Energy Management for Real-Time Embedded Systems with Reliability Requirements

Dakai Zhu
University of Texas at San Antonio
6900 N. Loop 1604 West
San Antonio, TX 78249
dzhu@cs.utsa.edu

Hakan Aydin
George Mason University
4400 University Drive
Fairfax, VA 22030
aydin@cs.gmu.edu

ABSTRACT

With the continued scaling of CMOS technologies and reduced design margins, the reliability concerns induced by transient faults have become prominent. Moreover, the popular energy management technique *dynamic voltage and frequency scaling (DVFS)* has been shown to have direct and negative effects on reliability. In this work, for a set of real-time tasks, we focus on the slack allocation problem to minimize their energy consumption while preserving the overall system reliability. Building on our previous findings for a single real-time application where a recovery task was used to preserve reliability, we identify the problem of reliability-aware energy management for multiple tasks as NP-hard and propose two polynomial-time heuristic schemes. We also investigate the effects of on-chip/off-chip workload decomposition on energy management, by considering a generalized power model. Simulation results show that ordinary energy management schemes could lead to drastically decreased system reliability, while the proposed reliability-aware heuristic schemes are able to preserve the system reliability and obtain significant energy savings at the same time.

1. INTRODUCTION

The phenomenal improvements in the performance of computing systems caused a drastic increase in power densities. For battery-operated embedded systems, energy has been promoted to be a first-class system resource [24] and energy-aware system design has recently become an important research area. The most common strategy to achieve energy savings is to run the system components at low-performance (thus, low-power) operation points, whenever possible. For instance, as a widely popular technique, *dynamic voltage and frequency scaling (DVFS)* scales down the CPU frequency and supply voltage simultaneously to save energy [23].

While most of the early Real-Time DVFS research focused on the *dynamic* and *on-chip/CPU* energy consumption, there is a growing awareness about the need for more comprehensive *system-wide* energy management frameworks [12, 16, 28]. Along the same lines, when one considers the *off-chip* components such as main memory and I/O devices, the assumption that the task execution time increases *linearly* with the decrease in frequency remains *no longer valid*. Recently, several attempts have been made to capture both on-chip and off-chip workload characteristics of tasks in energy management [1, 5, 6].

Reliability and fault tolerance have always been major concerns in computer system design. Due to the effects of hardware defects, electromagnetic interference or cosmic ray radiations, faults may occur at run-time, especially in systems deployed in dynamic environments. With continued scaling of CMOS technologies and adjustment of design margins for higher performance, it is expected that, in addition to the systems that are traditionally operated in electronics hostile environments (such as those in outer space), practically *all* digital systems will be much more vulnerable to the transient faults [11, 22]. Moreover, blindly applying DVFS for energy savings may cause significant degradation in system's reliability as it has been shown that DVFS has a direct and negative effect on transient fault rates [8, 28]. Therefore, for real-time embedded systems where reliability is as important as energy efficiency, *reliability-cognizant energy management* becomes a necessity.

Closely related work: For the primary/backup recovery model, Unsal *et al.* proposed to postpone the execution of backup tasks to minimize the overlap of primary and backup execution and thus the energy consumption [21]. The optimal number of checkpoints, evenly or unevenly distributed, to minimize energy consumption while tolerating one transient fault was explored by Melhem *et al.* in [17]. Elnozahy *et al.* proposed an *Optimistic TMR* scheme that reduces the energy consumption for traditional TMR systems by allowing one processing unit to slow down provided that it can catch up and finish the computation before the application deadline if the results from other two units are not in agreement [10]. The optimal frequency settings for OTMR were further explored in [29]. Assuming a Poisson fault model, Zhang *et al.* proposed an adaptive checkpointing scheme that dynamically adjusts checkpoint intervals for energy savings while tolerating a fixed number of faults for a single task [25]. The work is further extended to a set of periodic tasks [26].

Most of the previous research either focused on tolerating fixed number of faults [10, 17] or assumed constant fault rate [25, 26] when applying DVFS for energy savings. In our previous work, the effects of DVFS on transient fault rates have been studied and an exponential fault rate model was proposed in [28]. Using this fault rate model, Ejlali *et al.* studied the reliability - energy savings trade-offs in [9]. As an initial study, based on the single task model, we have proposed a *reliability-aware energy management* scheme, where an additional *recovery task* is scheduled for the task to recuperate the reliability loss due to DVFS [27]. Though important on its own, a fundamental limitation of [27] is that, the decisions are made by considering only one task at a time.

In this paper, we focus on the system design problem of determining the processing speeds (and supply voltages) for a set of real-time tasks, which may have different on-chip/off-chip workloads, to save energy while preserving the system reliability. As we show later in this paper, significant gains can be obtained in terms of both energy *and* reliability, if we consider *all* the tasks at the same time when allocating slack for energy and reliability management.

The remainder of this paper is organized as follows. The system models and assumptions are presented in Section 2. In Section 3, we first formulate the problem to be solved; then we study the effects of workload decomposition on system-wide energy management. After reviewing the concept of reliability-aware energy management and identifying the intractability of the problem considered, we propose two heuristic schemes. Simulation results are presented in Section 4. Section 5 concludes the paper.

2. SYSTEM MODELS AND ASSUMPTIONS

2.1 Application Model

We consider a real-time application that consists of a set of n independent tasks: $T_1, \ldots, T_n$. All tasks in the application should complete their executions by the deadline D. Note that, if the application is periodic, D can also represent the *period*. The worst-case execution time (WCET) of task T_i under the maximum CPU frequency (f_{max}) is denoted by c_i. We consider a system with DVFS capability where the clock frequency values are normalized with respect to f_{max}. In other words, we take $f_{max} = 1.0$.

2.2 Power Model

We adopt the system-level power model proposed in [28, 29], where the power consumption P in a system is given by:

$$P = P_s + \hbar(P_{ind} + P_d) = P_s + \hbar(P_{ind} + C_{ef} f^m) \qquad (1)$$

Here P_s is the *static power*, P_{ind} is the *frequency-independent active power* and P_d is the *frequency-dependent active power*. The static power, which may be removed only by powering off the whole system, includes (but not limited to) the power to maintain basic circuits, keep the clock running and the memory in sleep modes [12]. P_{ind} is a constant and corresponds to the power that is independent of processing frequencies (and supply voltages) but can be efficiently removed by putting systems into sleep states [7, 12]. P_d includes processor's dynamic power as well as any power that depends on processing frequencies (and supply voltages) [3].

$\hbar$ represents system states and indicates whether active powers are currently consumed in the system. Specifically, when the system is *active* (defined as having computation in progress) $\hbar = 1$; otherwise, the system is in sleep modes or turned off and $\hbar = 0$. The effective switching capacitance C_{ef} and the dynamic power exponent m (which is, in general, no smaller than 2) are system dependent constants and f is the processing frequency.

Despite its simplicity, the above model captures the essential components of power consumption in embedded systems for system-level energy management. Intuitively, lower frequencies result in less frequency-dependent active energy consumption. But with reduced frequency, tasks run longer and thus consume more static and frequency-independent active energy. Considering the prohibitive overhead of turning on/off a device [2], for the time interval considered (e.g., within the deadline), we assume that the system is always on (but some components may be put into sleep states for energy savings) and P_s is always consumed.

2.3 Fault Model

During the execution of an application, a fault may occur due to various reasons, such as hardware defects, software errors and the effects of cosmic ray radiations. Since *transient* faults occur much more frequently than *permanent* faults [4, 15], especially with the continued scaling of CMOS technologies and adjustment of design margins [11, 22], in this paper, we focus on transient faults, and explore *backward recovery* techniques to tolerate them. It is assumed that transient faults are detected using *sanity* or *consistency* checks [18] at the end of task's execution and the time overhead of fault detection is incorporated into task's WCET. The recovery is assumed to take place through the re-execution of the task [18].

Based on the observation that *soft error rate (SER)* increases with lower supply voltages due to the reduced *critical charge* (which is the smallest charge needed to cause a soft error) [14, 19, 30], we studied the negative effects of DVFS on transient fault rates in [28]. With the assumption that the radiation-induced transient faults follow a Poisson distribution [25, 26], for systems running at frequency f (and corresponding supply voltage V), the average transient fault rate λ is modeled as [28]:

$$\lambda(f) = \lambda_0 \cdot g(f) \qquad (2)$$

where λ_0 is the average fault rate corresponding to f_{max} (and corresponding supply voltage V_{max}). That is, $g(f_{max}) = 1$. With scaled processing frequencies (and supply voltages), the fault rate generally increases and $g(f) > 1$ for $f < f_{max}$.

Moreover, considering the relationship between soft error rates, critical charge, supply voltage and the number of particles in the cosmic rays [14, 19, 30], we proposed an exponential fault rate model: $g(f) = \lambda_0 10^{\frac{d(1-f)}{1-f_{min}}}$ where the exponent d (> 0) is a constant, indicating the sensitivity of fault rates to DVFS. That is, reducing the supply voltage and frequency for energy savings results in *exponentially* increased fault rates. The maximum average fault rate is assumed to be $\lambda_{max} = \lambda_0 10^d$, which corresponds to the lowest frequency f_{min} (and the supply voltage V_{min}).

3. SAVING ENERGY WHILE PRESERVING RELIABILITY

3.1 Problem Formulation

While DVFS is a powerful technique to save energy, the consideration of transient faults, and in general, reliability concerns, introduces new dimensions to the problem. Let us denote the *reliability* of task T_i by R_i^0, which is the probability of correctly completing T_i with its WCET at f_{max}. From the Poisson fault arrival pattern and the average fault rate λ_0, we have $R_i^0 = e^{-\lambda_0 c_i}$. As the system reliability R_0 depends on the correct execution of *all* the tasks, we obtain $R_0 = \prod_{i=1}^{n} R_i^0$. Without loss of generality, it is assumed that R_0 is *satisfactory*. However, when DVFS is used to save energy, the reliability of the tasks executed at the reduced frequency will be adversely affected, due to the both extended execution time **and** increased fault rates at lower frequencies/supply voltages.

Suppose that the amount of available slack is $S = D - \sum_{i=1}^{n} c_i$. In this work, we focus on the problem of **allocating the slack S to individual tasks for maximizing energy savings without sacrificing system reliability, while taking the effects of voltage scaling on fault rates into consideration**. In order to preserve R_0, for simplicity, we adopt conservative approaches that maintain the reliability of *each and every* task. That is, the schemes will guarantee that the probability of task T_i being correctly executed will be no less than R_i^0 ($i = 1, \cdots, n$), *even after energy management.*

Suppose that the amount of slack allocated to task T_i is s_i. In addition to being used to scale down the execution of T_i to save energy, the slack can also provide temporal redundancy to enhance T_i's reliability. Therefore, the problem can be formally stated as:

$$\text{minimize} \quad \sum_{i=1}^{n} E_i \tag{3}$$

$$\text{subject to} \quad \sum_{i=1}^{n} s_i \leq S \tag{4}$$

$$s_i \geq 0, i = 0, \ldots n \tag{5}$$

$$R_i \geq R_i^0, i = 0, \ldots n \tag{6}$$

where E_i and R_i are the energy consumption and the reliability achieved, respectively, after allocating s_i to task T_i, for both energy and reliability management. Here, the inequality (4) ensures that the total allocated slack does not exceed S, the inequality (5) states that slack allocation cannot be negative, and the inequality (6) guarantees the reliability of each task is at least equal to its original value (i.e. without energy management).

3.2 Task-Specific Energy-Efficient Frequency

Note that one "hidden" aspect of our problem involves determining the CPU frequency for the tasks that are selected for energy and reliability management. Before discussing the details of our solutions, we would like to address the issues with *task specific energy-efficient frequencies*, for the generic power and workload models.

Most of the previous research has assumed that the execution time scales linearly with the processing frequency. However, recent research has found that such assumptions may not be accurate, especially in embedded systems with limited cache sizes [20]. Considering that the off-chip access latencies, specifically the frequencies of memory and I/O buses, are mostly independent of the CPU clock frequency, a more accurate execution model based on the *on-chip* and *off-chip* workload decomposition, has been proposed [1, 5, 6]. In this model, the worst-case execution time c_i of a real-time task T_i at f_{max} can be expressed as

$$c_i = x_i + y_i \tag{7}$$

where x_i is the *frequency-dependent* component (due to on-chip execution) and y_i is the *frequency-independent* component (due to off-chip accesses). With the assumption of $f_{max} = 1$, the scaled execution time, $t_i(f)$, of task T_i at frequency f will be:

$$t_i(f) = \frac{x_i}{f} + y_i \tag{8}$$

Moreover, as tasks may perform different off-chip accesses (e.g. for different I/O devices) during the execution, P_{ind} may vary from task to task. Suppose that the frequency-independent power for task T_i is $P_{ind,i}$. Similarly, the effective switching capacitance $C_{ef,i}$ may also be different for each task T_i. Consequently, the total energy consumption of task T_i at frequency f will depend on task characteristics and it can be modeled as:

$$E_i = P_s \cdot D_i + (P_{ind,i} + C_{ef,i} f^m) \cdot t_i(f) \tag{9}$$

where D_i is the time period allocated to task T_i.

From Equations (7), (8) and (9), the energy consumption for executing task T_i will be minimized when the processing frequency f satisfies the following equation:

$$m \cdot C_{ef,i} \cdot y_i \cdot f^{m+1} + (m-1)C_{ef,i} \cdot x_i \cdot f^m - P_{ind,i} x_i = 0 \tag{10}$$

Therefore, by solving Equation (10), we can get a *task-specific energy-efficient frequency* for each task. Note that for $m = 2$ or $m = 3$, Equation (10) will yield *cubic* or *quartic* equations, which can be solved analytically. For other cases, observe that the left-hand side of Equation (10) represents a convex function, which is strictly increasing in the interval $[0, f_{max} = 1]$. Consequently, a binary search technique can be used to converge rapidly to the task-specific energy-efficient frequency. For the special case where no off-chip workload is considered (i.e., $y_i = 0$), the task-specific energy-efficient frequency $f_{ee,i}$ for task T_i can be obtained as a close formula:

$$f_{ee,i} = \sqrt[m]{\frac{P_{ind,i}}{(m-1)C_{ef,i}}} \tag{11}$$

From the above equation, we can see that, as $P_{ind,i}$ increases (i.e., as the frequency-independent power becomes more dominant) and $C_{ef,i}$ decreases (i.e., the frequency-dependent power becomes less important), the task-specific energy efficient frequency becomes higher, which implies that less slack could be used for scaling down the task T_i and less energy may be saved.

Intuitively, as the off-chip workload does not scale with reduced processing speeds, the total energy consumption of a task due to the off-chip workload will decrease monotonically when the processing speed decreases. Therefore, tasks with high off-chip workloads favor lower processing speeds and, in general, have a lower task-specific energy-efficient frequency.

3.3 Using Recovery Tasks with DVFS to Preserve/Improve Reliability

As a motivational example, consider Figure 1, where 3 units of slack S is allocated to the task T_k, which has the WCET as $c_k = 2$. Without considering reliability (and assuming that $y_k = 0$), the ordinary power management scheme would use all the slack to scale down the processing speed of task T_k to 0.4 for energy savings as shown in Figure 1b. However, by doing so, the probability of having *at least* one transient fault during the execution of T_k increases drastically[1] due to both extended execution time and exponentially increased fault rates at lower frequencies and supply voltages [28].

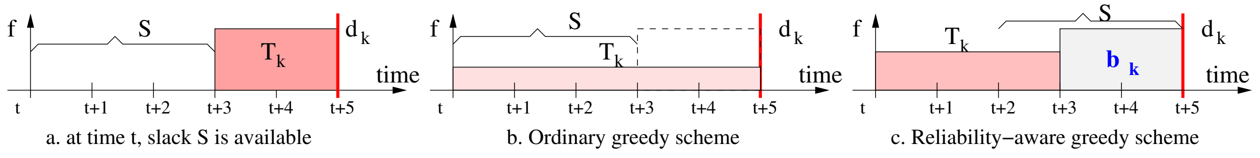
Instead of using all the slack for DVFS and energy management, one can reserve a portion of the slack to schedule one *recovery task* b_k (in the form of re-execution) for task T_k, to recuperate the reliability loss due to energy management (see Figure 1c) [27]. The remaining slack can still be used to scale down the processing of task T_k to save energy.

Note that the recovery task b_k will only be executed if a fault is detected at the end of T_k's execution. With b_k, the overall *reliability* R_k of task T_k will be the summation of the probability of primary task T_k being executed correctly <u>and</u> the probability of having transient fault(s) during T_k's execution while the recovery task b_k being executed correctly. Notice that, if the execution of the primary task T_k is faulty, the recovery task b_k will be executed at f_{max} and the probability of having no faults during its execution is $e^{-\lambda_0 c_k} = R_k^0$. Therefore, considering the recovery task b_k, the probability, R_k, of finishing task T_k correctly in time will be [27]:

$$R_k = e^{-\lambda(f_k)s_i} + \left(1 - e^{-\lambda(f_k)s_i}\right) R_k^0 > R_k^0 \tag{12}$$

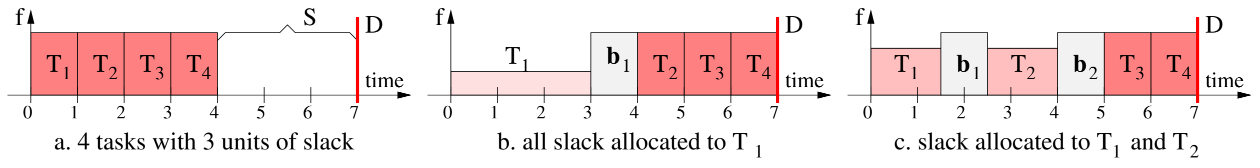
where f_k is the reduced clock frequency for T_k, $\lambda(f_k)$ is the corresponding fault rate and s_i is the slack allocated to T_k (part of which is reserved for recovery). That is, **whenever the available slack is larger than the task's WCET, by scheduling a recovery task, one can preserve the reliability of a real-time task while**

[1]In fact, such a slow-down may result in a reliability degradation of at least two orders of magnitude [27].



f
S
T_k
d_k
time
t
t+1
t+2
t+3
t+4
t+5
a. at time t, slack S is available
b. Ordinary greedy scheme
c. Reliability−aware greedy scheme
b_k

Figure 1: Ordinary and Reliability-Aware Energy Management [27].



f
T1 T2 T3 T4
S
D
time
0 1 2 3 4 5 6 7
a. 4 tasks with 3 units of slack
b1
b. all slack allocated to T1
b2
c. slack allocated to T1 and T2

Figure 2: Different slack allocations for multiple tasks.

still saving energy regardless of the exponent d in the fault rate model and the reduced processing frequency f_k [27]. Although checkpoints could be used to more efficiently use the slack time [17, 25, 26], it has been shown that checkpoints with one recovery section *cannot* guarantee to preserve task's reliability [27]. For simplicity, in this paper, we will focus on the re-execution of full tasks during recovery.

3.4 The Case of Multiple Tasks

The problem gains new dimensions when we consider a real-time application consisting of multiple tasks. In particular, we need to allocate available slack to multiple tasks, *possibly in different amounts*, to maximize the energy savings. Moreover, from the above discussion, whenever a task is scaled down for saving energy, a recovery task needs to be scheduled to preserve its reliability. This, in turn, will reduce the available slack for DVFS. All these considerations give rise to an interesting trade-off dimension.

In general, the problem of reliability-aware energy management for multiple tasks can be divided into two sub-problems: **selecting a subset of tasks to be managed (the remaining tasks are left intact and will run at the maximum frequency)** and **determining the processing speed for each task within the selected subset**.

In what follows, we first illustrate how different task selection decisions can yield different amounts of energy savings. Then, we identify the problem of finding the optimal slack allocation to multiple tasks for maximizing energy savings while preserving the system reliability as NP-hard. Consider the example given in Figure 2. We have four tasks, each having WCET of 1 time unit, that need to be completed by time $D = 7$. Therefore, there are 3 units of slack available in the system. For illustration purposes, here we assume that the execution time increases linearly with decreasing frequency (i.e., the off-chip workload is not considered), and that the power consumption is given by a cubic function.

Since the available slack is not sufficient to accommodate a separate recovery for each task, only a subset of tasks can be managed. If we decide to manage three tasks, we will need a total of 3 units of slack for recovery tasks. Consequently, no slack will be left for energy management and no energy savings can be obtained. On the other hand, if only one task (e.g., task T_1) is chosen, as shown in Figure 2b, we can schedule the recovery task b_1 for task T_1 and then use the remaining 2 units of slack to scale down the processing speed of T_1 to $\frac{1}{3}$ for energy savings. Simple algebra shows that, the energy savings would be $\frac{8}{9}E$, where E is the energy consumed by task T_1 without any power management. As explained earlier, the original reliability of T_1 would be preserved with the help of the recovery task b_1.

However, as shown in Figure 2c, if two tasks T_1 and T_2 are selected, after scheduling the recovery tasks b_1 and b_2, there is 1 unit of slack remaining for energy management. The execution of T_1 and T_2 can be uniformly stretched out and the energy savings would be $\frac{11}{9}E$, a significant improvement over the previous case. Also, the overall system reliability is preserved; in fact, *better* reliability figures are achieved for *both* tasks T_1 and T_2, as indicated by Equation (12).

3.5 Intractability of the Problem

A natural question to ask is whether there exists a fast (i.e. polynomial time) solution to the general reliability-aware energy management problem for multiple tasks. Unfortunately, the answer is negative, as we argue below.

Consider a special case of the general problem where tasks do not have off-chip workload (i.e., $y_i = 0$) and both P_{ind} and C_{ef} are the same for all tasks. In this case, due to the *same* convex relation between power and processing speed, the solution for the minimum energy consumption could be obtained by uniformly scaling down the execution of *selected* tasks using the slack that remains after reserving CPU time for recovery operations. Hence, the problem becomes essentially one of selecting the tasks to be managed.

Suppose that the total amount of computation is $L = \sum_{i=1}^{n} c_i$ and the amount of available slack is $S = D - L$. If the total WCET of the *selected tasks* is X, we have $X \leq L$ and $X \leq S$. After reserving X amount of slack for recovery (in the form of re-execution), the remaining slack $(S - X)$ could be used to scale down the processing frequency for the selected tasks. In that case, the amount of total *fault-free* energy consumption (without considering the execution of recoveries) will be:

$$E_{total} = S\left(P_{ind} + c_{ef} \cdot \left(\frac{X}{S}\right)^m\right) + (L - X)(P_{ind} + c_{ef} \cdot f_{max}^m) \tag{13}$$

where the first part is the energy consumption for the selected tasks and the second part is the energy consumption of unselected tasks. Simple algebra shows that, when $X = S \cdot (\frac{P_{ind}+C_{ef}}{m \cdot C_{ef}})^{\frac{1}{m-1}}$, E_{total} will be minimized. For example, if $P_{ind} = 0, C_{ef} = 1$ and $m = 3$, we will have $X = \frac{\sqrt{3}}{3}S$; that is, the optimal amount of computation to be managed equals $\frac{\sqrt{3}}{3}$ of the amount of available slack.

If $X \geq L$, all tasks could be managed. Otherwise, to minimize the energy consumption while preserving the reliability, the subset of tasks should be selected in such a way that the summation of their WCET requirements is *exactly* equal to X. In other words, such a choice would definitely be the optimal solution. Notice that, having a fast (polynomial time) solution to this problem would imply having a fast solution for SUBSET-SUM problem. The SUBSET-SUM problem involves finding whether there exists a subset of integers $n_1, n_2, \ldots, n_k$ in such a way that the sum of the numbers in the subset is exactly a given number K. Further, the SUBSET-SUM problem is known to be NP-hard [13]. Since this is only a special case, we reach the conclusion that the general reliability-aware energy management problem for multiple tasks should also be NP-Hard.

Considering the problem is computationally intractable, in the following subsections, we propose and evaluate two fast heuristics.

3.6 Reliability-Aware Greedy Heuristic

By extending the reliability-aware energy management scheme for single tasks [27], we can obtain a fast heuristic, that we call *reliability-aware greedy heuristic* for multiple tasks. In this scheme, tasks are selected for management one at a time. After a task is chosen, the task will be allocated as much slack as possible (including the slack for recovery) in an attempt to reduce its speed to its energy efficient frequency. If more slack remains after this assignment, additional tasks will be selected.

Obviously, the order of tasks being selected will affect the number of tasks to be managed as well as the total amount of energy savings. Notice that, because of the recovery needed for preserving reliability, the minimum amount of allocated slack for any selected task should be at least as large as its worst case execution time. Therefore, a certain amount of slack may be wasted at the end, since that slack may not be sufficient for managing the smallest un-selected task. Hence, to avoid wasting significant amount of slack, in this work, we focus on the longest-task-first (LTF) heuristic to determine the order of tasks to be selected for management. We underline that our experiments with other task-ordering rules (including, shortest-task-first) yielded less impressive results.

3.7 SUEF Heuristic

Note that the greedy scheme discussed in the preceding section attempts to reduce the processing speed for the selected tasks aggressively without considering overall energy efficiency. To evaluate the efficiency of slack usage for each task, we define the *slack usage efficiency factor (SUEF)* for task T_k running at speed f as:

$$SUEF_k(f) = \frac{E_k^0 - E_k(f)}{s_k(f)} \tag{14}$$

where E_k^0 and $E_k(f)$ is the energy consumption of task T_k at f_{max} and f, respectively; and $s_k(f)$ is the total amount of slack needed (including the slack reserved for recovery) for task T_k to run at f. That is, it is the ratio of the amount of energy saved to the total amount of slack needed when task T_k runs at a certain speed. The higher the value of SUEF is, the more energy could be saved per unit of slack usage.

Notice that, at the maximum frequency f_{max}, no slack is needed and no energy is saved, and the $SUEF(f_{max})$ for tasks is defined as 0. Intuitively, as f decreases, more energy could be saved and $SUEF(f)$ will increase. However, as the speed approaches the task's energy efficient frequency, less energy is saved for the same amount of slack used. Due to the effects of slack reserved for recovery, it is expected that $SUEF(f)$ will increase and then decrease after a certain threshold. Therefore, for each task T_k, there should exist an optimal speed, f_k^{opt} ($> f_{ee,k}$), at which $SUEF_k(f)$ is maximized. Note that,

$$s_k(f) = (\frac{x_k}{f} - x_k) + (x_k + y_k) = \frac{x_k}{f} + y_k \tag{15}$$

$$E_k(f) = s_k(f)(P_{ind,k} + C_{ef,k} f^m) \tag{16}$$

From Equations (14, 15 and 16) and differentiating $SUEF_k(f)$ with respect to f, we can get that $SUEF_k(f)$ is maximized when f satisfies:

$$mC_{ef,k} f^{m-1}(x_k + f \cdot y_k)^2 = x_k(x_k + y_k)(P_{ind,k} + C_{ef,k}) \tag{17}$$

When $m = 2$ or $m = 3$, this gives rise to *cubic* or *quartic* equations, respectively, and the optimal speed to maximize $SUEF_k(f)$ can be solved analytically. For the case of $y_k = 0$ (i.e., no off-chip workload), the optimal speed can be also easily solved as $f_k^{opt} = (\frac{P_{ind,k} + C_{ef,k}}{m \cdot C_{ef,k}})^{\frac{1}{m-1}}$.

Algorithm 1 SUEF-based Heuristic for Slack Allocation

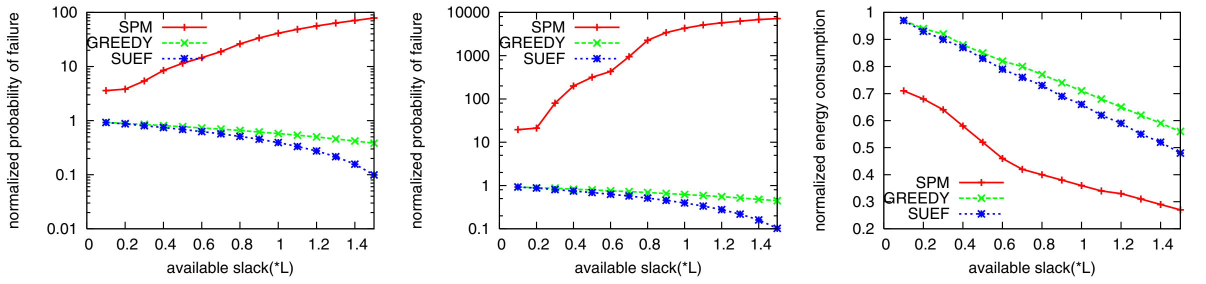
```
1: for all tasks, find $SUEF_k^{max}$, $f_k^{opt}$ and $s_k(f_k^{opt})$;
2: $S_{remain} = S$;
3: Put tasks into Queue in the order of decreasing $SUEF_k^{max}$;
4: while (Queue is not empty and $S_{remain}$ is larger than the
   smallest task in the Queue) do
5:     get the header task $T_k$ from Queue;
6:     if ($s_k(f_k^{opt}) <= S_{remain}$) then
7:         allocate $s_k(f_k^{opt})$ for task $T_k$;
8:         $S_{remain} -= s_k(f_k^{opt})$;
9:     else
10:        store task $T_k$ to the set $\Phi$ of 'unselected' tasks for energy
           management;
11:    end if
12: end while
```

Therefore, for a given task set, it would be most energy efficient to execute tasks that can achieve higher values of $SUEF$. Based on this observation, we propose the SUEF-based heuristic (see Algorithm 1), which selects tasks according to their slack usage efficiency factors. In the algorithm, S_{remain} stands for the amount of remaining slack. *Queue* is used to sort the tasks in the decreasing order of their maximum slack usage efficiency factor $SUEF_k^{max}$ (line 3). The slack is allocated to these tasks in that order, enabling them to run at their optimal speed f_k^{opt} (lines 6, 7 and 8). Φ contains the intact tasks to which no slack is allocated (line 10). Observe that the complexity of the algorithm is only $O(n \log n)$ where n is the number of tasks (and where the dominant term comes from sorting tasks according to their SUEF values).

4. SIMULATIONS AND DISCUSSIONS

To evaluate the effectiveness of our schemes on reliability and energy consumption, we implemented a discrete-event simulator. In the simulator, we implemented and compared the performance of the following schemes:

- **No power management (NPM)** does not explore DVFS but puts the system to power saving sleep states when it is idle. NPM is used as the baseline algorithm in our comparisons;
- **Ordinary Static Power Management (SPM)** uses the slack for energy savings without considering system reliability;
- **Reliability-aware greedy (GREEDY) scheme** selects tasks for management with the LTF heuristic;
- **SUEF-based heuristic (SUEF)** selects tasks for management by considering tasks' SUEF values.



a. normalized probability of failure for $d = 2$ b. normalized probability of failure for $d = 5$ c. normalized energy consumption for $d = 2$

Figure 3: The impact of slack and DVFS-induced fault rates on reliability and energy consumption ($\beta = 0.05$ **and** $C_{ef} = 1.0$)**.**

We simulated a DVFS-enabled environment where the CPU clock frequency can assume any of the five normalized frequency values in the set $\{0.2, 0.4, 0.6, 0.8, 1.0\}$. To capture the relationship between different power consumption components, we use a parameter β which is defined as the ratio of the frequency-independent active power to the maximum frequency-dependent active power. In other words, $\beta = \frac{P_{ind}}{P_{d,max}}$. The frequency-dependent active power is a cubic function of the frequency (i.e., $m = 3$).

We generated synthetic task tests where a task's worst case execution time is randomly chosen between 1 and 10 through a uniform distribution. Because of the space limitations, we present only the results for task sets each having 20 tasks – the results for different number of tasks yield very similar patterns. Finally, we note that each data point that we present corresponds to the average value of 100 task sets.

4.1 Impacts on System Reliability

First, we investigate how these schemes affect the system reliability. Again, the reliability of the real-time application is defined as the probability of completing *all* the tasks (and their recovery blocks, in case of primary tasks catch transient faults) successfully before the application deadline. For convenience, we present the *probability of failure*, which is defined as $1 - reliability$. Note that these reliability figures can be obtained analytically by using the fault rate model and execution time information. Figure 3a and Figure 3b present the results when the exponent d in the fault rate model equals 2 and 5, respectively. The results are normalized with respect to those of the baseline scheme NPM.

In the figures, L $(= \sum_{i=0}^{n} c_i)$ represents the summation of all tasks' WCET and the X-axis denotes the amount of slack available, when normalized with respect to L. Here, we assume that tasks have the same power characteristics (i.e., $\beta_i = 0.05$ and $C_{ef,i} = 1$) and the off-chip workload is negligible (i.e., $y_i = 0$). The effects of these factors are examined in the following subsection.

We observe that SPM leads to drastically decreased system reliability figures even when the fault rate only increases moderately with scaled frequencies and supply voltages (e.g. when $d = 2$). This is to be expected, since SPM uses all the available slack aggressively for energy management, without taking into account the effects on the reliability. However, both GREEDY and SUEF are reliability aware schemes by reserving slack for recovery before reducing the frequency, which preserve (in fact, improve) system reliability even when the fault rate increases sharply (e.g. when $d = 5$). This is consistent with the theoretical results presented in Section 3.3. Observe that, as the amount of slack increases, more tasks can be managed and slightly higher reliability numbers are obtained (i.e., lower probability of failure).

Figure 3c shows the normalized energy consumption for all the schemes under the same settings for the case of $b = 2$. We can see that, due to the recovery needed for preserving system reliability, less slack is available for power management and both GREEDY and SUEF schemes consume more energy (around 25%) than SPM. When there is only a small amount of slack available, SUEF yield only slightly better energy performance than GREEDY since the number of tasks that can be managed is limited. However, as more slack becomes available, more tasks can be managed and the energy consumption difference between SUEF and GREEDY becomes more significant (up to 8%). This is because, the GREEDY scheme tries to execute all the managed tasks at their minimum energy efficient frequencies, which is not the most effective approach considering the slack needed for recoveries. In contrast, SUEF considers the slack needed for recoveries and executes tasks at their optimal frequencies that maximize their slack usage efficiency and that are generally higher than tasks' energy efficient frequencies.

4.2 Impact of Task Characteristics

Next, we evaluate the effects of tasks' power characteristics and off-chip workload on energy savings. The amount of slack is assumed to be $1.5L$. The minimum frequency-independent power is set to $\beta_{min} = 0.05$. The value of β_i for each task is randomly generated between β_{min} and β_{max}, which is the varying parameter. When we investigate the effects of C_{ef} and y, β_i is set to β_{min}. Similar approaches are adopted to get the switching capacitance $C_{ef,i}$ and the off-chip workload y_i values.

Figure 4a shows the effects of varying frequency-independent active power on energy savings for the proposed schemes. As β_{max} increases, on average, the energy efficient frequencies for tasks become higher and the off-chip components tend to consume more power, as discussed in Section 3.2. Therefore, all the schemes will consume more energy. Observe that, the difference between the three schemes becomes less pronounced as β_{max} increases. However, SUEF provides better energy savings compared to GREEDY, throughout the spectrum.

Figure 4b shows that, when the effective switching capacitance increases, more energy can be saved. Again, this is due to the fact that, the energy efficient frequencies for tasks become slightly lower with increasing $C_{ef,max}$, allowing more chances to manage additional tasks. The same reasoning applies to the case of varying the off-chip workload y_{max} in Figure 4c. However, the variations on the off-chip workload lead to bigger difference between the energy efficient frequency and the optimal frequency for maximizing the tasks' Slack Usage Efficiency Factors. As a result, the performance difference between GREEDY and SUEF becomes more significant, where SUEF can provide 25% more energy savings.

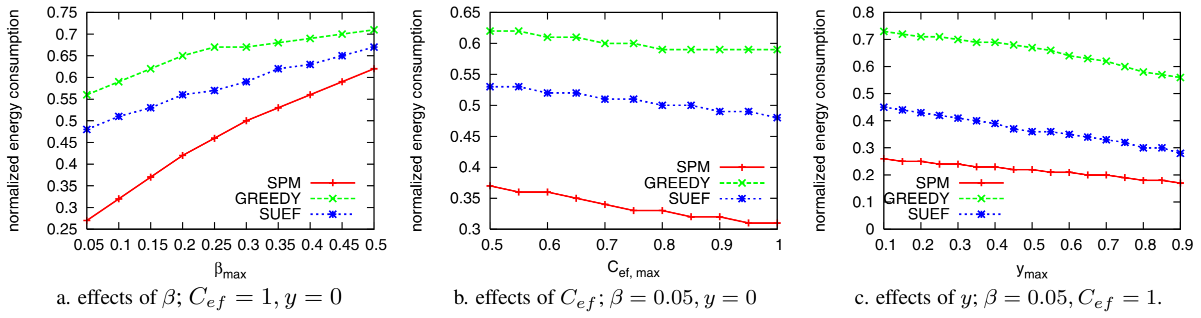


Figure 4: The effects of frequency-independent active power P_{ind}, (by varying β_{max}), active switch capacitance ($C_{ef,max}$) and off-chip workload (y_{max}) on energy savings for the proposed schemes.

5. CONCLUSIONS

With the scaled technology feature size, transient faults in *all* digital systems will become more common. The problem is exacerbated when frequencies/supply voltages are scaled for energy savings. Although both fault tolerance and energy management have been studied extensively, there are only a few researches addressing reliability and energy efficiency trade-offs, simultaneously.

Considering the effects of voltage scaling on transient faults and a generalized power/workload model, we studied the slack allocation problem for multiple tasks to minimize their energy consumption while preserving system reliability. Based on our previous finding of using a recovery to preserve task's reliability, we identified the problem as NP-Hard and proposed two greedy heuristics. The performance for the proposed schemes are evaluated through simulations with synthetic task sets. The results show that the ordinary energy management schemes which ignore the effects of energy management on fault rates are too optimistic and could lead to drastically decreased system reliability. Our heuristic algorithms preserve the system reliability by reserving slack for potential recovery tasks, before applying DVFS for energy savings. Our results indicate that the slack usage efficiency-based heuristic algorithm (SUEF) yields the best results by considering the different power and workload characteristics of tasks.

6. REFERENCES

[1] E. Bini, G.C. Buttazzo, and G. Lipari. Speed modulation in energy-aware real-time systems. *ECRTS*, 2005.

[2] P. Bohrer, E. N. Elnozahy, T. Keller, M. Kistler, C. Lefurgy, C. McDowell, and R. Rajamony. *The case for power management in web servers*, chapter 1. Power Aware Computing. Plenum/Kluwer Publishers, 2002.

[3] T. D. Burd and R. W. Brodersen. Energy efficient cmos microprocessor design. *HICSS*, 1995.

[4] X. Castillo, S. McConnel, and D. Siewiorek. Derivation and caliberation of a transient error reliability model. *IEEE Trans. on computers*, 31(7):658–671, 1982.

[5] K. Choi, W. Lee, R. Soma, and M. Pedram. Dynamic voltage and frequency scaling under a precise energy model considering variable and fixed components of the system power dissipation. *ICCAD*, 2004.

[6] K. Choi, R. Soma, and M. Pedram. Dynamic voltage and frequency scaling based on workload decomposition. *ISLPED*, 2004.

[7] Intel Corp. Mobile pentium iii processor-m datasheet. Order Number: 298340-002, Oct 2001.

[8] V. Degalahal, L. Li, V. Narayanan, M. Kandemir, and M. J. Irwin. Soft errors issues in low-power caches. *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, 13(10):1157–1166, Oct. 2005.

[9] A. Ejlali, M. T. Schmitz, B. M. Al-Hashimi, S. G. Miremadi, and P. Rosinger. Energy efficient seu-tolerance in dvs-enabled real-time systems through information redundancy. *ISLPED*, 2005.

[10] E. (Mootaz) Elnozahy, R. Melhem, and D. Mossé. Energy-efficient duplex and tmr real-time systems. *RTSS*, 2002.

[11] 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. *IEEE Micro*, 24(6):10–20, 2004.

[12] X. Fan, C. Ellis, and A. Lebeck. The synergy between power-aware memory systems and processor voltage. In *PACS*, 2003.

[13] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Mathematical Sciences Series. Freeman, 1979.

[14] P. Hazucha and C. Svensson. Impact of cmos technology scaling on the atmospheric neutron soft error rate. *IEEE Trans. on Nuclear Science*, 47(6):2586–2594, 2000.

[15] R.K. Iyer, D. J. Rossetti, and M.C. Hsueh. Measurement and modeling of computer reliability as affected by system activity. *ACM Trans. on Computer Systems*, 4(3):214–237, Aug. 1986.

[16] R. Jejurikar, C. Pereira, and R. Gupta. Leakage aware dynamic voltage scaling for real-time embedded systems. *DAC*, 2004.

[17] R. Melhem, D. Mossé, and E. (Mootaz) Elnozahy. The interplay of power management and fault recovery in real-time systems. *IEEE Trans. on Computers*, 53(2):217–231, 2004.

[18] D. K. Pradhan. *Fault Tolerance Computing: Theory and Techniques*. Prentice Hall, 1986.

[19] N. Seifert, D. Moyer, N. Leland, and R. Hokinson. Historical trend in alpha-particle induced soft error rates of the alpha microprocessor. *IRPS*, 2001.

[20] K. Seth, A. Anantaraman, F. Mueller, and E. Rotenberg. Fast: Frequency-aware static timing analysis. *RTSS*, 2003.

[21] O. S. Unsal, I. Koren, and C. M. Krishna. Towards energy-aware software-based fault tolerance in real-time systems. *ISLPED*, 2002.

[22] N.J. Wang, J. Quek, T.M. Rafacz, and S.J. Patel. Characterizing the effects of transient faults on a high-performance processor pipeline. *DSN*, 2004.

[23] M. Weiser, B. Welch, A. Demers, and S. Shenker. Scheduling for reduced cpu energy. *OSDI*, 1994.

[24] Heng Zeng, Xiaobo Fan, Carla Ellis, Alvin Lebeck, and Amin Vahdat. Ecosystem: Managing energy as a first class operating system resource. *ASPLOS*, 2002.

[25] Y. Zhang and K. Chakrabarty. Energy-aware adaptive checkpointing in embedded real-time systems. *DATE*, 2003.

[26] Y. Zhang, K. Chakrabarty, and V. Swaminathan. Energy-aware fault tolerance in fixed-priority real-time embedded systems *ICCAD*, 2003.

[27] D. Zhu. Reliability-aware dynamic energy management in dependable embedded real-time systems. *RTAS*, 2006.

[28] D. Zhu, R. Melhem, and D. Mossé. The effects of energy management on reliability in real-time embedded systems. *ICCAD*, 2004.

[29] D. Zhu, R. Melhem, D. Mossé, and E.(Mootaz) Elnozahy. Analysis of an energy efficient optimistic tmr scheme. *ICPADS*, 2004.

[30] J. F. Ziegler. Trends in electronic reliability: Effects of terrestrial cosmic rays. http://www.srim.org/SER/SERTrends.htm, 2004.