Journal of Systems Architecture 61 (2015) 71–81

Contents lists available at ScienceDirect

Journal of Systems Architecture

journal homepage: www.elsevier.com/locate/sysarc

Energy minimization for reliability-guaranteed real-time applications using DVFS and checkpointing techniques

Zheng Li [a,*], Shangping Ren [a], Gang Quan [b]

[a] Illinois Institute of Technology, Chicago, IL 60616, USA
[b] Florida International University, Miami, FL 33174, USA

ARTICLE INFO

Article history:
Received 5 September 2014
Received in revised form 24 December 2014
Accepted 27 December 2014
Available online 5 January 2015

Keywords:
DVFS
Checkpointing
Transient fault

ABSTRACT

This paper addresses the energy minimization issue when executing real-time applications that have stringent reliability and deadline requirements. To guarantee the satisfaction of the application's reliability and deadline requirements, checkpointing, Dynamic Voltage Frequency Scaling (DVFS) and backward fault recovery techniques are used. We formally prove that if using backward fault recovery, executing an application with a uniform frequency or neighboring frequencies if the desired frequency is not available, not only consumes the minimal energy but also results in the highest system reliability. Based on this theoretical conclusion, we develop a strategy that utilizes DVFS and checkpointing techniques to execute real-time applications so that not only the applications reliability and deadline requirements are guaranteed, but also the energy consumption for executing the applications is minimized. The developed strategy needs at most one execution frequency change during the execution of an application, hence, the execution overhead caused by frequency switching is small, which makes the strategy particularly useful for processors with a large frequency switching overhead. We empirically compare the developed real-time application execution strategy with recently published work. The experimental results show that, without sacrificing reliability and deadline satisfaction guarantees, the proposed approach can save up to 12% more energy when compared with other approaches.

1. Introduction

Power/energy consumption and system reliability have become increasingly critical in real-time system designs. As IC technology continues to scale, more and more transistors are integrated into a single chip causing power consumption to increase dramatically. In the meantime, as the aggressive scaling of CMOS technology continues, transistors become more vulnerable to radiation related external impacts [22], which often leads to rapid system reliability degradation. As reliability issues become more prominent, many design techniques dealing with the interplay of energy consumption and reliability are proposed [15,11].

However, power/energy conservation and reliability enhancement are often at odds. Taking the Dynamic Voltage and Frequency Scaling (DVFS) as an example, DVFS is a widely used technique for power management [31]. Recent work [25,28] has shown that transient fault rate increases dramatically when supply voltage for an IC chip is scaled down. Hence, more system resources are needed to recover from transient faults. Furthermore, real-time applications often have deadline constraints, reducing a task's working frequency increases its execution time and hence potentially causes tasks to miss their deadlines. As a result, it becomes a challenge to design a system that consumes the least amount of energy, but at the same time guarantees that both reliability and deadline constraints are satisfied.

To recover from transient faults, a commonly used technique is to utilize the slack time, i.e. the time differences between an application's completion time and its deadline, to do backward fault recovery [2]. Task re-execution and checkpointing are two common techniques used for backward fault recovery. With these techniques, when a fault occurs, the computation is repeated from the most recent checkpoint, rather than from its beginning. However, taking a checkpoint also takes time and consumes energy. Hence, where and when to take checkpoints during the application's execution needs to be well planned.

In this paper, we study a strategy that utilizes DVFS, checkpointing and fault recovery techniques to minimize energy consumption and at the same time guarantee the satisfaction of reliability and deadline constraints required by the application. The main contribution of the paper is threefold. First, it formally

* Corresponding author.
E-mail addresses: zli80@iit.edu (Z. Li), ren@iit.edu (S. Ren), gang.quan@fiu.edu (G. Quan).

http://dx.doi.org/10.1016/j.sysarc.2014.12.002
1383-7621/

proves that using backward fault recovery, if a processor actively executes an application for the same time duration, then executing the application with the same frequency, or neighboring frequencies if the desired frequency is not available, not only consumes the minimal energy but also results in the highest system reliability. Second, it presents algorithms for quickly determining an application's execution strategy, checkpointing strategy and fault recovery strategy, respectively. The proposed strategies not only guarantees both reliability and deadline requirements, but also minimizes the energy consumption. Third, it empirically evaluates algorithms by comparing them with those found in recent literature.

The rest of the paper is organized as follows. Related work is discussed in Section 2. Section 3 first introduces models and definitions used in the paper and then formally defines the problem to be addressed. The theoretical foundations are established in Section 4. Task set execution strategy and checkpointing strategy determinations are provided in Section 5 and Section 6, respectively. Section 7 presents the empirical evaluations. Finally, the conclusion is given in Section 8.

2. Related work

There has been increased research focusing on using DVFS and fault recovery strategy to minimize energy consumption and guarantee the reliability and deadline constraints.

Research found that lowering task execution frequency reduces processor's energy consumption [29,14]. In addition, Rizvandi showed that there is an optimal processor operation frequency that minimizes processor energy consumption [29]. However, the downside of using DVFS to lower processor execution frequency is that the lower the execution frequency, the higher the transient fault rate [14], and the lower the reliability if no actions are taken to deal with the increased transient fault rate. When fault recovery is not considered, Aupy et al. [30] proved that executing the whole task under the same frequency not only achieves the minimal energy but also the highest reliability.

Fault recovery technique usually uses a portion of slack time, i.e. the time difference between task's deadline and its execution time, to re-execute the failed tasks to improve system's reliability. To maintain system's required reliability, while at the same time, minimize the energy consumption, some heuristics as to how to decide the portion of slack time for fault recovery and task execution frequencies are proposed. Zhu et al. [25] developed the longest task first (LIF) and the slack usage efficiency factor (SUEF) based approaches [17]. Zhao et al. improved these approaches and proposed the shared recovery technique, which allows all tasks to share the reserved slack time, but only allows a single fault recovery during the entire application's execution. Recently, Zhao et al. [27] further extended the work and developed the generalized shared recovery approach which allows multiple fault recoveries. In addition, they also developed a uniform frequency (UF) approach and a heuristic incremental reliability configuration search (IRCS) algorithm to determine tasks' execution frequency. UF selects the lowest uniform frequency that satisfies the reliability and deadline requirements to execute the whole application. IRCS essentially adopts a dynamic programming approach and searches for a suitable frequency for each task to maximize energy savings.

However, task re-execution recovery strategy requires the entire task to be re-executed if a fault occurs. This strategy results in a large recovery cost if the tasks have long execution times and hence compromises energy saving performance. An alternative approach is to use a checkpointing strategy.

The optimal checkpointing strategy for *single* task execution is studied by Zhu et al. [28]. However, real-time applications often consist of multiple tasks with precedence relationship among tasks. Hence, the approach developed for single task execution may not be directly applied to task set execution where the reliability requirement is enforced on the task set, rather than on a single task. Punnekkat et al. [5] have studied the checkpointing strategy and developed a heuristic approach to decide where to take checkpoints for a task set. However, their approach is based on the assumption that the number of fault occurrences during the application's execution is given. Such an assumption is not appropriate for DVFS enabled systems as tasks may be executed under different frequencies which may result in different fault arrival rate.

Different from the work mentioned above, we study checkpointing strategy and task execution strategy for a set of dependent tasks on a DVFS enabled platform. Our goal is to minimize system's energy consumption without compromising reliability and deadline guarantees. In our work, we do not assume the number of fault occurrence is known at a prior.

3. System models and definitions

In this section, we introduce the models and definitions the rest of the paper is based upon.

3.1. Models

3.1.1. Processor model

The processor is DVFS enabled with q different working frequencies, i.e. $F = \{f_1, \ldots, f_q\}$ with $f_i < f_j$ if $i < j$, and $f_1 = f_{\min}, f_q = f_{\max}$.

In the following discussion, we assume the frequency values are normalized with respect to $f_{\max}$, i.e. $f_{\max} = 1$.

3.1.2. Application model

An application considered in this paper contains a set of tasks and is modeled as a directed acyclic graph [9], i.e., $A = G(V, E)$. Each task in the application is represented by a vertex $v_i \in V$ in the graph, the dependency between two connected tasks is represented by a directed edge $e_i = (v_i, v_j) \in E$. The application repeats periodically with a period p and the end-to-end deadline requirement for one iteration is $D \leqslant p$.

An example of an application task graph is given in Fig. 1. Task v_i's worst case execution time (WCET) under the maximum frequency $f_{\max}$ is denoted as c_i. We further assume the data transmission time cost on the edges is negligible.

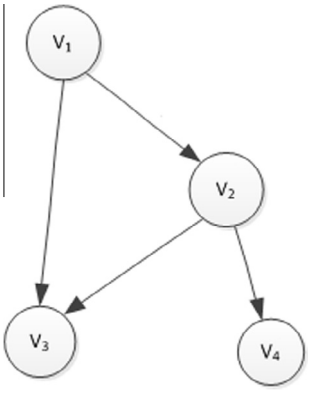


Fig. 1. DAG-based application.

3.1.3. Energy model

The energy model used in this paper is the same as in [26,19,28]. In particular, the power consumption of a system under operating frequency f is:

$$P(f) = P_s + hP_a = P_s + h(P_{\text{ind}} + C_{\text{ef}} f^{\theta}) \tag{1}$$

where P_s is the static power used to maintain the system in a standby state. The static power cannot be saved unless the system is turned off. P_a is the active power used when the system is in the working state. P_a has two components, frequency independent power (P_{ind}), such as the power used for memory and I/O operations, and frequency dependent power ($C_{\text{ef}} f^{\theta}$). Parameters C_{ef} and θ are system dependent constants and $\theta \geqslant 2$ [19,4]. Boolean parameter $h = 1$ indicates the system is in the working state, and $h = 0$ indicates the system is in the standby state.

We assume the system is always on and hence focus only on active power saving. As the energy consumption due to voltage and frequency scaling is independent of P_s, without loss of generality, we set $P_s = 0$.

We made the same assumption as in [26] that task execution time is linearly related to working frequency. Therefore, when task v_i is executed under frequency $f(v_i)$, its execution time is $\frac{c_i}{f(v_i)}$ and the energy consumption can be represented as:

$$\begin{aligned} E(f(v_i), c_i) &= (P_{\text{ind}} + C_{\text{ef}} f(v_i)^{\theta}) \tfrac{c_i}{f(v_i)} \\ &= P_{\text{ind}} \tfrac{c_i}{f(v_i)} + C_{\text{ef}} c_i f(v_i)^{\theta-1} \end{aligned} \tag{2}$$

From (2) it is clear that scaling down the processing frequency reduces frequency dependent energy ($C_{\text{ef}} c_i f(v_i)^{\theta-1}$). However, it also increases frequency independent energy ($P_{\text{ind}} \frac{c_i}{f(v_i)}$) due to longer execution time caused by the lower execution frequency. As a result, there is a balanced point, i.e. the *energy-efficient frequency* (f_{ee}) and further scaling down the processing frequency below f_{ee} will increase the total energy consumption. Early studies [26,28] concluded that

$$f_{ee} = \sqrt[\theta]{\frac{P_{\text{ind}}}{C_{\text{ef}}(\theta - 1)}} \tag{3}$$

3.1.4. Transient fault model

Although both permanent and transient faults may occur during task execution, transient faults are found more frequent than permanent faults [1,3]. Hence, in this paper, we focus on transient faults. Transient faults, also called soft errors, are usually triggered by radiations in semiconductor circuits. Various factors such as cosmic ray, transistor size and chip's supply voltage and frequency, could impact system's transient fault rate. Though several approaches have come out to estimate the accurate transient fault rate of a hardware platform [12,18,21], taking all the impact factors into consideration to derive a precise model is still a very challenging issue [10,28,16]. Hence, the research community generally assumes that transient fault rate follows Poisson distribution with an average fault rate λ [28,27,33]. When a system is running under frequency f, the average transient fault rate is usually expressed as:

$$\lambda(f) = \hat{\lambda}_0 10^{-\hat{d}f} \tag{4}$$

where $\hat{\lambda}_0 = \lambda_0 10^{\frac{d}{1-f_{\min}}}, \hat{d} = \frac{d}{1-f_{\min}}$. λ_0 is the average fault arrival rate when system running under the maximum frequency $f_{\max}$. Value $d\ (> 0)$ is a system-dependent constant, which indicates the sensitivity of the system's fault arrival rate to system voltage and frequency scaling, the larger the d value, the more sensitive the fault arrival rate to voltage and frequency scaling.

Under the fault rate model given by (4), if a task is executed under frequency f for t time units, the probability of exactly $k\ (\geqslant 0)$ transient faults occurring during a task's execution period can be expressed as [20]:

$$p(k, f, t) = \frac{(\lambda(f)t)^k e^{-\lambda(f)t}}{k!} \tag{5}$$

3.2. Definitions and notations

Checkpointing overhead (ε): the time overhead of taking a checkpoint. Different tasks may have different checkpointing overheads.

Task segments ($\vec{v}_i$): when checkpoints are inserted into a task, the task is partitioned into sections which are called task segments. For task v_i, we use $\vec{v}_i = [v_{i1}, \ldots, v_{il}]$ to denote all its segments. Execution time of v_{ij} includes the checkpointing overhead ε under execution frequency f_{max}. Notation $c_i' = \sum_{k=1}^{l} v_{ik}$ is to used denote the total execution time of all the task segments of v_i.

Task segment length ($len(i)$): the length of the ith *longest* task segment in $\vec{V}$ when they are executed under f_{max}.

Application checkpointing strategy (S_{cp}): the strategy that decides where to insert checkpoints in a given application, i.e., how each task in the application shall be segmented. It is represented by $S_{\text{cp}} = [\vec{v}_1, \ldots, \vec{v}_n]$.

Application processing strategy (S_{ps}): the strategy that decides the frequencies and the duration of each frequency that the application shall be executed with. It is represented by $S_{\text{ps}} = [(f_1, t_1), \ldots, (f_m, t_m)]$, which indicates the application will be executed under frequencies $f_1, \ldots, f_m$ for $t_1, \ldots, t_m$ time units, respectively. We abbreviate it as $S_{\text{ps}} = (\overrightarrow{(f_i, t_i)}; m)$, where m is the number of selected frequency and $m \leqslant q$.

Expected number of faults under a given processing strategy ($\varphi(S_{\text{ps}})$): for a given processing strategy $S_{\text{ps}} = (\overrightarrow{(f_i, t_i)}; m)$, $\varphi(S_{\text{ps}}) = \sum_{i=1}^{m} \lambda(f_i) t_i$ defines the expected number of faults when a task set is executed under the processing strategy S_{ps}.

Application recovery strategy (S_{rc}): the strategy that decides the number of faults (k) to be tolerated and the operating frequency (f) that failed task segments need to be re-executed with. It is denoted as $S_{\text{rc}} = (f, k)$. In this paper, $f_{\max}$ is assumed for task re-execution, i.e. $S_{\text{rc}} = (f_{\max}, k)$ and the application recovery strategy refers to the number of faults to be tolerated, i.e. k.

Application reliability ($R(S_{\text{cp}}, S_{\text{ps}}, S_{\text{rc}})$): the probability of successfully executing an application A under the given $S_{\text{cp}}, S_{\text{ps}}$, and S_{rc} strategies.

Based on the models and definitions introduced above, we are to formulate the problem the paper is to address.

3.3. Problem formulation

With the above definition, our checkpointing, processing and recovery (CPR) strategy decision problem can be formulated as follows:

Problem 1. (The CPR Strategy Decision Problem). Given a DVFS enabled processor with q different processing frequencies, i.e. $F = \{f_1, \ldots, f_q\}$, where $f_1 = f_{\min}, f_q = f_{\max}$ and $f_i < f_j$ if $i < j$, and a DAG-based application $A = G(V, E)$. Assume the worst case execution time of each task $\tau_i \in V$ under $f_{\max}$ is c_i, and the application's reliability and deadline constraints are R_g and D, respectively, decide a checkpointing strategy $S_{\text{cp}} = \vec{V}$, a processing strategy $S_{\text{ps}} = (\overrightarrow{(f_i, t_i)}; m)$, and a recovery strategy $S_{\text{rc}} = (f_{\max}, k)$ that satisfy the following:

Objective:

$$\min E(S_{cp}, S_{ps}, S_{rc})$$

Subject to:

$$R(S_{cp}, S_{ps}, S_{rc}) \geqslant R_g \tag{6}$$

$$\sum_{i=1}^{m} t_i + \sum_{j=1}^{k} len(j) \leqslant D \tag{7}$$

$$\sum_{i=1}^{m} f_i t_i = \sum_{v_i \in V} c_i' \tag{8}$$

where $E(S_{cp}, S_{ps}, S_{rc})$ is the energy consumption of executing the application under given strategies. Formula (6) denotes the reliability constraint, i.e. system's reliability meet the requirement. Formula (8) indicates the work load constraint, i.e. all the tasks within the application should be finished under the processing strategy $S_{ps} = ((\vec{f_i}, t_i); m)$. The deadline constraint is represented by formula (7), where $\sum_{i=1}^{m} t_i$ denotes the total task execution time and $\sum_{j=1}^{k} len(j)$ indicates the time reservation for task recoveries. It is worth pointing out that tasks' recovery is under f_{max}. As transient fault rate under f_{max} is rather low, we do not reserve time for task recovery under f_{max}.

Frequency switching also has the extra time and energy cost, we first assume the cost is negligible and later in Section 6 we will give the justification for this assumption. In the formulated problem, the checkpointing strategy, processing strategy, and recovery strategy are tightly coupled with each other. If the processing strategy chooses a lower frequency for execution, in order to meet reliability and deadline constraints, the recovery strategy may need to tolerate more faults and more checkpoints may be required by the checkpointing strategy to reduce the duration of recovery. Our approach is to uncouple them and solve the problem in two steps: first, decide the application processing strategy and recovery strategy for a given checkpointing strategy, and second, determine the checkpointing strategy. It is worth pointing out that the reliability aware energy minimization problem is NP-hard [17] and our approach falls into the heuristic category. Hence, it is possible the found solution may not be globally optimal.

Before presenting the application's processing, recovery and checkpointing strategies, we first lay the theoretical foundations extended from our earlier work [33] upon which the strategies are established.

4. Theoretical foundations

Lemma 1. *For a given application A, under the worst case scenario, i.e. tolerating k faults requires re-executing the longest k task segments, the application reliability under the checkpointing strategy* $S_{cp} = [\vec{v_1}, \ldots, \vec{v_n}]$, *processing strategy* $S_{ps} = (((\vec{f_i}, t_i); m)$, *and recovery strategy* $(f_{max}, k))$ *can be represented as:*

$$R^w(S_{cp}, S_{ps}, S_{rc}) = \sum_{i=0}^{k} \left[\frac{\left(\sum_{j=1}^{m} \lambda(f_j) t_j\right)^i e^{-\sum_{j=1}^{m} \lambda(f_j) t_j}}{i!} \cdot e^{-\lambda(f_{max}) \left(\sum_{j=1}^{i} len(j)\right)} \right] \tag{9}$$

Proof. We prove the lemma by induction on the number of frequencies (m) used in the processing strategy.

Step 1: When $m = 1$, i.e. $S_{ps} = [(f_1, t_1)]$, frequency f_1 is used for the entire execution of the application. According to formula (5), the probability of exactly i faults occurring under the processing strategy $S_{ps} = [(f_1, t_1)]$ is:

$$\frac{(\lambda(f_1) t_1)^i e^{-\lambda(f_1) t_1}}{i!}$$

In the worst case scenario, to recover these i faults, the longest i task segments are re-executed. The probability of successfully recovering these faults is:

$$e^{-\lambda(f_{max}) \left(\sum_{j=1}^{i} len(j)\right)}$$

Then the probability of successfully executing the application by tolerating exactly i faults can be written as:

$$\frac{(\lambda(f_1) t_1)^i e^{-\lambda(f_1) t_1}}{i!} e^{-\lambda(f_{max}) \left(\sum_{j=1}^{i} len(j)\right)}$$

Since $S_{rc} = (f_{max}, k)$ means up to k faults can be tolerated, then $R^w(S_{cp}, S_{ps}, S_{rc})$, i.e. the probability of successfully completing the application by tolerating at most k faults is:

$$R^w(S_{cp}, S_{ps}, S_{rc}) = \sum_{i=0}^{k} \left[\frac{(\lambda(f_1) t_1)^i e^{-\lambda(f_1) t_1}}{i!} \cdot e^{-\lambda(f_{max}) \left(\sum_{j=1}^{i} len(j)\right)} \right] \tag{10}$$

Hence, the Lemma holds for the case $m = 1$.

Step 2: Suppose $m = n\ (> 1)$, we have

$$R^w(S_{cp}, S_{ps}, S_{rc}) = \sum_{i=0}^{k} \left[\frac{\left(\sum_{j=1}^{n} \lambda(f_j) t_j\right)^i e^{-\sum_{j=1}^{n} \lambda(f_j) t_j}}{i!} \cdot e^{-\lambda(f_{max}) \left(\sum_{j=1}^{i} len(j)\right)} \right] \tag{11}$$

Step 3: When $m = n + 1$, i.e. $S_{ps} = [(f_1, t_1), (f_2, t_2), \ldots, (f_{n+1}, t_{n+1})]$. If exactly i faults happen during the application's execution, there must be $l\ (0 \leqslant l \leqslant i)$ of them occurring when the processing frequency is $f \in \{f_1, \ldots, f_n\}$ and the remaining $i - l$ faults occurring under $f = f_{n+1}$. Then the probability of executing the application successfully by recovering exactly i faults is:

$$\sum_{l=0}^{i} \left[\frac{\left(\sum_{j=1}^{n} \lambda(f_j) t_j\right)^l e^{-\left(\sum_{j=1}^{n} \lambda(f_j) t_j\right)}}{l!} \cdot \frac{(\lambda(f_{n+1}) t_{n+1})^{(i-l)} e^{-\lambda(f_{n+1}) t_{n+1}}}{(i-l)!} \cdot e^{-\lambda(f_{max}) \left(\sum_{j=1}^{i} len(j)\right)} \right]$$

Since up to k faults can be tolerated, $R^w(S_{cp}, S_{ps}, S_{rc})$ can be written as:

$$R^w(S_{cp}, S_{ps}, S_{rc}) = \sum_{i=0}^{k} \sum_{l=0}^{i} \left[\frac{\left(\sum_{j=1}^{n} \lambda(f_j) t_j\right)^l e^{-\left(\sum_{j=1}^{n} \lambda(f_j) t_j\right)}}{l!} \cdot \frac{(\lambda(f_{n+1}) t_{n+1})^{(i-l)} e^{-\lambda(f_{n+1}) t_{n+1}}}{(i-l)!} \cdot e^{-\lambda(f_{max}) \left(\sum_{j=1}^{i} len(j)\right)} \right]$$

According to the binomial formula, i.e.

$$\sum_{l=0}^{i}\frac{x^l y^{(i-l)}}{l!(i-l)!}=\frac{(x+y)^i}{i!}$$

we have

$$R^w(S_{cp},S_{ps},S_{rc})=\sum_{i=0}^{k}\left[\frac{\left(\sum_{j=1}^{m}\lambda(f_j)t_j\right)^i e^{-\sum_{j=1}^{m}\lambda(f_j)t_j}}{i!}\cdot e^{-\lambda(f_{max})\left(\sum_{j=1}^{i}len(j)\right)}\right]$$

These conclude the proof for Lemma 1. □

It is worth pointing out that formula (9) gives the application's reliability under the worst case scenario, i.e. the k faults occur in the longest k task segments. However, in reality, faults can occur in short task segments or multiple faults may occur in the same task segment. In other words, it may *not* necessarily need k longest recoveries to tolerate k faults. However, it is very hard to derive a closed formula for $R(S_{cp},S_{ps},S_{rc})$ where all possibilities of fault occurrences are taken into consideration. Nevertheless, we argue $R^w(S_{cp},S_{ps},S_{rc})$ can be used to approximate $R(S_{cp},S_{ps},S_{rc})$ with negligible approximation error.

Although different positions where faults occur result in different fault recovery durations, the positions only impact the reliability of fault recoveries, i.e. they may result in a higher value of $e^{-\lambda(f_{max})\left(\sum_{j=1}^{i}len(j)\right)}$ in formula (9). Therefore, $R^w(S_{cp},S_{ps},S_{rc})$ is the lower bound of $R(S_{cp},S_{ps},S_{rc})$, i.e., $R^w(S_{cp},S_{ps},S_{rc})\leqslant R(S_{cp},S_{ps},S_{rc})$. Hence, if we can guarantee $R^w(S_{cp},S_{ps},S_{rc})\geqslant R_g$, then the reliability requirement $R(S_{cp},S_{ps},S_{rc})\geqslant R_g$ is also satisfied.

In addition, all recoveries are executed under f_{max} and the fault arrival rate under f_{max} is very low (may as low as 10^{-9} [23]), i.e. $\lambda(f_{max})$ approaches to zero, hence, the reliability of fault recoveries $e^{-\lambda(f_{max})\left(\sum_{j=1}^{i}len(j)\right)}$ approaches 1. Therefore, the value of $R(S_{cp},S_{ps},S_{rc})$ and $R^w(S_{cp},S_{ps},S_{rc})$ are very close and we can use $R^w(S_{cp},S_{ps},S_{rc})$ to approximate $R(S_{cp},S_{ps},S_{rc})$, i.e.

$$R(S_{cp},S_{ps},S_{rc})=\sum_{i=0}^{k}\left[\frac{\left(\sum_{j=1}^{m}\lambda(f_j)t_j\right)^i e^{-\sum_{j=1}^{m}\lambda(f_j)t_j}}{i!}\cdot e^{-\lambda(f_{max})\left(\sum_{j=1}^{i}len(j)\right)}\right] \tag{12}$$

Based on the above analysis, we are ready to develop the theory about how to assign processing frequencies to tasks so that the system achieves highest reliability.

Lemma 2. *Given two different processing strategies S_{ps}^1 and S_{ps}^2, if $\varphi(S_{ps}^1)\leqslant\varphi(S_{ps}^2)$, then $R(S_{cp},S_{ps}^1,S_{rc})\geqslant R(S_{cp},S_{ps}^2,S_{rc})$, where $\varphi(S_{ps})=\sum_{i=1}^{m}\lambda(f_i)t_i$ denotes the expected number of faults under processing strategy S_{ps}.* □

Proof. To simplify the notation, let $b(i)=e^{-\lambda(f_{max})\sum_{j=1}^{i}len(j)}$. Formula (12) can be written as:

$$R(S_{cp},S_{ps},S_{rc})=\sum_{i=0}^{k}\frac{(\varphi(S_{ps}))^i e^{-\varphi(S_{ps})}}{i!}\cdot b(i)$$

The first derivative of $R(S_{cp},S_{ps},S_{rc})$ with respect to $\varphi(S_{ps})$ is

$$\frac{\partial(R(S_{cp},S_{ps},S_{rc}))}{\partial(\varphi(S_{ps}))}=e^{-\varphi(S_{ps})}\cdot\left(\sum_{i=0}^{k-1}\frac{(\varphi(S_{ps}))^i(b(i+1)-b(i))}{i!}-\frac{b(k)(\varphi(S_{ps}))^k}{k!}\right)$$

It is not difficult to see that when $i\geqslant 0, b(i+1)<b(i)$ and $0<b(i)\leqslant 1$. Hence, we have $\frac{\partial(R(S_{cp},S_{ps},S_{rc}))}{\partial(\varphi(S_{ps}))}<0$, which implies $R(S_{cp},S_{ps},S_{rc})$ is a decreasing function of $\varphi(S_{ps})$. Since $\varphi(S_{ps}^1)\leqslant\varphi(S_{ps}^2)$, we have $R(S_{cp},S_{ps}^1,S_{rc})\geqslant R(S_{cp},S_{ps}^2,S_{rc})$. □

Lemma 3. *For a given checkpointing strategy and a fixed execution time T, i.e. assuming different execution strategies take the same amount of execution time T to complete the application, the expected number of faults is minimized when the application is executed under a uniform frequency $f_u=\frac{\sum_{v_i\in V}c_i'}{T}$ if $f_u\in F$, or neighboring frequencies f_j and f_{j+1} with $f_j<f_u<f_{j+1}$ if $f_u\notin F$, i.e.*

1. *if $f_u\in F$, then*

$$\varphi([(f_u,T)])=min\left\{\varphi(\overrightarrow{(f_i,t_i)};m)|f_i\in F\wedge\left(\sum_{i=1}^{m}t_i=T\right)\wedge(f_uT=\sum_{i=1}^{m}f_it_i=\sum_{v_i\in V}c_i')\right\}$$

2. *if $f_u\notin F\wedge f_j,f_{j+1}\in F\wedge f_j<f_u<f_{j+1}$, then*

$$\varphi([(f_j,t),(f_{j+1},T-t)])=min\left\{\varphi(\overrightarrow{(f_i,t_i)};m)|f_i\in F\wedge\left(\sum_{i=1}^{m}t_i=T\right)\wedge\left(\sum_{i=1}^{m}f_it_i=f_jt+f_{j+1}(T-t)=\sum_{v_i\in V}c_i'\right)\right\}.$$ □

Proof. We prove the lemma based on the property of convex functions.[1] Since $S_{ps}=(\overrightarrow{(f_i,t_i)};m)$ represents a processing strategy satisfying $\sum_{i=1}^{m}f_it_i=\sum_{v_i\in V}c_i'$ and $\sum_{i=1}^{m}t_i=T$, then:

$$\varphi(\overrightarrow{(f_i,t_i)};m)=\sum_{i=1}^{m}\lambda(f_i)t_i$$

If $f_u\in F$, then

$$\varphi([(f_u,T)])=\lambda(f_u)T=\lambda\left(\frac{\sum_{i=1}^{m}f_it_i}{\sum_{i=1}^{m}t_i}\right)\left(\sum_{i=1}^{m}t_i\right)$$

As $\lambda(x)$ is a convex function of x, based on the properties of the convex function (formula (13)), we have

$$\varphi(\overrightarrow{(f_i,t_i)};m)\geqslant\varphi([(f_u,T)])$$

A similar proof can be derived for the case when $f_u\notin F$. □

Integrating Lemma 2 and Lemma 3, we have the following theorem:

Theorem 1. *For a fixed execution time T and a given checkpointing strategy S_{cp}, executing an application with a uniform frequency $f_u=\frac{\sum_{v_i\in V}c_i'}{T}$ if $f_u\in F$, or neighboring frequencies f_j and f_{j+1} if $f_u\notin F, f_i,f_{j+1}\in F$, and $f_i<f_u<f_{j+1}$, results in the highest system reliability.*

With S_{cp}, S_{ps}, and S_{rc}, we can calculate the expected energy consumption for executing an application. As the probability of fault occurrence is relatively small, the expected energy consumption

[1] If $g(x)$ is a convex function, $n(\geqslant 2)\in I^+, x_i,t_i\in\Re^+$ and $x_i<x_k$ if $i<k$, then we have:

$$\sum_{i=1}^{n}g(x_i)t_i\geqslant g(x_j)t_j+g(x_{j+1})t_{j+1}\geqslant g\left(\frac{\sum_{i=1}^{n}x_it_i}{\sum_{i=1}^{n}t_i}\right)\left(\sum_{i=1}^{n}t_i\right) \tag{13}$$

where $x_jt_j+x_{j+1}t_{j+1}=\sum_{i=1}^{n}x_it_i, x_j<\frac{\sum_{i=1}^{n}x_it_i}{\sum_{i=1}^{n}t_i}<x_{j+1}$, and $t_j+t_{j+1}=\sum_{i=1}^{n}t_i$.

for fault recovery is negligible [13] compared to the expected energy consumption for task set execution. Hence, the expected energy consumption $E(S_{cp}, S_{ps}, S_{rc})$ for executing an application with checkpointing strategy S_{cp}, processing strategy $S_{ps} = (\overrightarrow{(f_i, t_i)}; m)$, and fault recovery strategy S_{rc} can be represented as:

$$E(S_{cp}, S_{ps}, S_{rc}) = \sum_{i=1}^{m} (P_{ind} + C_{ef}(f_i)^{\theta}) t_i$$

Previous research [7,31] has studied the relationship between execution frequency and energy consumption and concluded that when the energy model is a convex function of frequency, using a uniform frequency, or neighboring frequencies if the desired frequency is not available, is the optimal strategy for energy saving purposes. We summarize this in Lemma 4.

Lemma 4. *For a fixed execution time T and a given checkpointing strategy S_{cp}, the minimal energy consumption can be achieved if the application is executed under a uniform frequency $f_u = \frac{\sum_{v_i \in V} c_i'}{T}$ if $f_u \in F$, or neighboring frequencies f_j and f_{j+1} with $f_j < f_u < f_{j+1}$ if $f_u \notin F$.* □

With the objective to minimize the energy consumption while at the same time, maximize the system's reliability, by integrating the Lemma 1 and Lemma 4, we can obtain the following conclusion:

Lemma 5. *For fixed execution time T, and the given checkpointing strategy S_{cp}, the highest reliability and the minimal energy consumption can be achieved when the application is executed under a uniform frequency $f_u = \frac{\sum_{v_i \in V} c_i'}{T}$ if $f_u \in F$, or neighboring frequencies f_j and f_{j+1} with $f_j < f_u < f_{j+1}$ if $f_u \notin F$.* □

Example 1. As shown in Fig. 2(a), consider an application, which has two tasks A and B with execution time 10 ms and 15 ms, respectively. The deadline of the application is $D = 45$ ms and the checkpointing overhead $\varepsilon = 1$ ms. We assume the system parameters related to the energy model are set as: $P_{ind} = 0.05$, $C_{ef} = 1$, and $\theta = 3$ [27]. The available discrete frequencies $F = \{0.4, 0.5, \ldots, 1\}$. If the checkpointing strategy takes one checkpoint in task B and the recovery strategy allows one fault recovery, then the recovery block with duration of 10 ms is reserved, which means the total time duration $T = 35$ ms can be used for the whole application execution. Based on Lemma 5, we obtain the uniform frequency $f_u = (10 + 15 + 1)/35 = 0.74$, then the processing strategy is shown as Fig. 2(b). As the frequency 0.74 is unavailable, based on Lemma 5, the neighboring frequencies of 0.7 and 0.8 are used instead. Fixed execution time means all the processing strategies execute exactly the same duration, i.e. 35 ms in this example. Suppose the processing strategy is $S_{ps} = ((0.7, x), (0.8, y))$, then we have $x \times 0.7 + y \times 0.8 = 10 + 15 + 1$ and $x + y = 35$. Solving the above equations, we have $x = 20$ and $y = 15$ and the corresponding processing strategy can be depicted as Fig. 2(c).

For the problem formulated in Section 3.3, system's energy consumption needs to be minimized under the condition that both reliability and deadline constraints are satisfied. Hence, Lemma 5 can not be directly applied to solve the problem we are to address. However, we can utilize Lemma 5 to derive the application's working frequency that meets objective with these constraints.

According to formula (12), if S_{cp} and S_{rc} are given, then the value of $\varphi(S_{ps})$ i.e. φ_0, which guarantees the reliability requirement R_g can be obtained by solving $R(S_{cp}, S_{ps}, S_{rc}) = R_g$.

Lemma 6. *If executing an application under a uniform frequency f can guarantee the satisfaction of reliability constraint R_g, then we have $f \geqslant f_r$ where*

$$\lambda(f_r)\left(\frac{\sum_{v_i \in V} c_i'}{f_r}\right) = \varphi_0 \tag{14}$$

□

Proof. To simplify the notation, let $g(f) = \lambda(f)\frac{\sum_{v_i \in V} c_i'}{f} = \frac{\lambda_0(\sum_{v_i \in V} c_i')}{10^{df} f}$. It is not difficult to see that $g(f)$ is a monotonically decreasing function of variable f. Since $g(f_r) = \varphi_0$, based on Lemma 2, in order to satisfy the reliability requirement, $g(f) \geqslant \varphi_0$ should be satisfied, which implies $f \geqslant f_r$. □

Lemma 7. *If executing an application under a uniform frequency f can guarantee the satisfaction of deadline constraint D, then $f \geqslant f_d$, where*

$$f_d = \frac{\sum_{v_i \in V} c_i'}{D - \sum_{i=1}^{k} len(i)} \tag{15}$$

□

Proof. We consider the worst case scenario when the longest k task segments are re-executed to tolerate k faults. To guarantee that the application successfully completes the execution before its deadline D, all tasks in the application must be executed in the time duration of $D - \sum_{i=1}^{k} len(i)$. Hence, the lowest uniform frequency is $f_d = \frac{\sum_{v_i \in V} c_i'}{D - \sum_{i=1}^{k} len(i)}$, and if $f \geqslant f_d$, the application deadline can be guaranteed.

Based on the definition of energy-efficient frequency f_{ee} (formula (3)), Lemma 3, and Lemma 7, the minimal frequency that guarantees the energy-efficient frequency constraint f_{ee}, reliability constraint R_g, and deadline constraint D can be determined as:

$$f_{rde} = \max\{f_{ee}, f_r, f_d\} \tag{16}$$

where f_{ee}, f_r, and f_d are given by (3), (14) and (15), respectively.

With formula (16), for given checkpointing strategy S_{cp} and recovery strategy S_{rc}, the application's processing frequency f_{rde} can be directly calculated. However, if $f_{rde} \notin F$, i.e., f_{rde} is not an available discrete frequency, according to the Lemma 5, the neighboring frequencies f_l and f_{l+1} should be used instead, where $f_l < f_{rde} < f_{l+1}$ and $f_l, f_{l+1} \in F$. Based on these analysis, next, we give our application execution strategy determination algorithm.

5. Determine application execution strategy

Application execution strategy determines how application tasks are processed, i.e. with what execution frequencies and for how long under each execution frequency, and how application tasks are recovered if faults occur. In other words, an application's execution strategy is the composition of application processing strategy and recovery strategy.

The processing strategy S_{ps} varies based on the recovery strategy S_{rc}, i.e. the number of faults to be tolerated. In order to derive the processing strategy with the minimal energy consumption, we start from zero fault recovery, i.e. $k = 0$, to search for the minimum number of faults to be recovered without violating the reliability constraint. If $f_r > f_d$, there is some slack time remaining due to a higher frequency being used to meet reliability constraints. In this

case, the remaining slack time is utilized to increase k, therefore increasing the reliability and at the same time to scale down the frequency to reduce energy consumption. On the other hand, $f_d > f_r$ implies that all the slack time is used out and the frequency cannot be lowered any further, hence $f_{\text{rde}} = \max\{f_d, f_{ee}\}$. If $f_{\text{rde}} \notin F$, based on Lemma 5, the neighboring frequencies f_l and f_{l+1} with $f_l < f_{\text{rde}} < f_{l+1}$ are used instead.

From the energy saving perspective, the longer the execution time under f_l, the less energy consumption there is. Therefore, the frequency switching instant, i.e. the time instant when the working frequency is switched from f_l to f_{l+1}, is delayed until the reliability and deadline requirements cannot be satisfied.

Algorithm 1 gives the detailed procedure of our proposed strategy for application execution strategy determination (ExD).

Algorithm 1. ExD($S_{\text{cp}}, V, R_g, D, f_{ee}, F$)

1: $k^{\text{opt}} = 0; f_{\text{rde}}^{\text{opt}} = f_{\max}; f'_{ee} = \min\{f_i | f_i \geqslant f_{ee}, f_i \in F\}$
2: $T = \sum_{i=1}^{|V|} |c'_i|; S_{\text{ps}}^{\text{opt}} = (f_{\max}, \frac{T}{f_{\text{rde}}}); S_{\text{rc}}^{\text{opt}} = (f_{\max}, 0)$
3: $slack = D - T; k = 0$
4: **while** $slack \geqslant 0$ **do**
5: calculate f_{rde} based on formula (16) with $f_{ee} = f'_{ee}$
6: $S_{\text{ps}} = (f_{\text{rde}}, \frac{T}{f_{\text{rde}}}); S_{\text{rc}} = (f_{\max}, k)$
7: **if** $f_{\text{rde}} \notin F$ **then**
8: find f_l and f_{l+1} with $f_l < f_{\text{rde}} < f_{l+1}$
9: S_{ps} = FIND_PS($S_{\text{cp}}, S_{\text{ps}}, V, R_g, D, f_l, f_{l+1}$)
10: **end if**
11: calculate $E_o = E(S_{\text{cp}}, S_{\text{ps}}, S_{\text{rc}})$
12: **if** $E^{\text{opt}} > E_o$ **then**
13: $E^{\text{opt}} = E_o; S_{\text{ps}}^{\text{opt}} = S_{\text{ps}}; S_{\text{rc}}^{\text{opt}} = S_{\text{rc}}$
14: **end if**
15: $k = k + 1; slack = slack - len(k)$
16: **end while**
17: **return** $S_{\text{ps}}^{\text{opt}}, S_{\text{rc}}^{\text{opt}}$

Lines 1–3 initialize variables used in the algorithm. The `while` loop (Lines 4–16) searches for the optimal processing strategy S_{ps} and recovery strategy S_{rc} for the given checkpointing strategy S_{cp}. For each possible number of fault recovery k, we find the corresponding processing strategy S_{ps}, if $f_{\text{rde}} \in F$, set $S_{\text{ps}} = (f_{\text{rde}}, \frac{T}{f_{\text{rde}}})$ (Line 6), otherwise, calculate it using algorithm `FIND_PS` (Line 9). The processing strategy S_{ps} that consumes the least amount of energy and the corresponding recovery strategy S_{rc} are recorded as the final solutions (Line 13). Line 8 finds the frequencies f_l and f_{l+1}, which are used to execute the entire task set. Line 9 i.e., Algorithm `FIND_PS` is to determine the frequency each task shall use for execution. Since the tasks in the application have dependencies, task execution order must satisfy the precedence constraint. Hence, we first sort all the tasks in the application by topological sorting to guarantee the precedence constraint is satisfied.

In Algorithm 2, we assume that task set V is the one after topological sorting. In addition, Lines 3–10 find the maximum p such that executing the first $p-1$ tasks in V under f_l can satisfy the deadline and reliability constraints. In order to further delay the frequency switching point, Lines 11–18 split the pth task into multiple partitions and execute as many partitions under f_l as possible if only both deadline and reliability constraints are not violated. β, which is a pre-defined integer constant, is the number of partitions.

Algorithm 2. FIND_PS($S_{\text{cp}}, S_{\text{ps}}, V, R_g, D, f_l, f_{l+1}$)

1: $T = \sum_{i=1}^{|V|} c'_i$
2: $t_l = \frac{T}{f_l}; t_{l+1} = 0; p = 0$
3: **for** $i = 1$ *to* $|V|$ **do**
4: $t_l = t_l - \frac{c'_i}{f_l}; t_{l+1} = t_{l+1} + \frac{c'_i}{f_{l+1}}$
5: $S_{\text{ps}} = ((f_l, t_l), (f_{l+1}, t_{l+1}))$
6: **if** $R(S_{\text{cp}}, S_{\text{ps}}, S_{\text{rc}}) \geqslant R_g \wedge t_l + t_{l+1} \leqslant D - \sum_{i=1}^{k} len(i)$ **then**
7: $p = i$
8: break
9: **end if**
10: **end for**
11: **for** $i = 1$ *to* β **do**
12: $\sigma = \frac{c'_p}{\beta}, t_l = t_l + \frac{\sigma}{f_l}; t_{l+1} = t_{l+1} - \frac{\sigma}{f_{l+1}}$
13: $S_{\text{ps}} = ((f_l, t_l), (f_{l+1}, t_{l+1}))$
14: **if** $R(S_{\text{cp}}, S_{\text{ps}}, S_{\text{rc}}) < R_g \vee t_l + t_{l+1} > D - \sum_{i=1}^{k} len(i)$ **then**
15: $S_{\text{ps}} = ((f_l, t_l - \frac{\sigma}{f_l}), (f_{l+1}, t_{l+1} + \frac{\sigma}{f_l}))$
16: break
17: **end if**
18: **end for**
19: **return** S_{ps}

Approximate the calculation of f_r: In Algorithm 1, in order to calculate f_{rde} (Line 5), we need the value of f_r which can be obtained by solving Eq. (14). However solving Eq. (14) is time consuming. To reduce the time cost, we search for an approximate value from $f_{\min}$ to $f_{\max}$ with step δ, where δ is set as $x\% f_{\max}$. The first frequency found after i steps, i.e. $f = f_{\min} + i \times \delta$, that satisfies $R(S_{\text{cp}}, (f, \frac{\sum_{\tau_i \in V} c'_i}{f}), S_{\text{rc}}) \geqslant R_g$ is assigned to f_r. Hence, the time cost to get f_r is a constant.

Time Complexity: The time complexity of Algorithm 1 is dominated by the `while` loop (lines 3–15). Since the total number of task segments is $|\vec{V}|$, the number of iterations of the `while` loop should be $O(|\vec{V}|)$. As the time complexity of `FIND_PS` is $O(|V|)$ (Line 8), where $|V|$ is the number of tasks in the application, hence the total time complexity is $O(|\vec{V}||V|)$.

6. Determine application checkpointing strategy

A checkpointing strategy is often designed under the assumption that the number of transient faults in a specific period is known or can be calculated from the statistical fault arrival model [8]. However, in the situation stated in this paper, the number of faults that can occur is neither set by the reliability requirement nor can be derived directly from the statistical fault model since the processing strategy is unknown and different processing strategies can result in different fault arrival rates. Even if given the number of transient faults that needs to be tolerated during the execution, finding the optimal checkpointing strategy is still difficult [5]. If the task set only contains one task, the optimal checkpointing strategy is to split the task into equal segments by inserting checkpoints. However, when the task set has multiple tasks and task execution times are different, faults occurring in different tasks require different time durations for fault recovery. Without taking checkpoints, in the worst case scenario, k faults could happen in the execution of the longest k tasks. As deadlines must be guaranteed for hard real-time systems, the slack time with the duration equals to the longest k

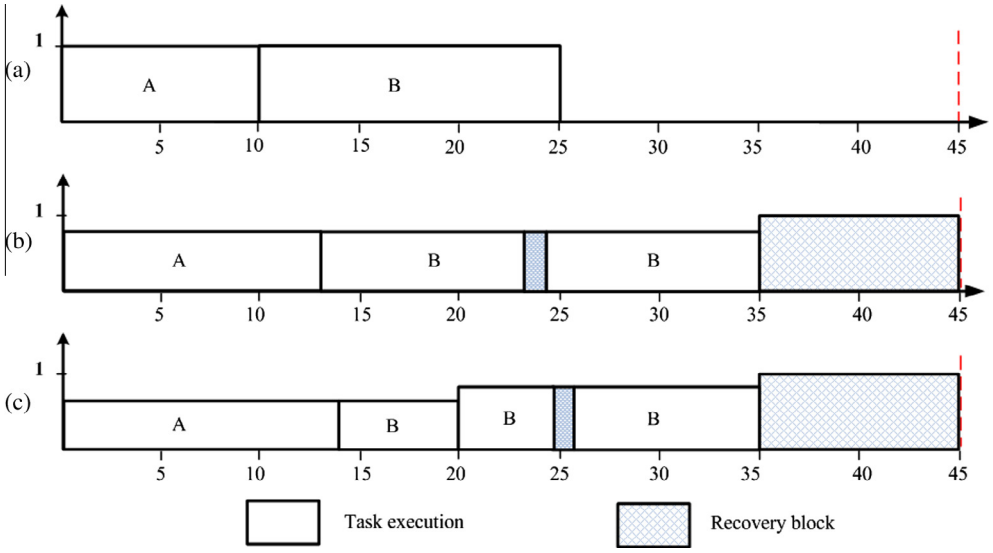


Fig. 2. Processing strategies of Example 1.

tasks must be reserved for fault recoveries. By taking checkpoints in the task set, tasks are divided into task segments and the k longest task segments are reserved. Hence, where to take checkpoints significantly impacts the time reservation for fault recovery.

Based on the theoretical analysis in Section 5, though we have no knowledge about what the final execution strategy is, for a given checkpointing strategy, the execution strategy can be derived using Algorithm 1. Hence, one of the approaches is, for each possible checkpointing strategy, we use Algorithm 1 to derive its execution strategy. Then the checkpointing strategy whose execution strategy results in the least amount of energy consumption is chosen as the one. However, exploring all the possible checkpointing strategies is computationally inefficient, so a heuristic approach is developed instead. If k fault recoveries are required, the reserved slack time is determined by the longest k task segments. Therefore, we always add new checkpoints into the task having the longest task segment, and distribute all the added checkpoints equally to partition the long task into shorter task segments. Starting with no checkpoints, one checkpoint is added to the current checkpointing strategy for each iteration until the maximum number of checkpoints is reached or until the slack time runs out.

The objective of taking checkpoints is to split the task (or task segments) into shorter task segments and hence reduce the duration of fault recovery. However, taking checkpoints itself has time overhead. For a task v_i with the WCET c_i and checkpointing overhead ε, taking more than $max\_cp(v_i)$ checkpoints will inversely increase the duration of fault recovery. Hence, for task v_i, its maximum number of checkpoints, i.e. $max\_cp(v_i)$, can be calculated as follows [5]:

$$max\_cp(v_i) = \left\lceil \sqrt{\frac{c_i}{\varepsilon}} \right\rceil \tag{17}$$

The details of checkpointing strategy determination algorithm (CkD) are given in Algorithm 3, where data structure $VC[i]$ keeps a record of where checkpoints are taken for task v_i. The values of $VC[i]$.wcet, $VC[i]$.chk, $VC[i]$.len and $VC[i]$.mcp represent task v_i's WCET under f_{max}, the number of inserted checkpoints, the length of the task segments within task v_i, and the maximum number of checkpoints for v_i, respectively.

Algorithm 3. `CkD`$(A, R_g, D, \varepsilon, F)$

1: sort all the tasks in V by topological sorting
2: **for** $i = 1$ *to* $|V|$ **do**
3: $VC[i]$.wcet = c_i; $VC[i]$.mcp = $max\_cp(v_i)$
4: $VC[i]$.chk = 0; $VC[i]$.len = c_i
5: **end for**
6: $k = 0; VC^{opt} = VC; E^{opt} = \infty$; $slack = D - \sum_{i=1}^{|V|} c_i$
7: $S_{ps}^{opt} = (f_{max}, \sum_{i=1}^{m} c_i)$; $S_{rc}^{opt} = (f_{max}, 0)$
8: **while** $slack > 0$ **do**
9: **if** $\exists j : VC[j].\text{len} = \max\{VC[i].\text{len} | VC[i].\text{chk} < VC[i].\text{mcp}\}$ **then**
10: $VC[j].\text{chk} = VC[j].\text{chk} + 1$;
11: $VC[j].\text{len} = \frac{c_i + \varepsilon \times VC[j].\text{chk}}{VC[j].\text{chk}}$
12: set S_{cp} based on VC
13: (S_{ps}, S_{rc}) = ExD (S_{cp}, V, R_g, D, F)
14: calculate $E_o = E(S_{cp}, S_{ps}, S_{rc})$
15: **if** $E^{opt} > E_o$ **then**
16: $E^{opt} = E_o; S_{cp}^{opt} = S_{cp}$; $S_{ps}^{opt} = S_{ps}; S_{rc}^{opt} = S_{rc}$;
17: **end if**
18: $slack = slack - \varepsilon$
19: **else**
20: break
21: **end if**
22: **end while**
23: **return** $S_{cp}^{opt}, S_{ps}^{opt}, S_{rc}^{opt}$

In Algorithm 3, Line 1 sorts all the tasks using topological sorting, Lines 2–7 initialize the variables, and Lines 8–22 compare all checkpointing strategies and record the one that has minimal energy consumption.

Time Complexity: The time complexity of Algorithm 3 is dominated by the `while` loop (lines 8–23). Since the maximum number of checkpoints in the whole application is $N = \sum_{i=1}^{|V|} max\_cp(v_i)$, the number of iterations in the `while` loop is hence $O(N)$. The N checkpoints partitions the application into $N + |V|$ task segments, i.e. $|\vec{V}| = N + |V|$. Hence, the time complexity of `ExD` (Line 13) is

$O((N+|V|)|V|)$ and the total time complexity of Algorithm 3 is $O((N+|V|)|V||N|)$.

Execution Frequency Switching Overhead: It is well-known that DVFS is an effective way to reduce energy consumption. However, recent studies [32,24] have shown that frequency switching has both time and energy overhead. Furthermore, frequent frequency switching may decrease the hardware component's life span. With our approach, in the case of fault-free execution, at most two frequencies are used to execute the whole task set, hence, there is at most one frequency switch. This shows that in addition to more energy saving, another big advantage of the developed approach is its negligible execution frequency switching overhead.

7. Performance evaluation

In this section, we evaluate the performance of our proposed approach from energy saving perspective. Three recent studies in the literature, i.e. *UF*, *IRCS* [27] and *GSSR-UNS-IS* [34], are chosen as baselines in our comparisons. The *UF* (uniform frequency) selects the lowest uniform frequency that can satisfy the reliability and deadline requirements to execute the whole application. The *IRCS* (Incremental Reliability Configuration Search) is a heuristic approach that searches for the best working frequencies for tasks based on the energy-reliability ratio, i.e. energy saving per unit reliability degradation. The *GSSR-UNS-IS* (General Subset Sharing with Uniform/Neighboring Scaling and Iterative Search) is the improvement of the *IRCS* by effectively removing the unnecessary resource reservation to leave more slack time for energy saving.

7.1. Experiment setting

In the experiments, the tested applications are generated by TGFF [6]. The results shown in the following figures are the average values of repeating the experiments for 1000 randomly generated applications. The system parameters related to the energy model are set as: $P_{\text{ind}} = 0.05$, $C_{\text{ef}} = 1$, and $\theta = 3$. These values are considered realistic and widely used in the research community [27,16]. The available discrete frequencies (normalized to f_{max}) are set as $F = \{0.4, 0.5, \ldots, 1\}$ and the checkpointing overhead ε is denoted as the time overhead under $f_{max} = 1$, which is assumed to be proportional to the average WCET of tasks in the application. For instance, $\varepsilon = 1\%$ means ε is 1% of average task execution time, i.e. $\frac{\sum_{\tau_i \in V} c_i}{|V|}$. If taking checkpoint under frequency f_i, then the time overhead is $\frac{\varepsilon}{f_i}$.

The energy consumption of the application under different execution strategies is normalized to the energy cost when the whole application is executed under $f_{\max}$. If R_0 is set to be the reliability of the application when the whole application is executed under $f_{\max}$ without fault recovery, then $1 - R_0$ is the corresponding probability of failure during the application's execution. We scale the probability of failure to vary the reliability requirement in our experiments. Particularly, we set $R_g = 1 - (1 - R_0)/S$, where S is a scaling factor.

The comparisons among the *CkD*, *UF*, *IRCS* and *GSSR-UNS-IS* approaches are from the following four perspectives:

1. Sensitivity to available slack time (L).
2. Sensitivity to reliability requirement (R_g).
3. Sensitivity to checkpointing overhead (ε).
4. Sensitivity to system's fault arrival rate (λ_0 and d).

where

$$L = \frac{D - \sum_{i=1}^{|V|} c_i}{\sum_{i=1}^{|V|} c_i}$$

is used to indicate the available slack time in the system.

The evaluation parameters are set as follows: $L = 1, S = 10$, $\varepsilon = 1\%, \lambda_0 = 10^{-6}$, and $d = 4$. When a specific aspect is evaluated, for instance, when the impact of available slack time is evaluated, the value of L varies, but the other parameters remain unchanged.

7.2. Experiment results and discussions

Sensitivity to available slack time

In this set of experiments, we increase the value of L from 0.2 to 1.6 and the experiment results are depicted in Fig. 3. Fig. 3 clear shows that our proposed *CkD* approach consumes the least amount of energy. It is 10% lower than the energy cost consumed by the *GSSR-UNS-IS* approach when $L = 1.6$. However, *IRCS* and *UF* approaches always consume more energy than that of *GSSR-UNS-IS*.

Sensitivity to reliability requirement R_g

This set of experiments investigates the energy saving performance of the four approaches under different reliability requirements. According to the results shown in Fig. 4, all the approaches consume more energy under a higher reliability requirement. This is because a higher reliability requirement requires more fault recoveries. As a result, less slack time can be used for reducing execution frequency. Compared with *GSSR-UNS-IS*, *IRCS* and *UF*, CkD consumes the least amount of energy.

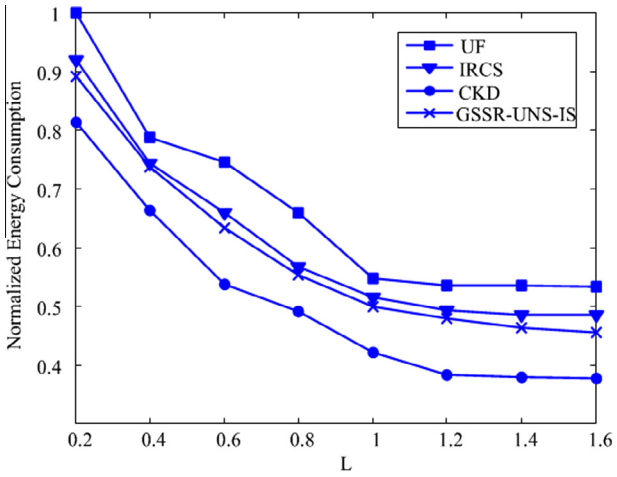


Fig. 3. Sensitivity to L ($\lambda_0 = 10^{-6}, d = 4, \varepsilon = 1\%, S = 10$).

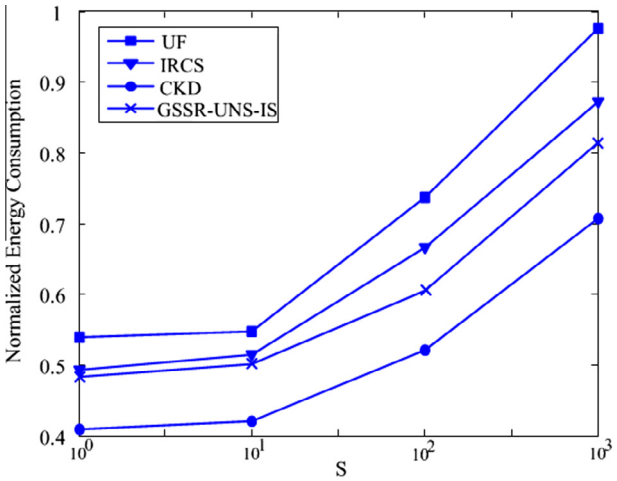


Fig. 4. Sensitivity to R_g ($\lambda_0 = 10^{-6}, d = 4, \varepsilon = 1\%, L = 1$).

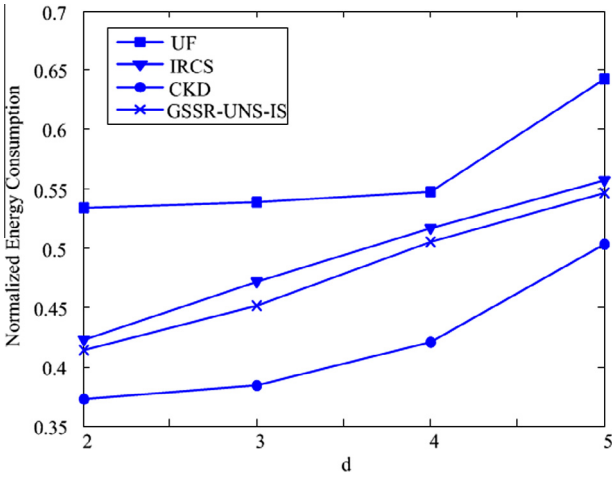


Fig. 5. Sensitivity to d ($\lambda_0 = 10^{-6}, \varepsilon = 1\%, L = 1, S = 10$).

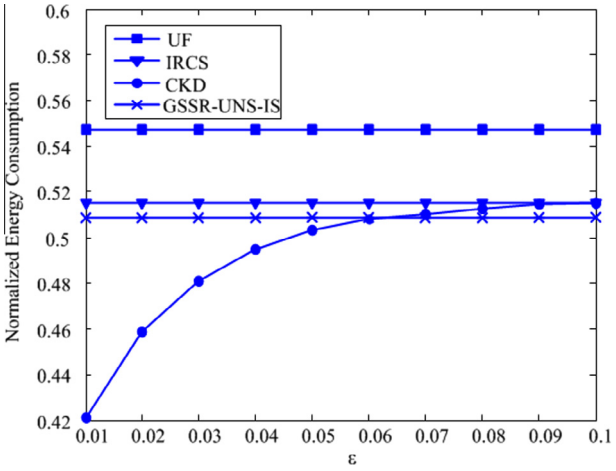


Fig. 7. Sensitivity to ε ($\lambda_0 = 10^{-6}, d = 4, L = 1, S = 10$).

When the reliability requirement becomes higher, i.e. S becomes larger, the advantage of CkD becomes more obvious. When $S = 1000$, i.e. $R_g = 1 - (1 - R_0)/1000$, the *CkD* approach can save about 12%, 18% and 28% more energy compared with the *GSSR-UNS-IS*, *IRCS* and *UF* approaches, respectively.

Sensitivity to the system's fault arrival rate

According to formula (4), the system parameters d and λ_0 have great impact on the fault arrival rate. This set of experiments investigates how sensitive each approach is to d and λ_0. We vary d between 2 and 5 and vary λ_0 between 10^{-9} and 10^{-6}, these are the typical values used in the related research work [27,25,17].

We first set $\lambda_0 = 10^{-6}$ and vary d from 2 to 5. The experiment results are shown in Fig. 5. When d becomes larger, all four approaches consume more energy. The experiments are then repeated with $d = 4$ and λ_0 varying from 10^{-9} to 10^{-6}. Based on the results illustrated in Fig. 6, all these four approaches consume more energy under a higher λ_0. This is due to the fact that a larger d and a higher λ_0 mean a higher fault arrival rate and hence more slack time is reserved for fault recovery and less can be used for energy saving. Among these four approaches, our proposed *CkD* approach wins the comparison and always saves the most energy.

Checkpointing overhead impact

Fig. 7 shows the impact of the checkpointing overhead on the performance of *CkD* approach. When ε is 1%, our proposed *CkD* approach can save about 8% more energy than *GSSR-UNS-IS* approach. However, the checkpointing overhead significantly impacts the energy saving performance. When ε is increased to 7% or even larger, the checkpointing overhead outweighs the gain by taking checkpoints in the application for energy saving purpose.

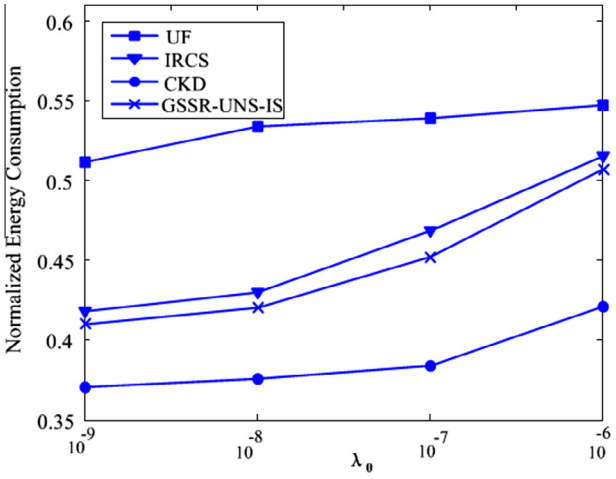


Fig. 6. Sensitivity to λ_0 ($d = 4, \varepsilon = 1\%, L = 1, S = 10$).

8. Conclusion

Reliability and deadline guarantee and power/energy conservation are the most critical issues in designing today's real-time system. However, these two design constraints are often at odds. This paper presents an approach that utilizes both DVFS and checkpointing techniques to reduce energy consumption while guaranteeing system reliability and deadline satisfaction of real-time task set. We formally prove that if a processor remains active within the same interval to accomplish the same workload, using the same constant speed not only consumes the least amount of energy, but also results in the highest system reliability. If such constant speed is not available, the neighboring speeds can be used to achieve the optimal solution. Based on the theoretical results, a novel DVFS strategy and checkpointing method are developed to minimize energy consumption and at the same time guarantee the satisfaction of system reliability and deadline requirements. Compared with existing approaches, extensive experimental results have shown that the proposed technique has better energy saving performance, i.e. up to 12% more energy saving compared with other existing approaches.

Acknowledgments

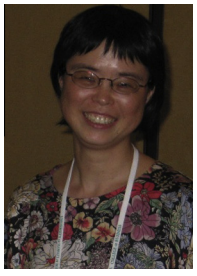
This work was supported, in part, by NSF CAREER 0746643, NSF CNS 1018731, NSF CNS-0917021 and NSF CNS-1018108.

References

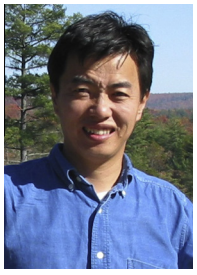
[1] X. Castillo, S.R. McConnel, D.P. Siewiorek, Derivation and calibration of a transient error reliability model, July 1982.
[2] Fault-tolerant Computing: Theory and Techniques, vol. 1, Prentice-Hall Inc., 1986.
[3] R.K. Iyer, D.J. Rossetti, M.C. Hsueh, Measurement and modeling of computer reliability as affected by system activity, Aug. 1986.
[4] T.D. Burd, R.W. Brodersen, Energy efficient CMOS microprocessor design, in: Proceedings of the 28th Hawaii International Conference on System Sciences, ser. HICSS.
[5] Sasikumar Punnekkat, Alan Burns, Analysis of checkpointing for schedulability of real-time systems, in: IEEE International Workshop on Real-Time Computing Systems and Applications (RTCSA).

[6] Robert P. Dick, David L. Rhodes, Wayne Wolf, Tgff: task graphs for free, in: Proceedings of the 6th International Workshop on Hardware/Software Codesign, ser. CODES/CASHE '98.
[7] Tohru Ishihara, Hiroto Yasuura, Voltage scheduling problem for dynamically variable voltage processors, in: Proceedings of the 1998 International Symposium on Low Power Electronics and Design, ser. ISLPED '98.
[8] Hyosoon Lee, Heonshik Shin, Sang-Lyul Min, Worst case timing requirement of real-time tasks with time redundancy, in: Proceedings of the Sixth International Conference on Real-Time Computing Systems and Applications (RTCSA), 1999.
[9] ed., Prentice Hall PTR, Upper Saddle River, NJ, USA, 2000.
[10] P. Shivakumar, M. Kistler, S.W. Keckler, D. Burger, L. Alvisi, Modeling the effect of technology trends on the soft error rate of combinational logic, in: Proceedings. International Conference on Dependable Systems and Networks, 2002. DSN 2002.
[11] Osman S. Unsal, Israel Koren, C. Mani Krishna, Towards energy-aware software-based fault tolerance in real-time systems, in: Proceedings of The International Symposium on Low Power Electronics Design, ser. ISLPED.
[12] Zhou Quming, K. Mohanram, Cost-effective radiation hardening technique for combinational logic, in: IEEE/ACM International Conference on Computer Aided Design, 2004. ICCAD-2004.
[13] Dakai Zhu, R. Melhem, D. Mosse, The effects of energy management on reliability in real-time embedded systems, in: Proceedings of the IEEE/ACM International conference on Computer-aided design, ser. ICCAD.
[14] D. Ernst, S. Das, S. Lee, D. Blaauw, T. Austin, T. Mudge, Nam Sung Kim, K. Flautner, Razor: circuit-level correction of timing errors for low-power operation, 2004.
[15] Atul Maheshwari, Student Member, Wayne Burleson, Senior Member, Russell Tessier, Trading off transient fault tolerance and power consumption in deep submicron (dsm) vlsi circuits, 2004.
[16] J.F. Ziegler, Trends in Electronic Reliability – Effects of Terrestrial Cosmic Rays, 2004. [Online]. Available: <http://www.srim.org/SER/SERTrends.htm>
[17] Dakai Zhu, H. Aydin, Energy management for real-time embedded systems with reliability requirements, in: Proceedings of IEEE/ACM International Conference on Computer-Aided Design, ser. ICCAD.
[18] Bin Zhang, Wei-Shen Wang, M. Orshansky, Faser: fast analysis of soft error susceptibility for cell-based designs, in: 7th International Symposium on Quality Electronic Design, 2006. ISQED '06.
[19] Hakan Aydin, Vinay Devadas, Dakai Zhu, System-level energy management for periodic real-time tasks, in: Proceedings of 27th IEEE International Real-Time Systems Symposium, ser. RTSS.
[20] Kyong Kim, Jong Kim, An adaptive DVS checkpointing scheme for fixed-priority tasks with reliability constraints in dependable real-time embedded systems, in: Embedded Software and Systems, ser. Lecture Notes in Computer Science.
[21] R.R. Rao, K. Chopra, D.T. Blaauw, D.M. Sylvester, Computing the soft error rate of a combinational logic circuit using parameterized descriptors, March 2007.
[22] Chandra, Vikas, Aitken, Robert, Impact of technology and voltage scaling on the soft error susceptibility in nanoscale CMOS, in: Proceedings of the IEEE International Symposium on Defect and Fault Tolerance of VLSI Systems, ser. DFT.
[23] Baoxian Zhao, H. Aydin, Dakai Zhu, Reliability-aware dynamic voltage scaling for energy-constrained real-time embedded systems, in: IEEE International Conference on Computer Design, 2008. ICCD 2008.
[24] Wonyoung Kim, Meeta S. Gupta, Gu-yeon Wei, David Brooks, System level analysis of fast, per-core DVFS using on-chip switching regulators, in: Proceedings of International Symposium on High-Performance Computer Architecture, 2008.
[25] Baoxian Zhao, Hakan Aydin, Dakai Zhu, Enhanced reliability-aware power management through shared recovery technique, in Proceedings of the International Conference on Computer-Aided Design, ser. ICCAD.
[26] Zhu Dakai, H. Aydin, Reliability-aware energy management for periodic real-time tasks, Oct. 2009.
[27] Baoxian Zhao, Hakan Aydin, Dakai Zhu, Generalized reliability-oriented energy management for real-time embedded applications, in: Proceedings of the Design Automation Conference, ser. DAC.
[28] Dakai Zhu, Reliability-aware dynamic energy management in dependable embedded real-time systems, 2011.
[29] Rizvandi, Nikzad Babaii, Javid Taheri, Albert Y. Zomaya, Some observations on optimal frequency selection in DVFS-based energy consumption minimization, Aug. 2011.
[30] Guillaume Aupy, Anne Benoit, Yves Robert, Energy-aware scheduling under reliability and makespan constraints, 2012.
[31] Nikzad Babaii Rizvandi, Albert Y. Zomaya, Young Choon Lee, Ali Javadzadeh Boloori, Javid Taheri, Multiple frequency selection in DVFS-enabled processors to minimize energy consumption, CoRR, 2012.
[32] Nasrin Jaberi, An open question about dependency of life time of hardware components and dynamic voltage scaling, 2012.
[33] Zheng Li, Li Wang, Shangping Ren, Gang Quan, Energy minimization for checkpointing-based approach to guaranteeing real-time systems reliability, in: IEEE 16th International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing (ISORC), 2013.
[34] Zheng Li, Li Wang, Shuhui Li, Shangping Ren, Gang Quan, Reliability guaranteed energy-aware frame-based task set execution strategy for hard real-time systems, Dec. 2013.

Zheng Li received the B.S. degree in Computer Science and M.S. degree in Communication and Information System from University of Electronic Science and Technology of China, in 2005 and 2008, respectively. He is currently a Ph.D candidate in the Department of Computer Science at the Illinois Institute of Technology. His research interests include real-time embedded and distributed systems.



Shangping Ren received her B.S. and M.S. in Computer Science from Hefei Polytechnique University in China, and Ph.D degree from the University of Illinois at Urbana-Champaign, in USA. She is currently an Associate Professor in the Department of Computer Science at the Illinois Institute of Technology. Her research interests include coordination models, distributed real-time systems, and programming languages. She is a senior member of the IEEE.



Gang Quan received the B.S. degree from Tsinghua University, Beijing, China, the M.S. degree from the Chinese Academy of Sciences, Beijing, China, and the Ph.D. degree from the University of Notre Dame, South Bend, IN. He is currently an Associate Professor with the Department of Electrical and Computer Engineering, Florida International University. His research interests include real-time systems, power-aware design, communication networks, and reconfigurable computing.