and can be obtained using (2). Let $P_{fail}(\mathcal{T}_n)$ be the probability that $\mathcal{T}_n$ sees a failure in a run. Then $P_{fail}(\mathcal{T}_n)$ can be expressed as

$$P_{fail}(\mathcal{T}_n) = 1 - \prod_{i=1}^{n} R_i. \quad (6)$$

According to (5) and (6), we can rewrite (4) as

$$MTTF_T = \sum_{k=1}^{\infty} (k-1)(1-P_{fail}(\mathcal{T}_n))^{k-1} \cdot T_{exe}(\mathcal{T}_n) \cdot P_{fail}(\mathcal{T}_n) + \sum_{k=1}^{\infty} (1-P_{fail}(\mathcal{T}_n))^{k-1} \cdot \sum_{i=1}^{n} T_{exe}(\mathcal{T}_i) \cdot P_{fail}(\tau_i), \quad (7)$$

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

$$\begin{cases} \sum_{k=1}^{\infty}(k-1)(1-P_{fail}(\mathcal{T}_n))^{k-1} = \frac{1-P_{fail}(\mathcal{T}_n)}{(P_{fail}(\mathcal{T}_n))^2} \\ \sum_{k=1}^{\infty}(1-P_{fail}(\mathcal{T}_n))^{k-1} = \frac{1}{P_{fail}(\mathcal{T}_n)} \end{cases}. \quad (8)$$

Finally, $MTTF_T$ can be calculated as

$$MTTF_T = \frac{T_{exe}(\mathcal{T}_n) + T_{exp}(\mathcal{T}_n)}{P_{fail}(\mathcal{T}_n)} - T_{exe}(\mathcal{T}_n), \quad (9)$$

where $T_{exp}(\mathcal{T}_n) = \sum_{i=1}^{n} T_{exe}(\mathcal{T}_i) \cdot P_{fail}(\tau_i)$ is the expected time to failure when the fault occurs in the first run. According to (9), given the execution time and reliability of each task, the MTTF due to transient fault can be readily evaluated.

The $MTTF_T$ expression in (9) is derived for a workload $\mathcal{T}_n$ being repeatedly executed forever. One immediate question is what the $MTTF_T$ would be if we treat two or more runs of $\mathcal{T}_n$ as the given workload being repeated forever. Clearly this new $MTTF_T$ should be exactly the same as the one in (9). We introduce a theorem below to show that computing $MTTF_T$ using (9) indeed leads to the desired conclusion.

Theorem 1: *For any given task set $\mathcal{T}_n$ and any integer m (≥ 2), let $\mathcal{T}_n^m$ be the task set containing m runs of $\mathcal{T}_n$, i.e., $\mathcal{T}_n^m = \{\underbrace{\tau_1, \tau_2, \cdots, \tau_n, \tau_1, \tau_2, \cdots, \tau_n, \cdots\cdots, \tau_1, \tau_2, \cdots, \tau_n}_{n \times m}\}$. Then $MTTF_T(\mathcal{T}_n^m) = MTTF_T(\mathcal{T}_n)$ holds.*

Theorem 1 can be proved by a series of inductions. We omit the proof due to page limit. Theorem 1 demonstrates that the $MTTF_T$ calculation given in (9) indeed satisfies the basic property that $MTTF_T$ should be independent on the number of runs of a given workload used to calculate $MTTF_T$.

IV. Framework To Maximize System Availability

We first define the optimization problem that we aim to solve, then analyze several scenarios associated with the problem, and finally present our framework to solve the problem.

As we have pointed out earlier, decisions on how to execute each task (replication or not, and using which frequency) can impact soft-error and lifetime reliability. From the user's point of view, the goal is to maximize system availability, defined as $\mathcal{A} = \frac{MTTF}{MTTF+MTTR}$, where $MTTR$ is the mean time to repair [17]. For a system that may suffer from both transient and permanent faults, maximizing system availability becomes

$$\max:\{\frac{MTTF_T}{MTTF_T + MTTR_T}, \frac{MTTF_P}{MTTF_P + MTTR_P}\}, \quad (10)$$

and (10) can be rewritten as

$$\max:\{\frac{MTTF_T}{MTTR_T}, \frac{MTTF_P}{MTTR_P}\}. \quad (11)$$

Since $MTTR_T$ depends on the actual recovery mechanism used and $MTTR_P$ depends on the product contract, we simply treat them as constants in this paper and leave the more detailed study of their values to future work. Therefore, the problem that we aim to solve can be defined as follows.

Maximizing System Availability: *Given task set $\mathcal{T}_n = \{\tau_1, \tau_2, \cdots, \tau_n\}$ to be executed repeatedly forever and initial task operating frequency set $\mathcal{F}_n = \{f_1, f_2, \cdots, f_n\}$ chosen to satisfy the throughput constraint, determine (i) whether any tasks should be replicated and (ii) the frequency that each task should be executed at, in order to maximize $\{\Upsilon MTTF_T, MTTF_P\}$[2], where $\Upsilon = \frac{MTTR_P}{MTTR_T}$ is a given constant.* Note that $\max\{\Upsilon MTTF_T, MTTF_P\}$ is equivalent to $\max\min\{\Upsilon MTTF_T, MTTF_P\}$.

To solve the above problem, we need to first determine $MTTF_T$ and $MTTF_P$. Using our proposed analytical method to calculate $MTTF_T$ and the system-level lifetime reliability modeling tool [16] to estimate $MTTF_P$, this can be readily achieved. Depending on which reliability dominates, different countermeasures may be activated accordingly to maximize system reliability. We group the relationship between $\Upsilon MTTF_T$ and $MTTF_P$ into four scenarios when running task set $\mathcal{T}_n$ under various core wear states, temperature profiles, and frequency setups: 1) $\Upsilon MTTF_T \ll MTTF_P$, 2) $\Upsilon MTTF_T < MTTF_P$, 3) $\Upsilon MTTF_T > MTTF_P$, 4) $\Upsilon MTTF_T \gg MTTF_P$.

Before discussing how to handle each of the four scenarios, we first show that all four scenarios indeed exist. We have carried out several simulations to evaluate $\Upsilon MTTF_T$ and $MTTF_P$ for different benchmark programs, core wear states, temperature profiles, and frequencies. The same core, benchmarks and parameter settings as in [18] are utilized

[2] $\max\{\Upsilon MTTF_T, MTTF_P\} = \max\{\frac{MTTR_P \cdot MTTF_T}{MTTR_T}, MTTF_P\}$ is the same goal as $\max\{\frac{MTTF_T}{MTTR_T}, \frac{MTTF_P}{MTTR_P}\}$ since $MTTR_P$ is a constant.

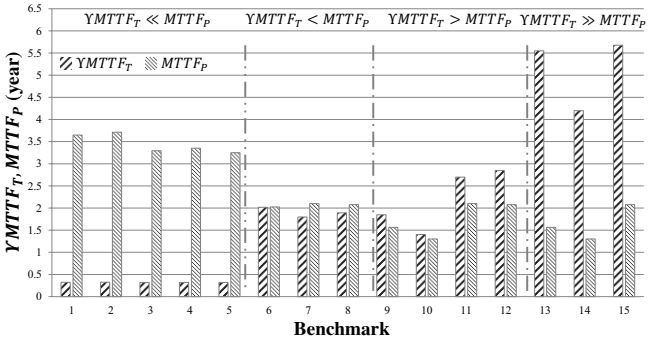


Fig. 2. $\Upsilon MTTF_T$ and $MTTF_P$ for 15 different benchmarks.

in the simulations[3]. The results of $\Upsilon MTTF_T$ and $MTTF_P$ are summarized in Fig. 2, which clearly show that all four scenarios exist depending on the actual benchmark.

We propose to solve the **Maximizing System Availability** problem by using the framework shown in Fig. 3. The framework operates as follows. It first takes the workload and processor specification to determine $MTTF_T$ and $MTTF_P$. Here it assumes that tasks are executed at their respective initial frequencies determined based on the throughput constraint and none of the tasks are replicated. It then compares the value of $\Upsilon MTTF_T$ and $MTTF_P$. If $MTTF_P$ is far greater than $\Upsilon MTTF_T$ (scenario 1), the system availability is dominated by $\Upsilon MTTF_T$ and *Full Replication and Speedup* can be safely used to increase $MTTF_T$. *Full Replication and Speedup* refers to the strategy that each original task has a recovery task, and both the original task and recovery task are executed at the maximum frequency. For this scenario, other existing techniques such as the ones in [9]–[11] can be used in order to increase $MTTF_T$. On the other hand, if $\Upsilon MTTF_T$ is far greater than $MTTF_P$ (scenario 4), *Lifetime Reliability-Aware Strategy* such as the ones in [5]–[8] can be used to maximize $MTTF_P$ and hence improve system availability.

If $MTTF_P$ is greater than $\Upsilon MTTF_T$ but not by too much (scenario 2), when maximizing system availability, the negative effect on lifetime reliability due to executing an increased workload (because of task replication) at the maximum frequency should be taken into account. We adopt *Partial Replication and Speedup* to select a subset of tasks to have recovery tasks and execute at the maximum frequency. The reason why we use f_{max} is that it achieves a lowest transient fault rate and avoid the trade-off between replication and frequency selection. On the other hand, if $\Upsilon MTTF_T$ is greater than $MTTF_P$ (scenario 3), *DVS-based Strategy* that reduces the operating temperature by scaling task frequency can be applied to increase $MTTF_P$ and hence improve system availability. However, frequency scaling may lead to violation of the throughput requirement, thus a trade-off between system availability and throughput is needed for this scenario.

Among the four strategies, *Full Replication and Speedup* is simple and easy to implement, *DVS-based Strategy* and *Lifetime Reliability-Aware Strategy* have been explored in the previous literature. Therefore, we focus on the design of the *Partial Replication and Speedup* strategy in this paper.

[3]The simulation results here are not meant to be comprehensive but just to demonstrate that all four scenarios identified can occur. In the simulation, the value of $MTTF_T$ is calculated using (9), the value of $MTTF_P$ is derived using the lifetime reliability modeling tool [16], and the Υ is set to 1.

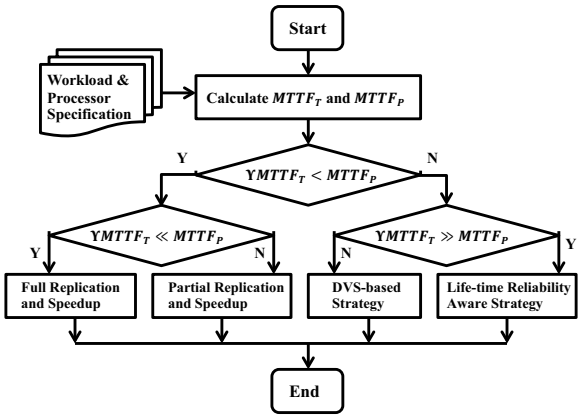


Fig. 3. Proposed framework to maximize system availability.

V. Partial Replication and Speedup

Replication and speed selection techniques can be used to maximize system availability for the scenario $\Upsilon MTTF_T < MTTF_P$ since it improves the system soft-error reliability due to transient faults, and simultaneously limits the negative effect on hardware aging due to the execution of an increased number of tasks. The authors in [14] and [19] have investigated the impact of selective replication and partial speedup on soft-error reliability and lifetime reliability, and observed that the technique is effective in balancing the two reliabilities. However, neither papers focus on designing 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 system availability for the scenario $\Upsilon MTTF_T < MTTF_P$.

A. Replication vs. Speedup for A Single Task

Since system availability is the minimum of $\Upsilon MTTF_T$ and $MTTF_P$, the main idea of PRS is to maximize $MTTF_T$ when $\Upsilon MTTF_T$ is below $MTTF_P$. This is achieved by iteratively making the best choice (replication or speedup) for each task in terms of improving soft-error reliability. In other words, the key is to determine the selection of which task to have a recovery task or to execute at the maximum frequency, i.e., determine RS_i where RS_i is defined by

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

As described in Section II-B, replicating a task can tolerate one transient fault and hence improve soft-error reliability, and increasing task operating frequency leads to exponentially decreasing transient fault rate and hence improve soft-error reliability. Thus, according to (1)-(3), the soft-error reliability increment of task τ_i achieved by replication and speedup are given by $\Delta R_i^r = R_i^{rep} - R_i$ 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 to compare the effectiveness of replication with that of speedup in terms

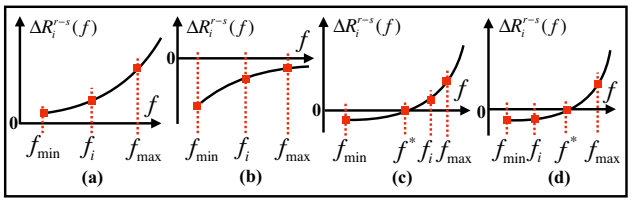


Fig. 4. An illustration of four different cases of ΔR_i^{r-s} values, where $\Delta R_i^{r-s} > 0$ for (a) and (c) and $\Delta R_i^{r-s} < 0$ for (b) and (d).

of improving soft-error reliability, which is formulated as

$$\Delta R_i^{r-s} = \Delta R_i^r - \Delta R_i^s = 2R_i - R_i^2 - R_{i|f_i=f_{max}}, \qquad (12)$$

where $\Delta R_i^{r-s} > 0$ indicates replication is better for τ_i and $\Delta R_i^{r-s} < 0$ indicates speedup is better. From (2), it is clear 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. (We omit the proof due to page limit.)

Theorem 2: *ΔR_i^{r-s} as defined in (12) increases monotonically as f_i increases.*

According to Theorem 2, given a specific frequency f_i for τ_i, ΔR_i^{r-s} must be in one of the cases illustrated in Fig. 4. In Fig. 4, f^* corresponds to the frequency where $\Delta R_i^{r-s}(f^*) = 0$, and is determined by (2) and (12). The significance of Theorem 2 is that for given f_i, f_{min}, f_{max}, and f^*, 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 soft-error reliability. We summarize the basic steps needed to determine RS_i for task τ_i in Alg. 1.

B. PRS Heuristics

Alg. 1 decides whether a task should be replicated or sped up in order to maximize soft-error reliability. However, if every task is replicated or sped up, the life-time reliability may be degraded. We now present a heuristic algorithm 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 algorithm works as follows. It first uses Alg. 1 to determine whether replication or speedup should be used for each task. Based on the replication/speedup choice for each task given by Alg. 1, it iteratively decides whether the replication/speedup choice for a task should actually be adopted. Some tasks may end up neither replicated or sped up because doing so would lead to degraded system MTTF.

The details of our heuristic algorithm is given in Alg. 2. Alg. 2 takes as inputs task set $\mathcal{T}_n$, a set of initial task frequencies $\mathcal{F}_n = \{f_1, f_2, \cdots, f_n\}$, f_{min} and f_{max}, and a given constant Υ (capturing the ratio between $MTTR_P$ and $MTTR_T$). It starts by determining the replication/speedup choice RS_i for each task τ_i using Alg. 1 (lines 1-2). Temporary task set $\mathcal{T}^{temp}$ is used to store candidate tasks when deciding which task's replication and speedup choice should be actually adopted. It keeps a copy of $\mathcal{T}_n$ at first (line 3) and will be updated as each decision is made. The system current workload $\mathcal{T}^{cur}$ is initialized to $\mathcal{T}_n$ (i.e., no replication and initial frequency) (line 4). The algorithm then iteratively compares the value of $\Upsilon MTTF_T$ and $MTTF_P$ for the system current workload. When the system is in the concerned scenario (line 5), the

Algorithm 1: $RS(f_i, f_{min}, f_{max}, f^*)$

1 **if** $\Delta R_i^{r-s}(f_{min}) > 0$ **then**
2 $\quad RS_i = 1$; // $\Delta R_i^{r-s}(f_i) > \Delta R_i^{r-s}(f_{min}) \Rightarrow \Delta R_i^r > \Delta R_i^s$
3 **if** $\Delta R_i^{r-s}(f_{max}) < 0$ **then**
4 $\quad RS_i = 0$; // $\Delta R_i^{r-s}(f_i) < \Delta R_i^{r-s}(f_{max}) \Rightarrow \Delta R_i^r < \Delta R_i^s$
5 **if** $f_i > f^*$ && $R_i^{r-s}(f^*) = 0$ **then**
6 $\quad RS_i = 1$; // $\Delta R_i^{r-s}(f_i) > \Delta R_i^{r-s}(f^*) \Rightarrow \Delta R_i^r > \Delta R_i^s$
7 **if** $f_i < f^*$ && $R_i^{r-s}(f^*) = 0$ **then**
8 $\quad RS_i = 0$; // $\Delta R_i^{r-s}(f_i) < \Delta R_i^{r-s}(f^*) \Rightarrow \Delta R_i^r < \Delta R_i^s$
9 **return** RS_i;

Algorithm 2: $PRS\_Heu(\mathcal{T}_n, \mathcal{F}_n, f_{min}, f_{max}, \Upsilon)$

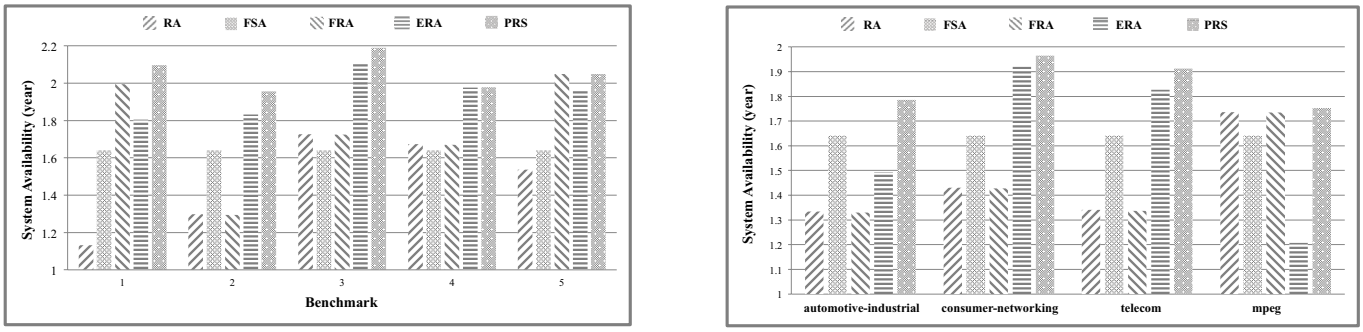
1 **for** $i = 1$ to n **do**
2 $\quad$ determine RS_i by **Algorithm 1**;
3 $\mathcal{T}^{temp} = \mathcal{T}_n$; // a temporary set
4 $\mathcal{T}^{cur} = \mathcal{T}_n$; // $\mathcal{T}_n$: no replication, initial frequency
5 **while** $\Upsilon MTTF_T(\mathcal{T}^{cur}) < MTTF_P(\mathcal{T}^{cur})$ **do**
6 $\quad$ **for** $i = 1$ to $sizeof(\mathcal{T}^{temp})$ **do**
7 $\quad\quad$ **if** $RS_i = 1$ **then**
8 $\quad\quad\quad \Delta\Upsilon MTTF_{T,i} = \Upsilon MTTF_T(\mathcal{T}^{cur} + \tau_i) - \Upsilon MTTF_T(\mathcal{T}^{cur})$;
9 $\quad\quad$ **else**
10 $\quad\quad\quad \Delta\Upsilon MTTF_{T,i} = \Upsilon MTTF_T(\mathcal{T}^{cur} - \tau_i + \tau_{i|f_i=f_{max}}) - \Upsilon MTTF_T(\mathcal{T}^{cur})$;
11 $\quad\quad$ **if** $i = 1$ **then**
12 $\quad\quad\quad \Delta\Upsilon MTTF_T^{max} = \Delta\Upsilon MTTF_{T,i}$;
13 $\quad\quad\quad j = 1$;
14 $\quad\quad$ **if** $\Delta\Upsilon MTTF_{T,i} > \Delta\Upsilon MTTF_T^{max}$ **then**
15 $\quad\quad\quad \Delta\Upsilon MTTF_T^{max} = \Delta\Upsilon MTTF_{T,i}$;
16 $\quad\quad\quad j = i$;
17 $\quad$ **if** $RS_j = 1$ **then**
18 $\quad\quad \mathcal{T}^{cur} = \mathcal{T}^{cur} + \tau_j$;
19 $\quad$ **else**
20 $\quad\quad \mathcal{T}^{cur} = \mathcal{T}^{cur} - \tau_j + \tau_{j|f_j=f_{max}}$;
21 $\quad \mathcal{T}^{temp} = \mathcal{T}^{temp} - \tau_j$;
22 **return** $\min\{\Upsilon MTTF_T(\mathcal{T}^{cur}), MTTF_P(\mathcal{T}^{cur})\}$;

algorithm first computes the increase in system MTTF for each candidate task $\tau_i \in \mathcal{T}^{temp}$ if τ_i is replicated (lines 7-8) or sped up (lines 9-10). The task that could lead to the maximum increase in system MTTF is found (lines 11-16) and allowed to use its replication (lines 17-18) or speedup choice (lines 19-20). The system current workload is hence updated. Then the task is removed from $\mathcal{T}^{temp}$ (line 21). Finally, when the system is no longer in the concerned scenario, the algorithm returns the achieved system availability (line 22).

It is easy to see that Alg. 2 is a greedy algorithm. It might be possible to improve $MTTF_T$ further if an exhaustive search is performed. Since computing $MTTF_P$ can be quite time consuming, we opt to use the simple heuristics. The study of the quality of the heuristics is left to future work.

VI. Evaluation

To evaluate the effectiveness of our approach, we have performed several simulation-based studies. Specifically, we compare the system availability of our approach PRS to the following four representative algorithms: 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; Full replication algorithm (FRA)



(a) (b)

Fig. 5. Results of system availability of 5 synthetic benchmarks (a) and 4 real-world benchmarks (b) using algorithms RA, FSA, FRA, ERA, and PRS.

where every task has a recovery task; Energy-efficient and reliability-aware algorithm (ERA) [20] where each task is executed at an energy-efficient frequency and all tasks share a common recovery task to enhance system reliability. We leave other performance evaluations to future work due to page limit.

We use Alpha 21264 microprocessor as the hardware platform since it is the only publicly available processor with well established power and thermal models [21]. The processor frequency/voltage levels are $1.0\text{GHz}/0.7\text{V}$, $1.25\text{GHz}/0.8\text{V}$, $1.5\text{GHz}/0.9\text{V}$, $1.75\text{GHz}/1.0\text{V}$, and $2.0\text{GHz}/1.1\text{V}$. The temperature profiles are obtained using HotSpot 5.0 [22]. The initial temperature and ambient temperature are set to 330°F and 318.15°F, respectively (the default temperatures of HotSpot). The $MTTF_P$ is obtained using the system-level lifetime reliability modeling tool [16]. For the calculation of $MTTF_T$, the parameters are set as in [23]: $\lambda_0 = 10^{-7}$ and $d = 3$. Finally, in our simulations, we set $\Upsilon = \frac{MTTR_P}{MTTR_T}$ to 1 and leave the investigation of its effect on our framework to future work.

Two sets of simulations have been carried out to validate our approach. In the first set of simulations, five synthetic task sets are randomly generated to verify the proposed algorithms, where each task set consists of 20 tasks. In the second set of simulations, four benchmarks based on real-world applications from the Embedded System Synthesis Benchmark Suite [24] are utilized to validate the proposed approach. The applications are automotive-industrial, consumer-networking, telecom, and mpeg, which consist of 16, 20, 17, 15 tasks, respectively.

The results of system availability of randomly generated task sets and benchmarks based on real-world applications using five different algorithms are shown in Figs. 5(a) and 5(b), respectively. The system availability is calculated by $\min\{\Upsilon MTTF_T, MTTF_P\}$, where $\Upsilon = 1$. The figure has demonstrated that our algorithm PRS outperforms RA, FSA, FRA, and ERA by 36.6%, 19.8%, 24.3%, and 11.5% on average, respectively. Furthermore, the improvements of our algorithm over RA, FSA, FRA, and ERA can be up to 85%, 33.5%, 51%, and 45.1%, respectively.

VII. Conclusion

In this paper, we have introduced a novel analytical approach to calculate the MTTF due to transient fault, and formulated a max-min problem to optimize system availability. We then designed a framework to solve the problem and proposed a task replication and frequency selection strategy. Our algorithm was shown to improve system availability by up to 85% compared with randomly selecting tasks to be replicated or sped up.

References

[1] J. Srinivasan, S. Adve, P. Bose, and J. Rivers, "The impact of technology scaling on lifetime reliability," in *Proc. Int. Conf. Depend. Syst. and Nwk.*, pp.177-186, 2004.

[2] J. Srinivasan, S. Adve, P. Bose, and J. Rivers, "Exploiting structural duplication for lifetime reliability enhancement," in *Proc. Int. Symp. Comput. Archit.*, pp. 520-531, 2005.

[3] "Failure mechanisms and models for semiconductor devices," Joint Electron Device Engineering Council, Tech. Rep., JEP 122-B, 2003.

[4] M. Ciappa, F. Carbognani, and W. Fichtner, "Lifetime prediction and design of reliability tests for high-power devices in automotive applications," *IEEE Trans. Device and Mater. Reliab.*, vol. 3, no. 4, pp. 191-196, 2003.

[5] H. Amrouch, V. Santen, T. Ebi, V. Wenzel, and J. Henkel, "Towards interdependencies of aging mechanisms," in *Proc. Int. Conf. Comput. Aided Design*, pp. 478-485, 2014.

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

[7] T. Chantem, Y. Xiang, X. Hu, and R. Dick, "Enhancing multicore reliability through wear compensation in online assignment and scheduling," in *Proc. Int. Conf. Design, Autom. and Test in Europe*, pp. 1373-1378, 2013.

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

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

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

[11] V. Sridharan and D. Kaeli, "Using hardware vulnerability factors to enhance AVF analysis," in *Proc. Int. Symp. Comput. Archit.*, pp. 461-472, 2010.

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

[13] J. Huang, J. Blech, A. Raabe, C. Buckl, and A. Knoll, "Analysis and optimization of fault-tolerant task scheduling on multiprocessor embedded systems," in *Proc. Int. Conf. Hw/Sw Codesign and Syst. Synthe.*, pp. 247-256, 2011.

[14] 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. Int. Conf. Design, Autom. and Test in Europe*, pp. 1-6, 2014.

[15] A. Coskun, R. Strong, D. Tullsen, and T. Rosing, "Evaluating the impact of job scheduling and power management on processor lifetime for chip multiprocessors," *ACM SIGMETRICS Perform. Eval. Review*, vol. 37, no. 1, pp. 169-180, 2009.

[16] Y. Xiang, T. Chantem, R. Dick, X. Hu, and L. Shang, "System-level reliability modeling for MPSoCs," in *Proc. Int. Conf. Hw/Sw Codesign and Syst. Synth.*, pp. 297-306, 2010.

[17] H. Pham, "System software reliability," *Springer-Verlag London*, 2007.

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

[19] T. Siddiqua and S. Gurumurthi, "Balancing soft error coverage with lifetime reliability in redundantly multithreaded processors," in *Proc. Int. Symp. Modeling, Analy. & Simulation of Comput. and Telecom. Syst.*, pp. 1-12, 2009.

[20] B. Zhao, H. Aydin, and D. Zhu, "Shared recovery for energy efficiency and reliability enhancements in real-time applications with precedence constraints," *ACM Trans. Design Autom. of Electron. Syst.*, vol. 18, no. 2, 2013.

[21] G. Quan and V. Chaturvedi, "Feasibility analysis for temperature-constraint hard real-time periodic tasks," *IEEE Trans. Ind. Inform.*, vol.6, no. 3, pp. 329-339, 2012.

[22] HotSpot 5.0. University of Virginia. Available: http://lava.cs.virginia.edu/HotSpot.

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

[24] E3S. Available: http://ziyang. eecs.umich.edu/ dickrp/e3s/. 2013.