Department of Computing Science
Illinois Institute of Technology

Energy Minimization for Embedded Systems Using Checkpointing and Dynamic Voltage and Frequency Scaling (DVFS)

Zheng Li, Shangping Ren, Li Wang

Department of Computing Science
Illinois Institute of Technology
10 W. 31st Street, Chicago, IL 60616

Telephone (312) 567-3993
Email {zli80, ren, lw64}@hawk.iit.edu

Technical Report IIT-CS-008-04-2012

Acknowledgements

The research is support in part by NSF CAREER 0746643 and NSF CNS 1018731.

Contents

List of Figures

Abstract

In this paper, we study the problem of how to employ checkpointing fault recovery and Dynamic Voltage and Frequency Scaling (DVFS) techniques to minimize the energy consumption for embedded applications with stringent deadline and reliability constraints. We find that, similar to the dynamic energy reduction, a constant working frequency is optimal in terms of reliability maximization when running a real-time task within a specified interval. If such a speed is not available, then using two closest neighboring speeds is the optimal choice. We formulate our observations into theorems and prove them analytically. Based on these observations, we further developed heuristics to determine the number of checkpoints and task execution working frequencies. Experimental results show that our proposed execution strategies can save up to 30% more energy than recent work.

1 Introduction

For the past decade, extensive power aware research has been conducted across different design abstraction levels, ranging from transistor level to system level [1, 2]. As more and more transistors are integrated into a single chip, the chip power consumption has been increasing exponentially. As a result, more and more aggressive power aware reduction techniques have been proposed, such as using extremely low supply voltages and threshold voltages [3, 4]. While these techniques can greatly reduce energy consumption, the reliability of the entire application is often degraded. Furthermore, with the continuing scaling of CMOS technologies and reducing the design margins for higher performance, soft errors of digital systems caused by transient faults occur more frequently than ever. Therefore, how to most effectively save energy and, at the same time, ensure the system's reliability has increasingly become a prominent issue for embedded system designers [5].

Improving reliability of a system and its energy saving performance are at odds for applications with deadline constraints. First, in order to recover from the transient errors to improve the system's reliability, extra computing resources must be reserved so that programs can be re-executed when fault strike. This implies that part of the slack time, i.e., time difference between the completion of a task and its deadline when the processor can potentially be idle, must be reserved for the purpose of recovering from faults, rather than being utilized for energy saving. The more transient faults that need to be recovered, the more resources need to be reserved, and thus the less energy saving the system can have.

Earlier research in the area of power management under timing constraints has shown that Dynamic Voltage and Frequency Scaling (DVFS) technique is one of the most effective techniques to reduce energy consumption and guarantee deadline satisfication [6, 7, 8]. However, when scaling down the frequency and lowering down the device supplying voltage, the device fault rate increases and hence causes system reliability to degrate.

A few papers have been published on the study of the tradeoffs between energy reduction and system reliability. For instance, Zhu et al. [5] provided reliability-aware power management (RA-PM) scheme to address the tradeoffs. The RA-PM approach allocates a recovery task for *every* real-time task whose execution frequency is scaled down so that when an error does occur, the recover task can be executed to ensure the required reliability. In order to decide which tasks should be selected for execution under scaled down frequency for energy saving purpose, some heuristic approaches are proposed, such as longest task first (LTF) [9] and slack usage efficiency factor (SUEF) based heuristic algorithms [9].

It is not difficult to see that the RA-PM approach is conservative as it statically divides the available slack time to multiple recovery blocks for a pre-determined set of tasks. Zhao [10] proposed a shared recovery (SHR) technique. The SHR approach reserves a recovery block, which can be shared by multiple tasks that need to recover at run-time. However, this technique works only for a single fault recovery. In other words, when a fault occurs during an execution of a task and used up the recovery block, then the remaining tasks have to be executed under the maximum processing frequency. Zhao et al later extended the work to allow multiple fault recoveries [11]. A heuristic approach, i.e., the Incremental Reliability Configuration (IRCS), is provided to find the number of recovery tasks and task execution frequency assignments.

Backward fault recovery strategy is a commonly used method for fault recovery. It restores the system state to a previous safe state and repeats the computation if a fault happens [12, 13]. Task re-execution and checkpointing are the two common techniques for backward fault recovery. For task

re-execution technique, fault detection is done at the end of each task. Many recent work in real-time community has focused on using task re-execution for fault recovery [9, 10, 11]. If fault occurs during the initial execution and no intermediated state is recorded, to recover from the fault, the task has to be re-executed from the beginning which could be time-consuming for a task with long execution time.

A widely adopted approach to overcome the long re-execution time is to use checkpoining. By adding checkpoints into tasks, long tasks are sliced into several short task segments. When a fault is detected, the recovery only requires re-executing the segment where the fault is located and hence reduces the recovery cost and potentially saves energy. However, checkpointing itself has overhead and, as pointed out Zhu et al. [13], the checkpointing overhead can have significant impact on total energy savings. Therefore, how frequent to take a checkpoint and where to take a checkpoint have to be carefully delt with; otherwise, it may compromise the performance with respect to system's energy saving and reliability.

In this paper, we are interested in studying the problem of minimizing the energy consumption for embedded applications with given reliability and deadline constraints. We adopt the checkpointing based fault recovery technique in our research. We investigated how we can choose appropriate numbers of checkpoints for the given real-time tasks (checkpointing stragegy) and also judiciously vary the processor's processing frequencis when executing these tasks (execution stragety) such that their deadline and reliability constraints can be satisfied, and the energy consumption can be minimized. To this end, we made a number of interesting findings, which we formally formulate them as lemmas and theorems with proofs. We then present two heuristic algorithm, i.e. one to determine the checkpoint numbers and the second one to determine the task execution frequencies accordingly. Simulation results show that our approach can achieve up to 30% energy saving improvement over the most recent work.

The rest of the paper is organized as following. In Section 2, we introduce system models and definitions the research is built upon, and then formally formulate the problem this paper intends to address. We discuss the optimal checkpointing stragety and the optimal execution strategy that minimize energy consumption and guarantee the satisfication of application's reliability and deadline constraints in Section 3 and Section 4 for continuously and descretely scalable frequencies, respectively. Experimental results and discussions are presented in Section 5. We conclude in Section 6.

2 System Models and Problem Formulation

In this section, we introduce system models and definitions the research is built upon, and then formally formulate the problem this paper intends to address.

2.1 Models and Definitions

Processor Model

The processor is DVFS enabled with working frequency in the range of $[f_{\min}, f_{\max}]$, where $f_{\min} \leq f_{\max}$. We assume that the frequency values are normalized with respected to $f_{\max}$, that is, $f_{\max} = 1$.

Application Model

The application being considered has m independent tasks $\{\tau_1, \cdots, \tau_i, \cdots, \tau_m\}$ which share a common deadline D. The worst case execution time (WCET) of task τ_i under the maximum processor frequency $f_{\max}$ is c_i. When the processor runs at frequency f, $f_{\min} \leq f \leq f_{\max}$, the corresponding execution time of task τ_i is $\frac{c_i}{f}$.

Energy Model

We adopt the system-level energy model as given in [14]. In particular, if a processor operating under frequency f for t time duration, the consumed energy is given below:

$$E(f) = (P_{\text{ind}} + C_{\text{ef}} f^{C_m}) \times t \tag{1}$$

where P_{ind}, C_{ef}, and $C_m (\geq 2)$ are system-dependent, but frequency-independent constants.

From (1), it is not difficult to see that scaling down the processing frequency can save frequency-dependent energy, but, on the other hand, it can also increase the frequency-independent energy because of longer execution time due to lower frequency. Hence, the *Energy-Efficient frequency* (f_{ee}) exists,

under which, further scaling down the processing frequency will increase the total energy consumption. Previous research studies [14] have given the definition:

$$f_{ee} = \sqrt[C_m]{\frac{P_{\text{ind}}}{C_{\text{ef}}(C_m - 1)}} \tag{2}$$

Transient Fault and Reliability Model

In this paper, we only consider soft errors caused by transient faults. We take the same assumptions made in [14] that the arrival of transient faults follows Poisson distribution and in [5] that the average fault arrival rate at frequency f ($< f_{max}$) can be expressed as

$$\lambda(f) = \hat{\lambda}_0 10^{-\hat{p}f} \tag{3}$$

where $\hat{\lambda}_0 = \lambda_0 10^{\frac{p}{1-f_{\min}}}$, $\hat{p} = \frac{p}{1-f_{min}}$. λ_0 is the average fault arrival rate at f_{max} and $p(\geq 3)$ is a system-dependent constant, representing the sensitivity of transient fault due to DVFS.

If we define the reliability of an application as the probability of completing the execution successfully, then for the case without fault tolerance, the reliability of an application under processing frequency f can be represented as:

$$R_i(f) = e^{-\lambda(f)\cdot\frac{C}{f}} \tag{4}$$

where C is the total execution time under $f_{\max}$.

Base on the models defined above, it is not difficult to see that task execution order within an application has no impact on the application's energy consumption and reliability. Hence, in the following sections, we assume task τ_i is executed before τ_j if $i < j$.

To simplify the representation in the following sections, we introduce the following terms.

- Task segments ($\vec{\tau_i}$): when checkpoints are inserted into a task, the task is partitioned into sections which are called task segments. For task τ_i, we use $\vec{\tau_i} = [\tau_{i1}, \cdots, \tau_{il}]$ to denote all its segments in task τ_i, noticing that, if no checkpoint inserted, $\vec{\tau_i} = [\tau_1]$.

- Checkpointing strategy (S_{cp}): the strategy as to how many checkpoints are needed and where to insert the checkpoints in a task set. Hence, the checkpointing strategy decides task segments. We use $S_{\text{cp}} = (\vec{\mathcal{T}}; h)$ to denote a checkpointing strategy, where $\vec{\mathcal{T}} = \{\vec{\tau_1}, ..., \vec{\tau_m}\}$ and h is the total number of inserted checkpoints. Notation $len(i)$ will be used to denote the length of the ith *longest* task segment among all the ones in $\vec{\mathcal{T}}$. We also assume the checkpointing overhead q is a constant for a given application.

- Processing strategy (S_{ps}): the strategy as to under what frequencies an application should be executed and for how long under each frequency. The processing strategy is denoted as $S_{\text{ps}} = (\overrightarrow{(f_i, t_i)}; n)$, where $(\overrightarrow{(f_i, t_i)}; n) = [(f_1, t_1), \cdots, (f_n, t_n)]$. (f_i, t_i) indicates that the processor is operating under frequency f_i for t_i time units. It is worth pointing out that that multiple task or task segments can be executed under the same processing frequency and multiple frequencies can be used for a single task or task segment.

- Recovery strategy (S_{rc}): the strategy as to under what operating frequency a failed task or task segments should be re-executed. In this paper, we assume the maximum frequency, i.e., $f_{\max}$, is used for recovery, i.e., $S_{\text{rc}} = (f_{\max})$

- Application execution strategy (S_{app}): the composition of its checkpointing strategy, processing strategy, i.e., $(S_{\text{cp}}, S_{\text{ps}})$.

- Processing stage: the state when a processor is executing a task, or a task segment, or taking a checkpoint.

- Recovery stage: the state when a processor is re-executing a task or task segment when fault recoveries are invoked.

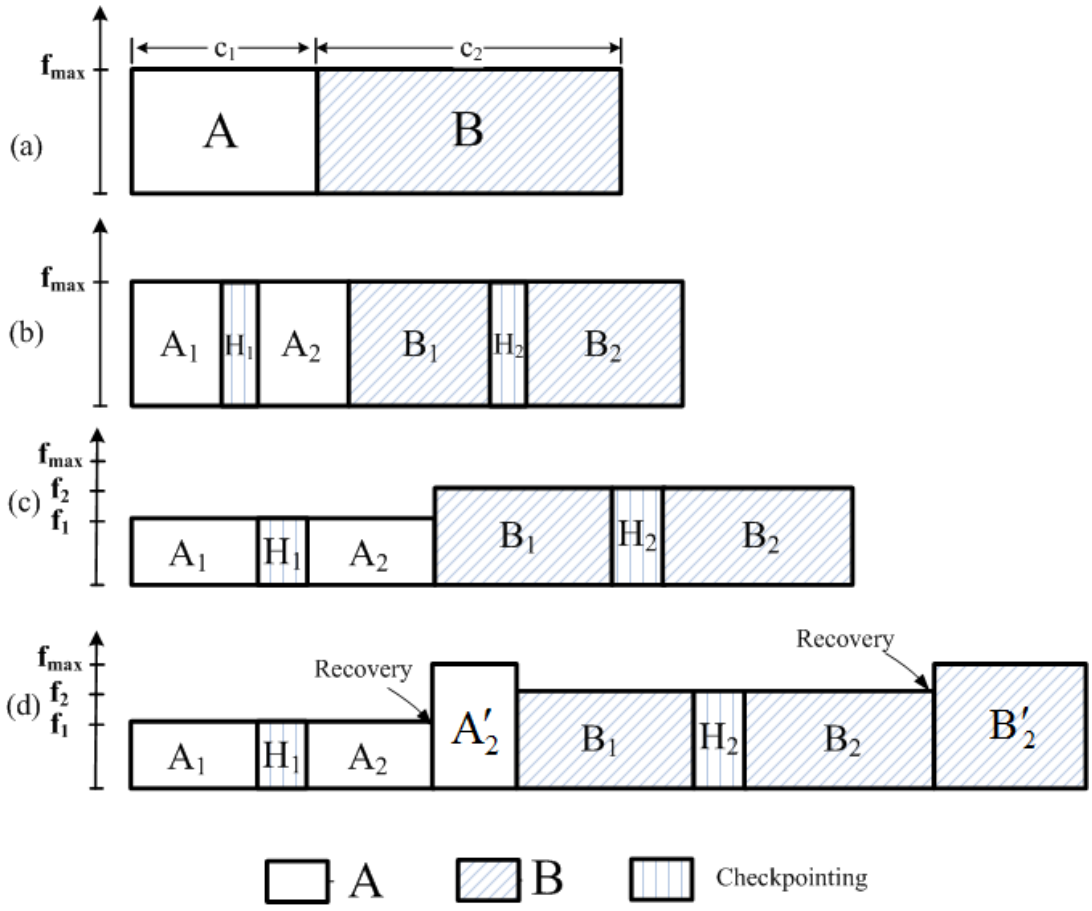

Figure 1: Example of application execution strategy

We use an example to further explain the concepts introduced so far.

Example 1. *Given an application with two tasks A and B and their WCET under f_{max} are c_1 and c_2, respectively, as shown in Fig. 1(a). Fig. 1(b) shows a checkpointing strategy where two checkpoints are added to the application, one in each task. Because of the checkpoints, tasks, including checkpointing overhead, are sliced into 4 task segments, i.e., $A_1 + H_1$, A_2, $B_1 + H_2$ and B_2. Hence, the checkpointing strategy S_{cp} can be represented as $\{[A_1 + H_1, A_2, B_1 + H_2, B_2]; 2\}$. Fig. 1(c) depicts a processing strategy, where frequency f_1 is used to execute $A_1 + H_1$ and A_2, and $B_1 + H_2$ and B_2 is executed under f_2. In other words, processing strategy is $S_{\text{ps}} = [(f_1, \frac{c_1+q}{f_1}), (f_2, \frac{c_2+q}{f_2})]$. Fig. 1(d) gives a fault recovery strategy, where faults are detected at the end of A_2 and B_2, their recovery are executed under f_{max}.*

□

Based on the models and terms introduced above, an application reliability and its energy consumption under a given application execution strategy can be derived.

Lemma 1. *Given an application $A = \{\tau_1, ..., \tau_m\}$, and its execution stragegy $S_{\text{app}} = (S_{\text{cp}}, S_{\text{ps}})$, where $S_{\text{cp}} = \{\vec{\mathcal{T}}; h\}$, $S_{\text{ps}} = (\overrightarrow{(f_i, t_i)}; n)$, respectively. If up to k faults are to be tolerated, then the application reliability is:*

$$R_A(S_{\text{cp}}, S_{\text{ps}}, k) = \sum_{i=0}^{k} \frac{(\sum_{j=1}^{n} \lambda(f_j)t_j)^i e^{-\sum_{j=1}^{n} \lambda(f_j)t_j}}{i!} \cdot e^{-\lambda(f_{\max})(\sum_{j=1}^{i} len(j))} \tag{5}$$

and the energy consumption is:

$$E_A(S_{\text{cp}}, S_{\text{ps}}, k) = \sum_{i=1}^{n} (P_{\text{ind}} + C_{\text{ef}} {f_i}^{C_m}) t_i \tag{6}$$

□

To avoid diverging from the main flow of the paper, we leave the proof in the appendix section.

Formula (5) and (6) can be simplified (7) and (8), respectively.

$$R_A(S_{\text{cp}}, S_{\text{ps}}, k) = r(\sum_{i=1}^{n} \lambda(f_i)t_i, S_{\text{cp}}, k) \tag{7}$$

$$E_A(S_{\text{cp}}, S_{\text{ps}}, k) = \sum_{i=1}^{n} \delta(f_i)t_i \tag{8}$$

where

$$r(x, S_{\text{cp}}, k) = \sum_{i=0}^{k} \frac{x^i e^{-x}}{i!} \cdot e^{-\lambda(f_{max})(\sum_{j=1}^{i} len(j))} \tag{9}$$

and

$$\delta(x) = P_{\text{ind}} + C_{\text{ef}} x^{C_m} \tag{10}$$

With the models defined above, we are ready to formally define the research problem this paper indents to solve, i.e., find an optimal application execution strategy for minimizing energy consumption while meeting the application's reliability and deadline constraints. We call the problem the ERD problem for short.

2.2 Problem Formulation

Problem 1. *Given an application A with m independent tasks, i.e., $A = \{\tau_1, \cdots, \tau_i, \cdots, \tau_m\}$ and the WCET of task τ_i under $f_{\max}$ is c_i and they share a common deadline D. The reliability requirement of the application is R_g and the available processing frequency is f, $f_{min} \leq f \leq f_{max}$ with $f_{min} \geq f_{ee}$. Decide an execution strategy (S_{app}), i.e., a checkpointing strategy $S_{\text{cp}} = (\vec{\mathcal{T}}; h)$ and a processing strategy $S_{\text{ps}} = (\overrightarrow{(f_i, t_i)}; n)$, with*
Objective:

$$\min E_A(S_{\text{cp}}, S_{\text{ps}}, k)$$

Subject to:

$$R_A(S_{\text{cp}}, S_{\text{ps}}, k) \geq R_g$$
$$\sum_{i=1}^{n} t_i + \sum_{j=1}^{k} len(j) \leq D$$

where k is the number of faults can be tolerated.

□

We address the problem in two steps: we first consider the case when the processor frequency can be continuously scaled within $[f_{\min}, f_{\max}]$ (Section 3), and then consider the case when there is only a set of discrete frequencies available for scaling (Section 4).

3 Optimal Application Execution Strategy under Continuous Frequence Scaling

In this section, we assume the processor's processing frequency can be continuously scaled and call the problem defined in Section 2.2 under this assumption as C-RDE problem.

3.1 Properties of the Optimal Execution Strategy

Lemma 2 (Uniform Frequency)**.** *Given an application A which has a set of independent tasks $\{\tau_1, \cdots, \tau_i, \cdots, \tau_m\}$ with τ_i's WCET as c_i under $f_{\max}$, and the checkpointing strategy $S_{\text{cp}} = \{\vec{\mathcal{T}}; h\}$ is determined, if all processing strategies take the same time D' to complete the application and up to $k(\geq 0)$ faults can be tolerated, then processing the application under uniform frequency $f_0 = max\{\frac{\sum_{i=1}^{m} c_i + qh}{D'}, f_{\min}\}$ has the best performance, i.e., achieves the highest reliability and consume the least energy consumption.*

Proof. We use $(\overrightarrow{(f_i, t_i)}; n) = [(f_1, t_i), \cdots, (f_n, t_n)]$ to denote any other processing strategy except $[(f_0, D')]$. As all the processing strategies have to complete the the same workload within the same time duration, therefore, we have:

$$\sum_{i=1}^{n} f_i t_i = f_0 D'$$

and,

$$\sum_{i=1}^{n} t_i = D'$$

Based on formula (7), the application reliability of processing strategy $(\overrightarrow{(f_i, t_i)}; n)$ and $[(f_0, D')]$ can be expressed as:

$$R_A(S_{\text{cp}}, (\overrightarrow{(f_i, t_i)}; n), k) = r(\sum_{i=1}^{n} \lambda(f_i) t_i, S_{\text{cp}}, k)$$

and,

$$R_A(S_{\text{cp}}, (\overrightarrow{(f_0, D')}; 1), k) = r(\lambda(f_0)D', S_{\text{cp}}, k)$$

respectively.

As $\lambda(x)$ is convex for $x > 0$, according to formula (11) [1], we have:

$$\sum_{i=1}^{n} \lambda(f_i)t_i \geq \lambda(f_0)D'$$

Furthermore, $r(x, S_{\text{cp}}, k)$ is a monotonically decreasing function when $x > 0$ for a given S_{cp} and k [2], hence:

$$r(\sum_{i=1}^{n} \lambda(f_i)t_i, S_{\text{cp}}, k) \leq r(\lambda(f_0)D', S_{\text{cp}}, k)$$

i.e.,

$$R_A(S_{\text{cp}}, (\overrightarrow{(f_i, t_i)}; n), k) \leq R_A(S_{\text{cp}}, \overrightarrow{(f_0, D')}; 1), k)$$

Similarly, according to formula (8), the energy consumption of the above two processing strategies can be expressed as:

$$E_A(S_{\text{cp}}, (\overrightarrow{(f_i, t_i)}; n), k) = \sum_{i=1}^{n} \delta(f_i)t_i$$

and,

$$E_A(S_{\text{cp}}, \overrightarrow{(f_0, D')}; 1), k) = \delta(f_0)D'$$

respectively. As $\delta(x)$ is a convex function for $x > 0$, then:

$$\sum_{i=1}^{n} \delta(f_i)t_i \geq \delta(f_0)D'$$

i.e.,

$$E_A(S_{\text{cp}}, (\overrightarrow{(f_i, t_i)}; n), k) \geq E_A(S_{\text{cp}}, (\overrightarrow{(f_0, D')}; 1), k).$$

These conclude the proof. □ □

Noticing that, Lemma 2 is also valid for the case that no checkpoints are inserted, which is the special case when $h = 0$. Although Lemma 2 gives the properties of the optimal processing strategy under a given checkpointing strategy, how to design the checkpointing strategy is yet to know, which will be discussed in the next section.

3.2 Checkpointing Based Application Execution Strategy

The question about a checkpointing strategy can be divided into two sub-questions:
1) Do we need to take checkpoints?
2) If needed, where to take the checkpoints?

Before giving the answer to question 1), we introduce two Lemmas first.

[1] Given a convex function $g(x)$, for $n \in I^+$ and $x_i, t_i \in \Re^+$, we have

$$\sum_{i=1}^{n} g(x_i)t_i \geq g(\frac{\sum_{i=1}^{n} x_i t_i}{\sum_{i=1}^{n} t_i})(\sum_{i=1}^{n} t_i) \tag{11}$$

which can be directly derived from convex function definition.

[2] The proof will be given in appendix.

Lemma 3 (Uniform Frequency under Deadline)**.** *Given an application A with a set of independent tasks* $\{\tau_1, \cdots, \tau_i, \cdots, \tau_m\}$ *with* τ_i*'s WCET as* c_i *under* $f_{\max}$*, and the end-to-end deadline is D. Without considering reliability constraint, the optimal execution strategy is to take zero checkpoint and executing the all the tasks under the uniform frequency* f_d*, which will consumes the least energy while meeting the deadline.*

$$f_d = \max\{\frac{\sum_{i=1}^{m} c_i}{D}, f_{\min}\} \tag{12}$$

Proof. Without considering reliability constraint, obviously, no need to take checkpoints. Then, based on Lemma 2, by setting $h = 0$, we get the above result.

□

Lemma 4 (Uniform Frequency for Reliability)**.** *Given an application A which has a set of independent tasks* $\{\tau_1, ..., \tau_i, \cdots, \tau_m\}$ *with* τ_i*'s WCET as* c_i *under* $f_{\max}$*. The application's reliability constraint is* R_g*. If all processing strategies take the same time to complete the application and zero fault is tolerated, without considering the deadline constraint, then processing the application under the uniform frequency* f_r *given by* (13) *consumes the least energy while meeting the reliability constraint.*

$$f_r = max\{f'_r, f_{min}\} \tag{13}$$

where f'_r *satisfies the following condition:*

$$10^{\hat{p} f'_r} f'_r = \frac{\hat{\lambda}_0 \sum_{i=1}^{m} c_i}{-ln R_g} \tag{14}$$

Proof. As zero fault tolerance, no need to take checkpoints, that is $h = 0$. Hence, based on the reliability model given in (4), if all the tasks in the application are executed under the uniform processing frequency f_r, to satisfy the reliability constraint, we have:

$$e^{-\lambda(f_r) \cdot \frac{\sum_{i=1}^{m} c_i}{f_r}} \geq R_g$$

i.e.,

$$10^{\hat{p} f_r} f_r \geq \frac{\hat{\lambda_0} \sum_{i=1}^{m} c_i}{-ln R_g}$$

As $10^{\hat{p}x} x$ is a strictly increasing function when $x > 0$, hence, if f'_r is the value to make the above inequality get the equal sign, then $f_r \geq f'_r$. As the processing frequency is no less than f_{min}, we have $f_r = max\{f'_r, f_{min}\}$. According to Lemma 2, executing the application under uniform frequency achieves highest reliability with minimum energy consumption, as f_r is the lowest processing frequency to meet reliability constraint (or the minimum available frequency), then using f_r to execute the application costs the least energy while meeting reliability constraint. □

As a valid execution strategy must meet both the reliability and deadline constraints, which are affected by the processing execution frequencies, we take the *conquer the dominant* approach.

Definition 1 (Dominant Relation ($\preceq$))**.** *For constraint* $\mathcal{C}_1$ *dominates* $\mathcal{C}_2$*, i.e.,* $\mathcal{C}_1 \preceq \mathcal{C}_2$ *iff the satisfaction of* $\mathcal{C}_1$ *implies the satisfaction of* $\mathcal{C}_2$*.* □

Based on Lemma 3 and Lemma 4, if $f_d \geq f_r$, the deadline constraint dominates the reliability constraint. Hence, the application's optimal executing strategy is to use frequency $f_{opt} = f_d$ for the entire execution and no need to take the checkpoints. However, if $f_r > f_d$, the reliability constraint is the dominant one. In this case, if there exists slack time before the deadline, we need to consider the question 2), i.e., where to take checkpoints to utilize the slack time to further reduce the energy consumption without violating the reliability constraint.

Different checkpointing strategy will partition a given application into different task segments, which will significantly impact the application execution performance with respect to the reliability and energy consumption. As we mentioned before, if k faults are guaranteed to be tolerated, the fault recovery stage may take the longest k task segments, that is, the length of longest k task segments determine the

duration of recovery stage, and the longest one contribute most. If we add more checkpoints to the task having the longest task segment, then we will get shorter task segments instead (noticing that, a task is always evenly splited by the inserted checkpoints), based on the new task segments, the length of longest k task segments may reduce. Hence, more slack time can be used for frequency adjustment to save the energy. Based on these senses, our strategy is, if more checkpoints available, we always insert it to the task that have the longest task segment, which is shown in Algorithm 1.

Algorithm 1 `INSERT-CHK(TC, q)`

1: sort all the TC by len in descending order
2: $TC[1].chk = TC[1].chk + 1$;
3: $TC[1].len = TC[1].wcet/TC[1].chk + q$;
4: **return** TC;

An array of data structure TC is used to record the checkpointing strategy for application A, one TC for each task, the four tags: *index*, *wcet*, *chk*, and *len* represents the task's index, WCET under f_{max}, number of checkpoints, and the length of task segment within the task, respectively. We first sort all the TC by the *len* in descending order (line 1) to get the task having the longest task segment, which is indicated by $TC(1).index$, then we add one more checkpoint to this task (line 2 - 3).

Having a checkpointing strategy, based on Lemma 2, we know that executing the application under a uniform frequency has the highest reliability and consumes the least energy. Now, we give the algorithm, Algorithm 2, for finding this optimal processing frequency.

Algorithm 2 `FIND-OPT` $(TC, R_g, D, \varepsilon, F)$

1: get the checkpointing strategy $S_{\text{cp}} = \{\vec{\mathcal{T}}; h\}$ from TC
2: calculate total number of task segments as N
3: **for** $k = 0$ *to* N **do**
4: **if** $R_A(S_{\text{cp}}, ((g(S_{\text{cp}}, k), \frac{C}{g(S_{\text{cp}},k)}); 1), k) < R_g$ and $g(S_{\text{cp}}, k) < f_{max}$ **then**
5: $k = k + 1$;
6: **else**
7: break;
8: **end if**
9: **end for**
10: $f_{\text{coarse}} = min\{g(S_{\text{cp}}, k), f_{\text{max}}\}$;
11: $f_{\text{opt}} = f_{\text{coarse}}$;
12: **if** $k > 0$ **then**
13: $f' = g(S_{\text{cp}}, k)$;
14: **while** $f' < f_{coarse}$ **do**
15: **if** $R_A(S_{\text{cp}}, ((f', \frac{C}{f'}); 1), k - 1) \geq R_g$ **then**
16: $f_{\text{opt}} = f'$;
17: $k = k - 1$;
18: break;
19: **else**
20: $f' = f' + \varepsilon$;
21: **end if**
22: **end while**
23: **end if**
24: $E_{\text{opt}} = E_A(S_{\text{cp}}, (f_{\text{opt}}, \frac{C}{(f_{\text{opt}}}); 1), k)$;
25: **return** $E_{\text{opt}}, f_{\text{opt}}, k$;

Before giving the detailed explanations for the algorithm, we introduce a function $g(S_{\text{cp}}, k)$ first, which is used to calculate the optimal processing frequency for the case when $D' = D - \sum_{i=1}^{k} len(i)$ in

Lemma 2.

$$g(S_{\text{cp}}, k) = max\{\frac{C + qh}{D - \sum_{i=1}^{k} len(i)}, f_{\min}\} \tag{15}$$

We can see executing the application under $g(S_{\text{cp}}, k)$ is optimal, i.e., has highest reliability and consumes least energy, when the all the available time except the one reserved for k fault recovery in recovery stage are used to process the application.

Algorithm 2 is divided in to two procedures: coarse-grain search (line 3 - 10) and fine-grain search (line 12 -23).

In the coarse-grain search stage, as shown in line 3 - 10, we set the granularity as the length of task segments. If the coarse-grain search returns with $k = 0$ at line 8, which means no fault tolerance is needed. Otherwise, k faults to be tolerated, where $k \geq 1$. The reason for increasing the tolerated fault from $k-1$ to k due to reliability constraint is not satisfied, i.e., $R_A(S_{\text{cp}}, (\overrightarrow{(g(S_{\text{cp}}, k-1), \frac{C}{g(S_{\text{cp}}, k-1)}); 1), k-1) < R_g$ and $g(S_{\text{cp}}, k-1) < f_{\max}$. Hence by adding one more tolerated faults, that is, increasing the recovery stage and decreasing the processing stage by the granularity of kth longest task segment, then we get the f_{coarse} (assume running the application under f_{max} can be guaranteed as a valid solution), which can meet the reliability and deadline constraint. Therefore, we can conclude that the optimal processing frequency is within the interval $[g(S_{\text{cp}}, k-1), f_{\text{coarse}}]$.

In fine-search stage (line 12 -23), we gradually increase the frequency from $g(S_{\text{cp}}, k-1)$ to f_{coarse} by the granularity of ε to approach the optimal solution.

Having Algorithm 1 and Algorithm 2, now we propose our execution strategy for C-RDE problem, which is given in Algorithm 3.

The basic idea is as follows. Adding checkpoints may reduce the recovery time and hence more slack time can be utilized to scale down the frequency to save energy; however, reduced frequency also reduces reliability. Hence, to meet application reliability constraint, we may need to increasing the number of tolerated faults, which, however, needs more recovery time, and hence, processing frequency should be scaled up and more energy will be consumed. Therefore, we can perform actions of taking checkpoints and increasing number of faults to be tolerated iteratively until no more energy can be further saved or no slack time is available for adding more checkpoints. Hence, we implement it in the following steps:

1. find the optimal processing strategy according using Algorithm 2 based on the current checkpointing strategy (The initial state is no checkpoints inserted).

2. compare the recorded application execution strategy with the one returned by step 1), then record the one with less energy consumption (Initially, the recorded execution strategy consume infinite energy).

3. if the slack time is available for more checkpoints, change the checkpointing strategy using Algorithm 1.

4. repeat the above three steps until no more checkpoints are available and return the recorded execution strategy as the optimal one.

Algorithm 3 gives the details, TC_{opt} and f_{opt} are to record the currently optimal checkpointing strategy and the processing frequency, respectively. Line 1 - 5 is to do the initialization, line 8 is step 1), line 9 - 11 is step 2), line 13 is step 3). And the while loop (line 9) will be broken if already adding the maximum number of checkpoints, which can be calculated as:

$$max\_chk = \frac{D - \sum_{i=1}^{m} c_i}{q} \tag{16}$$

The time complexity of the `CHK-C-RDE` algorithm is dominated by `while` loop (line 7-14). For line 8, that is, `FIND-OPT`, the complexity of which depends on how to choose ε, if we set ε to $x\%f_{max}$, the time complexity for this step is $O(m + max\_chk)$. And the amortized time cost of INSERT-CHK (line 13) is $O(m + max\_chk)$, as max_chk should be constant for a given application, then the total time complexity should be $O(m)$.

Algorithm 3 CHK-C-RDE(A, R_g, D, q, F)

1: **for** $i = 1$ *to* m **do**
2: $TC(i).index = i$; $TC(i).wcet = c_i$;
3: $TC(i).chk = 0$; $TC(i).inv = c_i$;
4: **end for**
5: $h = 0$; $TC_{\text{opt}} = TC$; $E_{\text{opt}} = \infty$;
6: calculate max_chk based on formula (16);
7: **while** $h \leq max\_chk$ **do**
8: (E, f, k) = FIND-OPT (TC, R_g, D, F);
9: **if** $E_{\text{opt}} > E$ **then**
10: $E_{\text{opt}} = E$;$f_{\text{opt}} = f$;$TC_{\text{opt}} = TC$;
11: **end if**
12: $h = h + 1$;
13: TC = INSERT-CHK (TC, q);
14: **end while**
15: **return** $TC_{\text{opt}}, f_{\text{opt}}, k$;

4 Optimal Application Execution Strategy under Discrete Frequence Scaling

In this section, we discuss the case that only a set of discrete processing frequencies, i.e., $\{f_1, f_2, ..., f_n\}$ with $f_i < f_j$ if $i < j$, are available for scaling. We call the problem defined in Section 2.2 D-RDE problem for short.

4.1 Properties of the Optimal Execution Strategy

Lemma 5. *Assume $F = \{f_1, \cdots, f_n\}$ are available frequencies with $f_i < f_j$ if $i < j$, given a checkpointing strategy, if the optimal processing strategy for C-RDE problem is $[(f_{\text{opt}}, t_{\text{opt}})]$, then for D-RDE problem, by applying the same checkpointing strategy, the optimal performance, i.e., the minimum energy consumption while meeting the reliability and deadline constraints can be obtained by:*

1. *using the frequency f_{opt} to process the whole application, if $\exists f_v \in F$* with *$f_v = f_{\text{opt}}$.*
2. *using the neighboring frequencies f_v and f_{v+1} to process the application, if $\exists \{f_v, f_{v+1}\} \subset F$* with *$f_v < f_{\text{opt}} < f_{v+1}$.*

The proof of lemma 5 will be given in the appendix section.

4.2 Checkpointing Based Application Execution Strategy

According to Lemma 5, when $f_v = f_{opt}$, just use the frequency f_v to execute the whole application. If f_{opt} is not available, we should use the neighboring frequencies f_v and f_{v+1}, but for how long they need to be executed, respectively, is still unknown. In order to minimize the disturbance for the application execution, we assume the frequency adjustment is only allowed at the end of the task segments. Then the problem turns how to assign the frequencies f_v and f_{v+1} to the task segments to achieve the minimum energy consumption while satisfying the deadline and reliability constraints.

Our proposed algorithm to do the frequency assignment ASSIGN-FRE is given in Algorithm 4. As both f_v and f_{v+1} are greater than f_{ee}, from energy saving perspective, the frequency of f_v should be used to execute the application as long as possible if reliability and deadline constraints are not violated. This is the basic idea of the Algorithm 4.

The details of Algorithm 4 are as follows. A_v and A_{v+1} are the stacks used to record the task segments to be executed under f_v and f_{v+1}, respectively. By default, all tasks' execution frequencies are set as f_{v+1} (line 1). Then for each possible number of tolerated faults, i.e., k', we move the longest task segment from A_{v+1} to A_v as long as reliability and deadline constraints are satisfied (line 4 - 17). The optimal assignment will be recorded in $\tilde{A}_{v+1}$ and $\tilde{A}_v$.

Having these in mind, Algorithm 5 is our proposed execution strategy for D-RDE problem. Line 1-2 is to use CHK-C-RDE algorithm to get the frequencies f_v and f_{v+1}. Line 3 is the frequencies assignment.

Algorithm 4 `ASSIGN-FRE (TC, $R_g$, $D$, $F'$)`

```
1: push all the task segments in TC to A_{v+1} by their lengths in ascending order
2: calculate the number of task segments N
3: Ã_{v+1} = A_{v+1}; Ã_v = A_v = Φ;
4: for k' = 0 to N do
5:    while |A_{v+1}| > 0 do
6:       if Reliability ≥ R_g and Deadline ≤ D then
7:          Ã_{v+1} = A_{v+1}; Ã_v = A_v;
8:          τ_ij = pop(A_{v+1});
9:          push(A_v, τ_ij);
10:      else
11:         break;
12:      end if
13:   end while
14:   if Reliability ≥ R_g and Deadline ≤ D then
15:      Ã_{v+1} = A_{v+1}; Ã_v = A_v;
16:   end if
17: end for
18: return Ã_v, Ã_{v+1}
```

Algorithm 5 `CHK-D-RDE ($A$, $R_g$, $D$, $F$)`

```
1: (TC_opt, f_opt, k) = CHK-C-RDE (A, R_g, D, q, F)
2: get F' = {f_v, f_{v+1}} satisfying f_v < f_opt < f_{v+1}
3: (A'_v, A'_{v+1}) = ASSIGN-FRE(TC_opt, R_g, D, F')
4: return A'_v, A'_{v+1}
```

5 Evaluation and Discussion

We first introduce two baseline algorithms and one definition.

- *LTF* (longest-task-first) [9]: Always select the task with longest WCET and allocate as much slack to it as possible.
- *SUEF* (slack usage efficiency factor) [9]: Always select the task with the largest ratio of the amount of energy saved to the required slack time.

Definition 2. *Tasks execution time's heterogeneity (TETH)*

$$TETH = \sqrt{\frac{max\{c_i | 1 \leq i \leq n\}}{min\{c_i | 1 \leq i \leq n\}}}$$

Where c_i is the WCET of task i.

In the following experiments, we set the average soft error rate $\lambda_0 = 10^{-6}$, $C_m = 3$ and $P_{ind} = 0.05P_d$, and $C_{ef} = 1$. The WCET of the tasks are randomly generated between the $min\{c_i\}$ and $max\{c_i\}$, where $min\{c_i\}$ is set to 20. As LTF and $SUEF$ only work when the reliability constraint is set to the reliability of executing the application under f_{max} without fault tolerance, to be fairness, which is also set as the reliability constraint R_g in our experiments. And we use q to indicate checkpointing cost and $utilization(\Gamma)$ to represent the utilization of the processor, which can be written as:

$$utilization(\Gamma) = \frac{\sum_{i=1}^{m} c_i}{D}$$

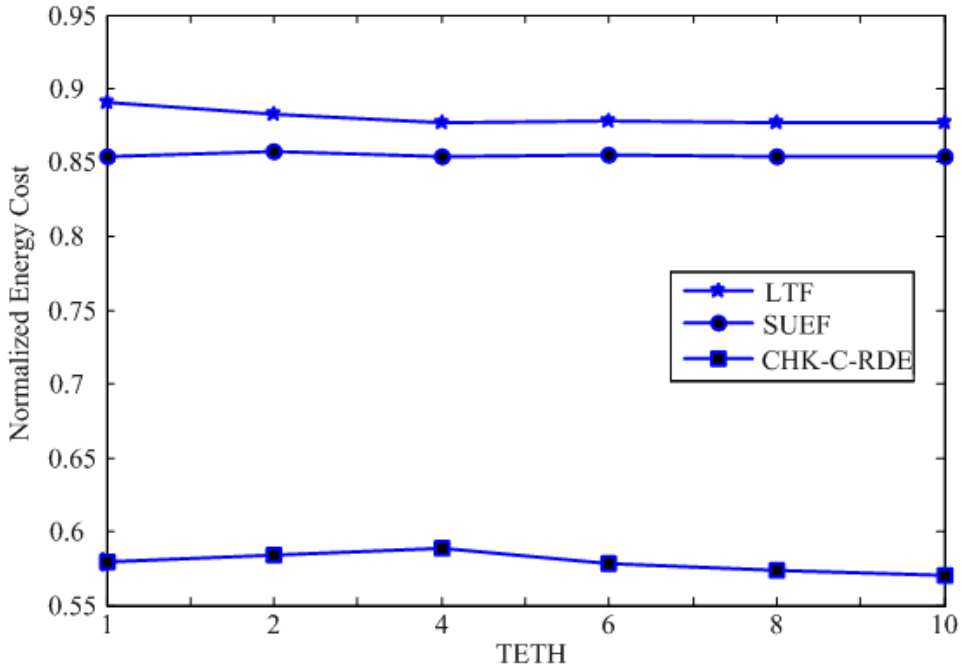

Figure 2: Impact of TETH ($utilization(\Gamma) = 0.7$, $q = 2$, $p = 3$)

We use normalized energy cost to evaluate the energy performance, which is defined as the energy cost normalized to the one of executing the application under f_{max}.

5.1 Simulation Results for the C-RDE problem

In this section, we will evaluate the performance of our proposed algorithms for C-RDE problem under different scenarios.

Fig. 2 evaluates the impact of task execution time variation. From which, we can see the energy performance of all the three algorithms are not sensitive to the task execution time variation. Among these algorithms, *CHK-C-RDE* always has the least energy consumption, which can save up to 30% more normalized energy than $SUEF$ and LTF.

Fig. 3 gives the utilization impact. LTF and $SUEF$ always consume more energy than *CHK-C-RDE*, while the gap becomes smaller when utilization becomes larger. The reason is, high utilization means less slack time, and hence, fewer checkpoints can be used to for energy saving. However, even under $utilization(\Gamma) = 0.9$, *CHK-C-RDE* still can save about 10% more normalized energy cost than the other two.

Fig. 4 evaluates the impact of checkpointing overhead on *CHK-C-RDE* algorithm, which is compared with the strategy without checkpointing, that is, the fault recovery will re-execute the whole task where fault occurs, we name it as *TRE-C-RDE* for short. In this experiment, the WCET of shortest task and longest one are 20 and 500, respectively, and the average WCET is 260. When $q = 2$, i.e., 0.7% of the average WCET, *CHK-C-RDE* algorithm saves 7% normalized energy more than *TRE-C-RDE*, when the q is increased to 8% of the average WCET ($q = 20$), the two algorithms have the same energy consumption.

Fig. 5 shows the impact of p, which is a system constant indicating the sensitivity of soft-error. We can see that the energy performance of *CHK-C-RDE* degrades slowly when p becomes larger. Even for the case $p = 5$, the advantage of our *CHK-C-RDE* is still obvious, which can save up to 25% more normalized energy consumption than LTF and $SUEF$.

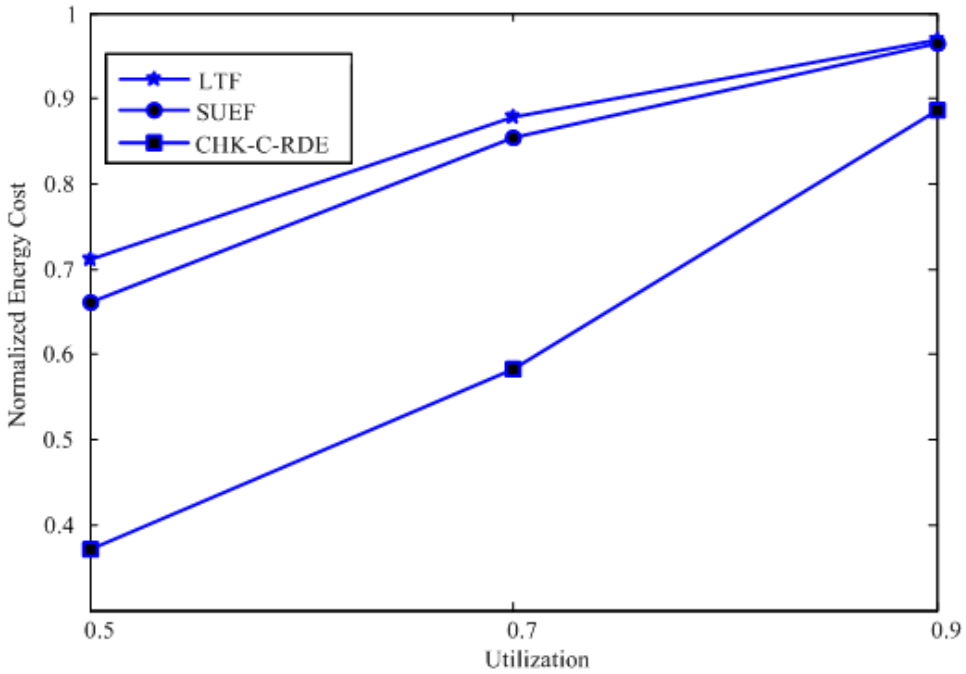

Figure 3: Impact of utilization ($TETH = 5$, $chk\_cost = 2$, $p = 3$)

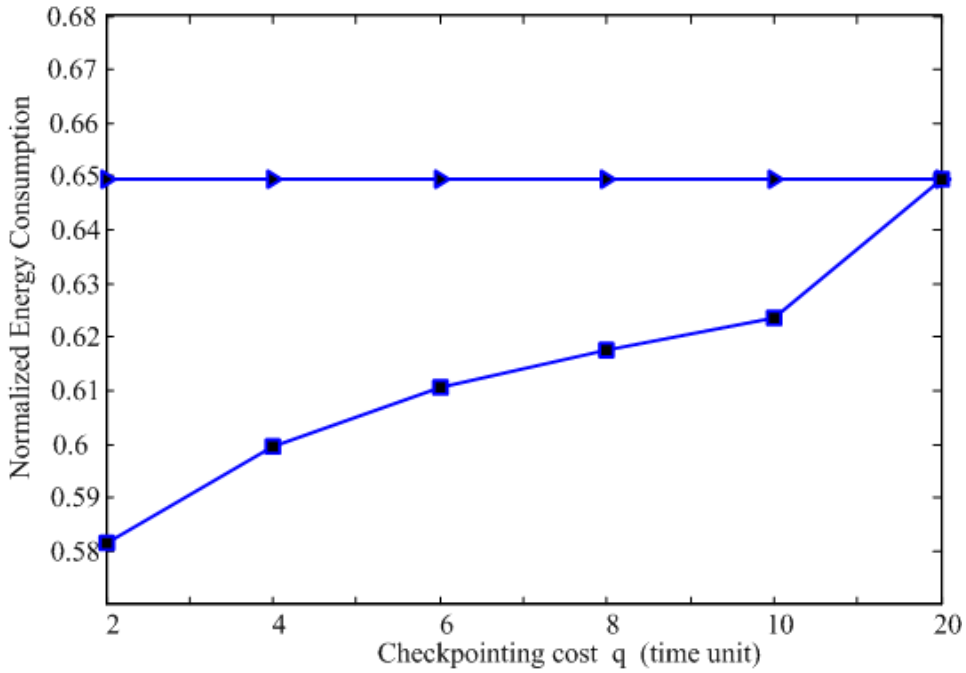

Figure 4: Impact of checkpointing cost ($TETH = 5$, $utilization(\Gamma) = 0.7$, $p = 3$)

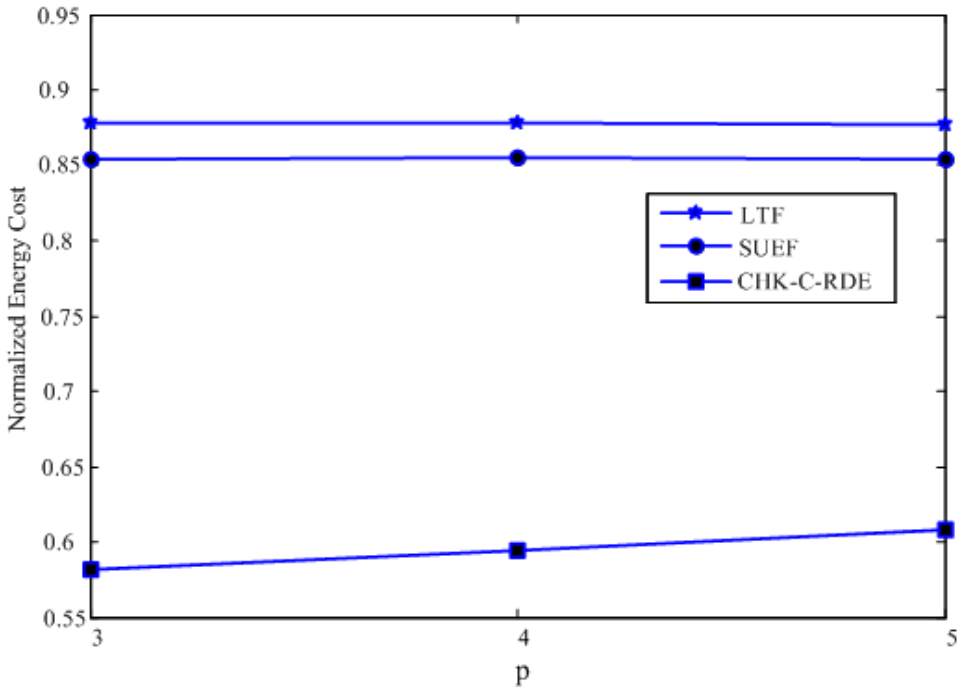

Figure 5: Impact of d ($TETH = 5$, $utilization(\Gamma) = 0.7$, $q = 2$)

5.2 Simulation Results for the D-RDE problem

We show the effectiveness of our proposed execution strategy for D-RDE problem in this section, the available discrete frequencies are set as $\{0.4, 0.6, 0.8, 1.0\}$.

Fig. 6 shows the effectiveness of the *CHK-D-RDE* algorithm, the energy consumption of which is closed to that of *CHK-C-RDE* (about 3% gap), and the utilization variation seems have no impact.

Fig. 7 investigates the impact of tasks execution time variation. When the $TETH$ increases form 1 to 6, the performance gaps between *CHK-C-RDE* and *CHK-D-RDE* are always around 2% to 5%.

6 Conclusion

In this paper, we have discussed the problem about how to improve the energy performance for a real-time application under given reliability and deadline constraints. We assume that fault arrival rate follows Poisson distribution, by focusing on checkpointing based fault recovery technique, we theoretically analyze the problem and derive some necessary properties about the optimal application execution strategy that has best performance with respect to energy saving and at the same time guarantees meeting the application's reliability and timing constraints. Based on these theoretical results, we also give our application execution strategies when the processor's frequency can be scaled continuously, or discretely. Comparing with other heuristics in literature, the experiment results show that our approaches outperform by as much as 30% with respective to energy saving.

The current work is based on uniprocessor. Extending the work to multi-processor environment will be our immediate next step.

References

[1] D. Ernst, S. Das, S. Lee, D. Blaauw, T. Austin, T. Mudge, N. S. Kim, and K. Flautner, "Razor: circuit-level correction of timing errors for low-power operation," *Micro, IEEE*, vol. 24, no. 6, pp. 10 –20, nov.-dec. 2004.

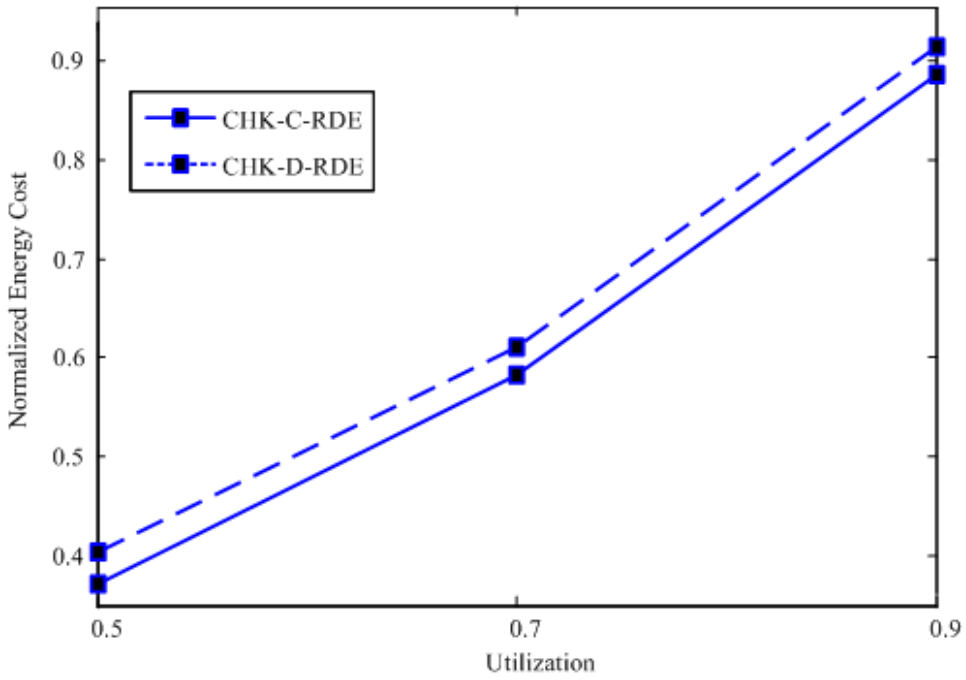

Figure 6: Impact of utilization for algorithms of C-RDE and D-RDE ($TETH = 5$, $q = 2$, $p = 3$)

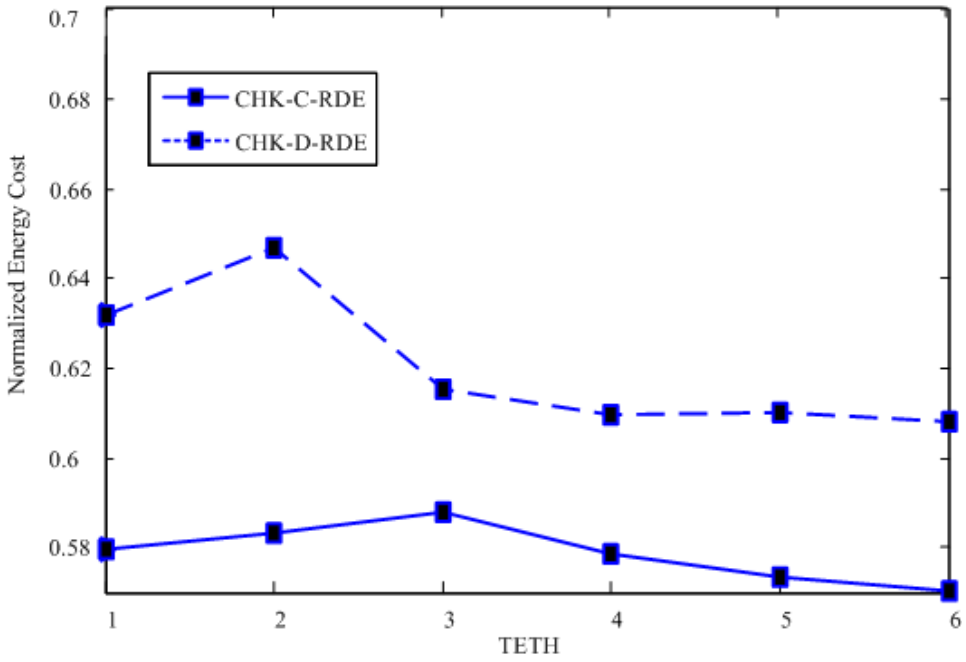

Figure 7: Impact of TETH for algorithms of C-RDE and D-RDE ($utilization(\Gamma) = 0.7$, $q = 2$, $p = 3$)

[2] R. K. Iyer, D. J. Rossetti, and M. C. Hsueh, "Measurement and modeling of computer reliability as affected by system activity," *ACM Trans. Comput. Syst.*, vol. 4, no. 3, pp. 214–237, Aug. 1986.

[3] F. Yao, A. Demers, and S. Shenker, "A scheduling model for reduced cpu energy," in *ANNUAL SYMPOSIUM ON FOUNDATIONS OF COMPUTER SCIENCE*. IEEE Computer Society, 1995, pp. 374–382.

[4] M. Weiser, B. Welch, A. Demers, and S. Shenker, "Scheduling for reduced cpu energy," in *Proceedings of the 1st USENIX conference on Operating Systems Design and Implementation*, ser. OSDI '94. Berkeley, CA, USA: USENIX Association, 1994.

[5] D. Zhu, "Reliability-aware dynamic energy management in dependable embedded real-time systems," in *Real-Time and Embedded Technology and Applications Symposium, 2006. Proceedings of the 12th IEEE*, april 2006, pp. 397 – 407.

[6] R. Jejurikar and R. Gupta, "Dynamic voltage scaling for systemwide energy minimization in real-time embedded systems," in *Low Power Electronics and Design, 2004. ISLPED '04. Proceedings of the 2004 International Symposium on*, aug. 2004, pp. 78 – 81.

[7] H. Aydin, R. Melhem, D. Mosse, and P. Mejia-Alvarez, "Power-aware scheduling for periodic real-time tasks," *Computers, IEEE Transactions on*, vol. 53, no. 5, pp. 584 – 600, may 2004.

[8] Y. Zhang and K. Chakrabarty, "Energy-aware adaptive checkpointing in embedded real-time systems," in *Design, Automation and Test in Europe Conference and Exhibition, 2003*, 2003, pp. 918 – 923.

[9] D. Zhu and H. Aydin, "Energy management for real-time embedded systems with reliability requirements," in *Computer-Aided Design, 2006. ICCAD '06. IEEE/ACM International Conference on*, nov. 2006, pp. 528 –534.

[10] B. Zhao, H. Aydin, and D. Zhu, "Enhanced reliability-aware power management through shared recovery technique," in *Proceedings of the 2009 International Conference on Computer-Aided Design*, ser. ICCAD '09. New York, NY, USA: ACM, 2009, pp. 63–70.

[11] ——, "Generalized reliability-oriented energy management for real-time embedded applications," in *Proceedings of the 48th Design Automation Conference*, ser. DAC '11. New York, NY, USA: ACM, 2011, pp. 381–386.

[12] D. K. Pradhan, Ed., *Fault-tolerant computing: theory and techniques; vol. 1*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1986.

[13] D. Zhu, "Reliability-aware dynamic energy management in dependable embedded real-time systems," *ACM Trans. Embed. Comput. Syst.*, vol. 10, no. 2, pp. 26:1–26:27, Jan. 2011.

[14] H. Aydin and D. Zhu, "Reliability-aware energy management for periodic real-time tasks," *Computers, IEEE Transactions on*, vol. 58, no. 10, pp. 1382 –1397, oct. 2009.

[15] K. Kim and J. 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, Y.-H. Lee, H.-N. Kim, J. Kim, Y. Park, L. Yang, and S. Kim, Eds., 2007, vol. 4523, pp. 560–571.

7 Appendix

In this section, we first show that $r(x, S_{\text{cp}}, k)(k \geq 0)$ is a decreasing function when $x > 0$ and then provide the proof for *Lemma 1*.

According to the definition of $r(x, S_{\text{cp}}, k)$, we have

$$r(x, S_{\text{cp}}, k) = \sum_{i=0}^{k} \frac{x^i e^{-x}}{i!} \cdot e^{-\lambda(f_{\max})(\sum_{j=1}^{i} len(j))}$$

If $k = 0$, then $r(x, S_{\text{cp}}, 0) = e^{-x}$, which is obviously decreasing for $x > 0$. For the scenario $k > 0$, as the S_{cp} is determined, then $len(j)$ is fixed and we can treat it as constant. If we define:

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

then we have $b(i+1) < b(i)$ and $0 < b(i) \leq 1$ when $i \geq 0$. Hence, the first derivation of $r(x, S_{\text{cp}}, k)$ is:

$$r'(x, S_{\text{cp}}, k) = (\sum_{i=0}^{k-1} \frac{x^i(b(i+1) - b(i))}{i!} - \frac{b(k)x^k}{k!}) \cdot e^{-x}$$

As $b(i+1) - b(i) < 0$ and $b(k) > 0$ when $x > 0$, we have $r'(x, S_{\text{cp}}, k) < 0$, which implies $r(x, S_{\text{cp}}, k)$ is decreasing when $x > 0$. These conclude the proof.

Lemma 1: Given an application $A = \{\tau_1, ..., \tau_m\}$, and its execution stragegy $S_{\text{app}} = (S_{\text{cp}}, S_{\text{ps}})$, where $S_{\text{cp}} = \{\vec{\mathcal{T}}; h\}$, $S_{\text{ps}} = (\overrightarrow{(f_i, t_i)}; n)$. If up to k faults are to be tolerated, then the application reliability is:

$$R_A(S_{\text{cp}}, S_{\text{ps}}, k) = \sum_{i=0}^{k} \frac{(\sum_{j=1}^{n} \lambda(f_j)t_j)^i e^{-\sum_{j=1}^{n} \lambda(f_j)t_j}}{i!} \cdot e^{-\lambda(f_{\max})(\sum_{j=1}^{i} len(j))}$$

and the energy consumption is:

$$E_A(S_{\text{cp}}, S_{\text{ps}}, k) = \sum_{i=1}^{n} (P_{\text{ind}} + C_{\text{ef}} f_i^{C_m}) t_i$$

Proof: According to the property of poisson distribution, when the fault arrival rate is $\lambda(f)$, the probability of at most k faults arriving in the time interval of $[0,\ t]$ can be expressed as[15]:

$$R(\lambda(f), t, k) = \sum_{i=0}^{k} \frac{(\lambda(f)t)^i e^{-\lambda(f)t}}{i!}$$

Then, assuming the checkpointing strategy S_{cp} is given, when k faults happen, in the worst case scenario, fault recoveries will take the duration of the longest k task segments. Having these in mind, we prove the lemma by induction.

Step 1): When $n = 1$, then $S_{\text{ps}} = [(f_1, t_1)]$, we have

$$R_A(S_{\text{cp}}, S_{\text{ps}}, k) = \sum_{i=0}^{k} \frac{(\lambda(f_1)t_1)^i e^{-\lambda(f_1)t_1}}{i!} \cdot e^{-\lambda(f_{\max})(\sum_{j=1}^{i} len(j))}$$

This is obviously true.

Step 2): Suppose $n = n_1 (> 1)$, we have

$$R_A(S_{\text{cp}}, S_{\text{ps}}, k) = \sum_{i=0}^{k} \frac{(\sum_{j=1}^{n_1} \lambda(f_j)t_j)^i e^{-\sum_{j=1}^{n_1} \lambda(f_j)t_j}}{i!} \cdot e^{-\lambda(f_{\max})(\sum_{j=1}^{i} len(j))}$$

Step 3): When $n = n_1 + 1$, then $S_{\text{ps}} = [(f_1, t_1), (f_2, t_2), \cdots, (f_{n_1+1}, t_{n_1+1})]$. If exactly i faults happen in the processing stage, there must be l $(0 \leq l \leq i)$ of them occur when the processing frequency is $f \in \{f_1, \cdots, f_{n_1}\}$ and the remaining $i - l$ faults happen when $f = f_{n_1+1}$, then the reliability of the application can be written as:

$$R_A(S_{\text{cp}}, S_{\text{ps}}, k) = \sum_{i=0}^{k} \sum_{l=0}^{i} \frac{(\sum_{j=1}^{n_1} \lambda(f_j)t_j)^l e^{-(\sum_{j=1}^{n_1} \lambda(f_j)t_j)}}{l!} \cdot \frac{(\lambda(f_{n_1+1})t_{n_1+1})^{(i-l)} e^{-\lambda(f_{n_1+1})t_{n_1+1}}}{(i-l)!} \cdot e^{-\lambda(f_{\max})(\sum_{j=1}^{i} len(j))}$$

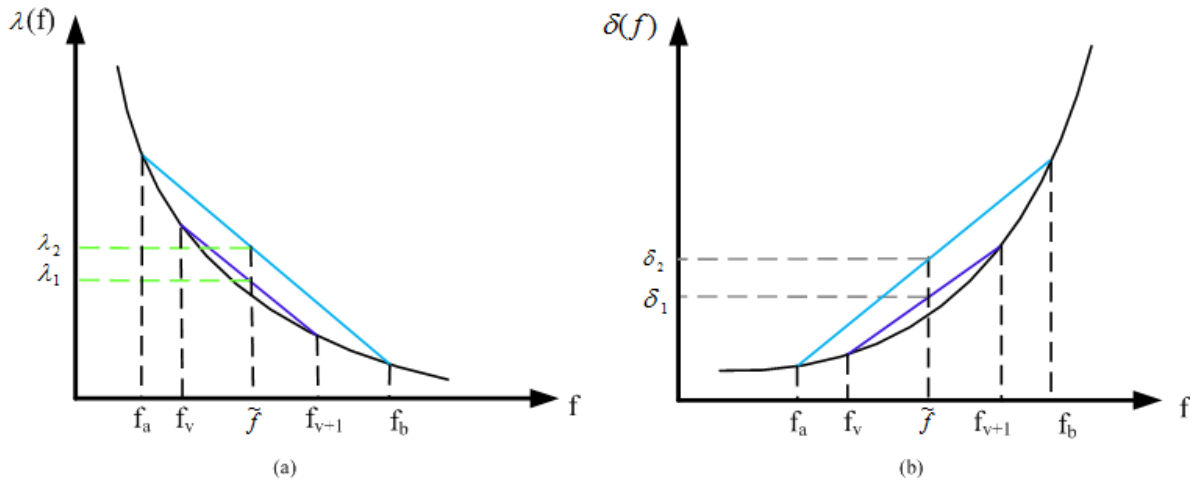

Figure 8: convex function: $\lambda(f)$ and $\delta(f)$

According to binomial formula, that is:

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

we have:

$$R_A(S_{\text{cp}}, S_{\text{ps}}, k) = \sum_{i=0}^{k} \frac{(\sum_{j=1}^{n} \lambda(f_j)t_j)^i e^{-\sum_{j=1}^{n} \lambda(f_j)t_j}}{i!} \cdot e^{-\lambda(f_{\max})(\sum_{j=1}^{i} len(j))}$$

For the energy consumption, as the probability of fault occurrence is relatively small, the expected energy consumption for recovery stage is ignorable comparing to the energy consumption for processing stage. Hence, the expected energy consumption of the application can be approximately defined as:

$$E_A(S_{\text{cp}}, S_{\text{ps}}, k) = \sum_{i=1}^{n} (P_{\text{ind}} + C_{\text{ef}} {f_i}^{C_m}) t_i$$

These conclude the proof.

Lemma 5: Assume $F = \{f_1, \cdots, f_n\}$ are available frequencies with $f_i < f_j$ if $i < j$, given a checkpointing strategy, if the optimal processing strategy for C-RDE problem is $[(f_{\text{opt}}, t_{\text{opt}})]$, then for D-RDE problem, by applying the same checkpointing strategy, the optimal performance, i.e., the minimum energy consumption while meeting the reliability and deadline constraints can be obtained by:

1. using the frequency f_{opt} to process the whole application, if $\exists f_v \in F$ with $f_v = f_{\text{opt}}$.
2. using the neighboring frequencies f_v and f_{v+1} to process the application, if $\exists \{f_v, f_{v+1}\} \subset F$ with $f_v < f_{\text{opt}} < f_{v+1}$.

Proof: It is trivial for case 1), we only need to give the proof for case 2).

For case 2), we assume the optimal processing strategy is $S_{\text{ps}} = [(f_v, t_v), (f_{v+1}, t_{v+1})]$ when the chosen processing frequencies are f_v and f_{v+1}, and up to k faults can be tolerated under this strategy. Then, we need to prove this strategy S_{ps} consumes less energy than any other valid processing strategy, i.e., executing the application under which can meet the reliability and deadline constraints. As one, two or multiple (three or even more) processing frequencies can be chosen in the processing strategy, hence, we need to prove S_{ps} is optimal among all these three scenarios.

A) $S_{\text{ps}} = [(f_v, t_v), (f_{v+1}, t_{v+1})]$ consumes less energy than any processing strategy under uniform processing frequency.

For any processing strategy with uniform processing frequency f_j, as $f_{\text{opt}} > f_v$, and all the processing strategies need to finish the same workload, i.e., $f_{opt}t_{opt} = f_j t_j$, as $[(f_{opt}, t_{opt})]$ is the optimal processing strategy when frequency can be scaled continuously, hence, we have $f_j \geq f_{v+1}$. As all the f_v, f_{v+1} and f_j are higher than f_{ee}, executing the tasks under higher frequency consumes more energy, so the processing strategy under uniform frequency f_j costs more energy.

B) $S_{\text{ps}} = [(f_v, t_v), (f_{v+1}, t_{v+1})]$ consumes less energy than any other processing strategy using the combination of two processing frequencies except f_v and f_{v+1}.

Let $S'_{\text{ps}} = ([(f_a, t_a), (f_b, t_b)])$ denotes any other valid processing strategy using two frequencies except $[(f_v, t_v), (f_{v+1}, t_{v+1})]$, and k' is the number of faults can be tolerated at most under S'_{ps}. Then the possible combinations of f_a and f_b should fall into the following two categories:

B.1) $f_a \geq f_{v+1}, f_b \geq f_{v+1}$

B.2) $f_a \leq f_v, f_b > f_{v+1}$ or $f_a < f_v, f_b \geq f_{v+1}$

For B.1), as both f_a and f_b are no lower than f_{v+1}, therefore, S'_{ps} consumes more energy than the processing strategy as $([(f_v, t_v), (f_{v+1}, t_{v+1})])$.

For B.2), assume the checkpointing strategy is S_{cp}, then the reliability and energy consumption under processing strategy S_{ps} can be expressed as $R_A(S_{\text{cp}}, S_{\text{ps}}, k) = r(\lambda(f_v)t_v + \lambda(f_{v+1})t_{v+1}, S_{\text{cp}}, k)$ and $E_A(S_{\text{cp}}, S_{\text{ps}}, k) = \delta(f_v)t_v + \delta(f_{v+1})t_{v+1}$, respectively. Similarly, we have $R_A(S_{\text{cp}}, S'_{\text{ps}}, k') = r(\lambda(f_a)t_a + \lambda(f_b)t_b, S_{\text{cp}}, k')$, $E_A(S_{\text{cp}}, S'_{\text{ps}}, k') = \delta(f_a)t_a + \delta(f_b)t_b$, respectively. Now we need to prove $E_A(S_{\text{cp}}, S'_{\text{ps}}, k') > E_A(S_{\text{cp}}, S_{\text{ps}}, k)$.

Considering another processing strategy $S''_{\text{ps}} = [(f_v, t'_v), (f_{v+1}, t'_{v+1})]$ with $t'_v + t'_{v+1} = t_a + t_b$ and $f_v t'_v + f_{v+1}t'_{v+1} = f_a t_a + f_b t_b$, then we know at most k' faults can be tolerated and $R_A(S_{\text{cp}}, S''_{\text{ps}}, k') = r(\lambda(f_v)t'_v + \lambda(f_{v+1})t'_{v+1}, S_{\text{cp}}, k')$. By defining $\tilde{t} = t_a + t_b$ and $\tilde{f} = \frac{f_a t_a + f_b t_b}{\tilde{t}}$, we have $f_v t'_v + f_{v+1}t'_{v+1} = f_a t_a + f_b t_b = \tilde{t}\tilde{f}$. As $\lambda(f)$ is decreasing and convex, by denoting $\lambda_1 = \lambda(f_v)\frac{t'_v}{\tilde{t}} + \lambda(f_{v+1})(\frac{t'_{v+1}}{\tilde{t}})$ and $\lambda_2 = \lambda(f_a)\frac{t_a}{\tilde{t}} + \lambda(f_b)(\frac{t_b}{\tilde{t}})$, we have $\lambda_1 < \lambda_2$ (Fig. 8(a)). As $r(x, S_{\text{cp}}, k')$ is decreasing for given S_{cp} and k' when $x > 0$, we have $r(\tilde{t}\lambda_1, S_{\text{cp}}, k') > r(\tilde{t}\lambda_2, S_{\text{cp}}, k')$, which implies $R_A[S_{\text{cp}}, S''_{\text{ps}}, k'] > R_A(S_{\text{cp}}, S'_{\text{ps}}, k')$. As S'_{ps} is valid, and the processing strategy S''_{ps} costs the same time unit and has higher reliability than S'_{ps}, hence, which is also valid.

For the energy consumption, as $\delta(f) = P_{ind} + C_{ef} f^{C_m} (C_m \geq 2)$ is increasing and convex for $f > 0$, as shown in Fig. 8(b), by setting $\delta_1 = \delta(f_v)\frac{t'_v}{\tilde{t}} + \delta(f_{v+1})\frac{t'_{v+1}}{\tilde{t}}$ and $\delta_2 = \delta(f_a)\frac{t_a}{\tilde{t}} + \delta(f_b)\frac{t_b}{\tilde{t}}$, where $\tilde{t} = t_a + t_b = t'_v + t'_{v+1}$, we have $\delta_1 < \delta_2$, then $\delta(f_v)t'_v + \delta(f_{v+1})t'_{v+1} < \delta(f_a)t_a + \delta(f_b)t_b$, which implies $E_A(S_{\text{cp}}, S''_{\text{ps}}, k') < E_A(S_{\text{cp}}, S'_{\text{ps}}, k')$. As S_{ps} is the optimal processing strategy if frequencies f_v and f_{v+1} are chosen, then we have $E_A(S_{\text{cp}}, S_{\text{ps}}, k) \leq E_A(S_{\text{cp}}, S''_{\text{ps}}, k')$, hence, we have $E_A(S_{\text{cp}}, S_{\text{ps}}, k) < E_A(S_{\text{cp}}, S'_{\text{ps}}, k')$.

C) The processing strategy S_{ps} consumes less energy than any valid one with multiple (three or even more) processing frequencies.

Suppose $PR''' = [(f_c, t_c), (f_{c+1}, t_{c+1}), \cdots, (f_d, t_d)]$ with $f_d > ... > f_{c+1} > f_c$ and $d - c + 1 \geq 3$ is a valid processing strategy, and up to k'' faults can be tolerated. Then either

C.1) $f_c \geq f_{v+1}$

or,

C.2) $f_c \leq f_v$ and $f_d \geq f_{v+1}$

For C.1), all the processing frequencies in S'''_{ps} is no lower than f_{v+1}, which implies the energy consumption of which is higher than that under S_{ps}.

For C.2), assume the processing strategy is $[(f_c, t_c), ..., (f_j, t_j), (f_{j+1}, t_{j+1}), (f_d, t_d)]$ with $f_j \leq f_v$ and $f_{j+1} \geq f_{v+1}$, then according to lemma 2, the processing strategy $([(f'_c, \sum_{i=c}^{j} t_i), (f'_d, \sum_{i=j+1}^{d} t_i)])$ with $f'_c(\sum_{i=c}^{j} t_i) = \sum_{i=1}^{j} f_i t_i$ and $f'_d(\sum_{i=j+1}^{d} t_i) = \sum_{i=j+1}^{d} f_i t_i$ has higher reliability and less energy consumption.

Based on proof in B), the processing strategy $[(f_v, t_v), (f_{v+1}, t_{v+1})]$ is the optimal one when two frequencies are chosen, which means it costs less energy than PR''', hence, we get the conclusion.

Finally, A), B) and C) conclude the proof for Lemma 5.