Generalized Reliability-Oriented Energy Management for Real-time Embedded Applications[*]

Baoxian Zhao, Hakan Aydin
Department of Computer Science
George Mason University
Fairfax, VA 22030
bzhao@gmu.edu, aydin@cs.gmu.edu

Dakai Zhu
Department of Computer Science
University of Texas at San Antonio
San Antonio, TX 78249
dzhu@cs.utsa.edu

ABSTRACT

DVFS remains an important energy management technique for embedded systems. However, its negative impact on transient fault rates has been recently shown. In this paper, we propose the *Generalized Shared Recovery (GSHR)* technique to optimally use the DVFS technique in order to achieve a given *reliability goal* for real-time embedded applications. Our technique determines the optimal number of recoveries to deploy as well as task-level processing frequencies to minimize the energy consumption while achieving the reliability goal and meeting the timing constraints. The recoveries may be shared among tasks, improving the prospects of DVFS compared to existing reliability-aware power management frameworks. The experimental evaluation points to the close-to-optimal energy savings of our proposed technique.

Categories and Subject Descriptors

D.4.1 [**Operating Systems**]: Process Management—*Scheduling*; D.4.7 [**Operating Systems**]: Organization and Design—*Real-time systems and embedded systems*

General Terms

Algorithms, Design, Reliability

Keywords

Real-Time Embedded Systems, Energy Management, DVFS

1. INTRODUCTION

Portable embedded systems often have to operate under severe power and timing constraints. Consequently, research on low-power real-time embedded systems has been very active in the last decade. In that regard, *Dynamic Voltage and Frequency Scaling (DVFS)* has been one of the most commonly used techniques to manage energy. With DVFS, the processor frequency and supply voltage are lowered at run-time to reduce energy consumption. Since the tasks take longer to execute at lower frequencies, several research studies investigated the problem of guaranteeing the timing constraints in embedded applications while saving energy through DVFS [2, 5, 8, 11, 17].

However, with reduced feature sizes and aggressive low-power design techniques, *soft errors* that result from *transient faults* become more frequent and also an increasing concern for system designers and engineers [1]. More importantly, recent research [7, 16] has illustrated significantly increased (often by several orders of magnitude [16]) transient fault rates when operating at low voltage/frequency levels with DVFS.

As a result, recently a number of research studies promoted the so-called *Reliability-Aware Power Management (RA-PM)* framework, where DVFS is applied for energy management only in a controlled manner while keeping an eye on reliability [6,9,15,16]. Specifically, the RA-PM schemes reserve some CPU time (slack) for *recovery tasks* that may be invoked at run-time upon the detection of faults at the end of task executions. However, existing RA-PM studies share certain fundamental characteristics and limitations. First, the necessary and sufficient objective is casted as to *preserve* the task set's *original reliability*, which is defined as the reliability when all tasks run at the maximum frequency, and without any recovery task. Second, to achieve this aim a *separate* recovery task is scheduled for every task that runs at a low-frequency.

The main objective of this paper is to lay the foundations of a more general RA-PM framework by addressing these two limitations. Our motivation to address the first point is very natural: a more comprehensive framework is highly desirable in order to achieve *any* reliability goal, which can be set by the designer to be *lower* or *higher* than the application's original reliability. This is because, some modest reliability degradation might be acceptable in energy-scarce settings. In contrast, it may be necessary to achieve very high reliability levels for safety-critical systems (e.g. electronics-hostile settings encountered in space applications). Clearly, merely maintaining task-level reliabilities would be too conservative or insufficient, respectively, in these two scenarios.

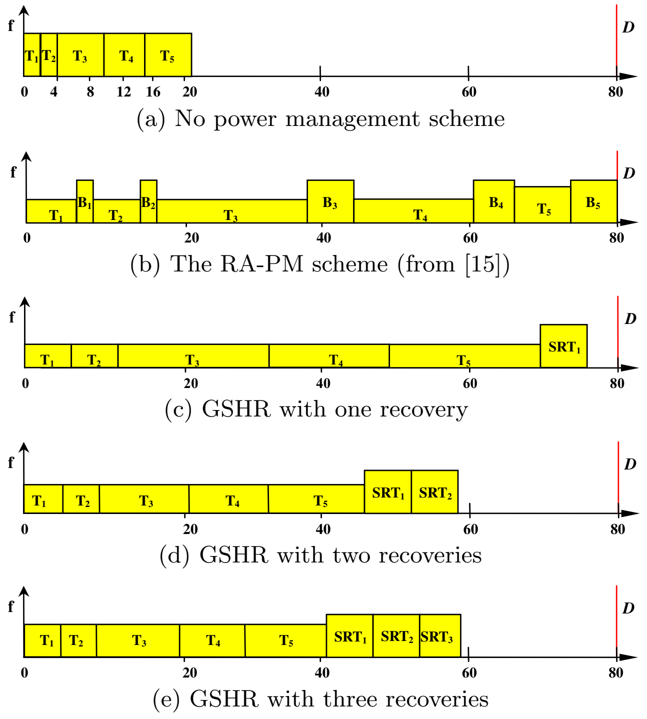
The second motivation is to further *reduce energy consumption*: by scheduling a separate recovery for every scaled task, the existing RA-PM solutions inherently reduce the amount of available slack for DVFS and thus the extent

[*]This work was supported by US National Science Foundation awards CNS-1016855, CNS-1016974, and CAREER Awards CNS-0546244 and CNS-0953005.

of energy savings. A preliminary solution to this problem was recently proposed: the technique, called *shared recovery (SHR)*, reserves only one shared recovery task that can be used by any *single* faulty task at runtime [14]. Hence, SHR can save more energy as larger slack is made available for slow-down through DVFS. Upon the detection of a fault, the shared recovery task is invoked and the remaining tasks are forced to run at the maximum frequency until the end of the current period to preserve the system's original reliability [14].

In this paper, we propose the *Generalized Shared Recovery (GSHR)* technique to potentially use *any* number of recoveries, and using this as a leverage, we tackle the generalized **Energy-Optimal Reliability Configuration (EORC)** problem: *determine the number of recoveries and task-level frequency assignments to minimize the system level energy consumption while satisfying given target reliability and deadline constraints*. Note that each additional (potential) recovery task, while improving the reliability, reduces the available slack for DVFS and limits the opportunities for scaling down frequencies. In other words, the problem mandates an efficient technique in three-dimensional *timing, energy,* and *reliability* space to satisfy all the constraints. For this problem we develop a fast algorithm with comparable performance to that of an optimal, but computationally expensive exhaustive-search based solution.

Motivational Example. To illustrate the potential gains that can be obtained by the new GSHR technique, let us consider an application with five tasks and a common period/deadline of $80\,ms$. The worst-case execution times of T_1 and T_2 are given as 2 ms each, while those of T_3 and T_5 are $6\,ms$ and that of T_4 is $5\,ms$. As can be seen in Figure 1(a), originally there is $80 - 21 = 59\,ms$ slack when all tasks execute at the maximum frequency $1\,GHz$ (*No Power Management* case). In that case, by using the fault rate model from [15], the original *probability of failure (PoF)* is computed as $\rho_0 = 2.1 \times 10^{-7}$. Here the PoF is defined as $1 -$ *reliability*, where the *reliability* is the probability that all tasks will complete without encountering soft errors caused by transient faults. If we consider the traditional greedy RA-PM scheme [15], five *separate* recovery tasks (denoted by $\{B_i\}$) will be statically allocated to five tasks, yielding the RA-PM schedule depicted in Figure 1(b). In that schedule, the first four tasks are executed at the energy-efficient frequency [8] $f_{ee} = 0.29\,GHz$ and the fifth task runs at the frequency $0.78\,GHz$. Applying the power model from [15] indicates that the energy consumption of the RA-PM solution is 7.88 and its actual PoF is 9×10^{-10}. Now, if the objective is to merely preserve the original reliability ρ_0, alternatively, we can use the GSHR technique with one shared recovery task of size $6\,ms$ (denoted by SRT_1) as shown in Figure 1(c). Then, all the tasks can be executed at the frequency $0.31\,GHz$. Simple calculation shows that the total energy consumption is 5.4, giving a 32% reduction with respect to RA-PM. Figure 1(d) presents the GSHR solution with two recoveries that yield the exact reliability obtained from traditional RA-PM scheme (Figure 1(b)). Thus, one can obtain the reliability achieved by RA-PM through GSHR with 15% less energy (6.68), by using the frequency $0.46\,GHz$. Notice that with two shared recoveries, the system can also potentially recover from two faults affecting the same task (unlike RA-PM that statically allocates one recovery to each task), which enables achieving the target



(a) No power management scheme
(b) The RA-PM scheme (from [15])
(c) GSHR with one recovery
(d) GSHR with two recoveries
(e) GSHR with three recoveries

Figure 1: The motivational example

reliability yielded by RA-PM ($PoF = 9 \times 10^{-10}$) with more energy savings. Further, through GSHR one can achieve reliability levels even higher than that achieved by RA-PM, for example $PoF = 5 \times 10^{-10}$, by deploying three shared recoveries and still consuming less energy than RA-PM (7.47) at the uniform frequency level of $0.51\,GHz$. In all these examples, we illustrated the potential gains of GSHR in terms of energy and reliability by using a simple *uniform frequency* approach – as we demonstrate later in this paper, these gains can be further improved by optimizing the frequency selection.

The remainder of this paper is organized as follows. Section 2 introduces the system models used in this paper and presents our assumptions. The generalized shared recovery technique (GSHR) and the energy-optimal reliability configuration problem are introduced Section 3. Our specific solution to the problem, the *IRCS* algorithm, is presented in Section 4. Section 5 presents the simulation results and Section 6 concludes the paper.

2. SYSTEM MODELS

2.1 Application Model

In this work, we consider a frame-based real-time embedded application with n independent real-time tasks. The tasks $\{T_1, T_2, ..., T_n\}$ share a common deadline D, which can be also viewed as the application's period. Without loss of generality, we assume that the task indices indicate the order of execution within each frame (period).

The tasks are executed on a DVFS enabled uniprocessor system and there are ℓ available discrete frequencies, which are sorted in increasing order as $f_{min} = f_1 < f_2 < ... < f_\ell = f_{max}$. We assume that the frequency values are normalized with respect to f_{max} (i.e. $f_{max} = 1$). The worst-case execution time of task T_i under f_{max} is denoted by c_i. We assume the task may take up to $\frac{c_i}{f_i}$ time units when executed at the frequency f_i.

2.2 Power/Energy Model

Our system-level power model follows the previous RA-PM research studies [14,15] by distinguishing the frequency-independent and frequency-dependent power components. As in these works, we assume that shutting down the entire system during the operation of the system is not an option due to the prohibitive overhead and that static power is not manageable. Hence, we concentrate on the active power consumption P_d, which is given as [16]:

$$P_d = P_{ind} + P_{dep} = P_{ind} + C_{ef} f^{\alpha} \quad (1)$$

where P_{ind} is the frequency-independent active power, denoting the power consumed by off-chip devices such as main memory and external devices. However, when putting systems into sleep states, P_{ind} can be efficiently removed [3]. P_{ind} may vary from task to task, and its value for task T_i is denoted by $P_{ind,i}$. P_{dep} is the frequency-dependent active power, including the CPU power, and any power that depends on the processing frequency f [3]. Similarly, P_{dep} will vary with the task currently in execution, in particular with the processing frequency f_i assigned to T_i.

The effective switching capacitance C_{ef} and the dynamic power exponent α (which is, in general, no smaller than 2) are system-dependent constants and f is the processing frequency. Hence, considering only the active power, the *energy* consumption of a task T_i running at the frequency level f_i and with the frequency-independent power $P_{ind,i}$ can be expressed as [16]:

$$E_i(f_i) = (P_{ind,i} + C_{ef} f_i^{\alpha}) \cdot \frac{c_i}{f_i} = P_{ind,i} \frac{c_i}{f_i} + C_{ef} c_i f_i^{\alpha-1} \quad (2)$$

The frequency-independent power component implies the existence of a frequency threshold below which DVFS adversely affects the total energy. This threshold, commonly known as *energy-efficient frequency* (or, *critical speed*), can be computed by standard methods [8,17].

2.3 Fault/Reliability Model

During the execution of an application, faults may occur due to various reasons, such as hardware failures, software errors and the effects of electromagnetic interference and cosmic ray radiations. Previous research has shown that *transient* faults occur much more frequently than *permanent* faults [4]. Therefore, in this work, we focus on *transient* faults and adopt the backward recovery technique to tolerate such faults [10].

More specifically, transient faults are assumed to follow Poisson distribution with an average arrival rate λ [13]. Considering the effects of voltage scaling on transient faults, the average fault rate λ will depend on system supply voltage and processing frequency. In this paper, we use the exponential fault rate model used in [14–16]:

$$\lambda(f) = \lambda_0 \, 10^{\frac{d(1-f)}{1-f_{min}}} \quad (3)$$

where the exponent d (> 0) is a constant, indicating the sensitivity of fault rates to voltage scaling. λ_0 is the average fault rate corresponding to the maximum frequency $f_{max} = 1$ (and supply voltage V_{max}). That is, reducing the supply voltage and frequency for energy savings will result in exponentially increased fault rates. The maximum average fault rate is assumed to be $\lambda_{max} = \lambda_0 \cdot 10^d$, which corresponds to the lowest frequency f_{min} (and supply voltage V_{min}).

The *reliability* of a task is defined as the probability of completing its execution successfully (i.e. without encountering failures triggered by transient faults) [15, 16]. As a result, during a single frame, the reliability of the task T_i with the worst-case execution time c_i, when executed at the frequency level f_i is [16]:

$$R_i(f_i) = e^{-\lambda(f_i) \cdot \frac{c_i}{f_i}} \quad (4)$$

where $\lambda(f_i)$ is given by Equation (3).

Following the previous research [14–16], we assume that faults are tolerated through backward recovery technique and the recovery takes the form of re-execution at the maximum frequency (f_{max}) when a fault is detected at the end of a task's execution. The use of f_{max} for recovery leaves more slack for fault-free executions and improves energy savings, as transient faults occur with low probability.

3. GENERALIZED SHARED RECOVERY

In [14], a *shared recovery (SHR)* technique has been proposed where all tasks share one recovery, which is utilized to recover the first faulty task and the remaining tasks within one frame (period) will run at f_{max} for reliability preservation. Extending this idea, for general reliability goals, we can employ a fixed number ($j \geq 0$) of recovery blocks, which will be shared by n tasks within a frame. By choosing proper task frequency assignments, a broad spectrum of reliability can be obtained. We call this scheme *Generalized Shared Recovery (GSHR)* technique. Unlike SHR, *task-level* reliability degradation is allowed in GSHR, as long as the system's target reliability is still achieved. Specifically, in GSHR, after a fault and possible recovery/ies, the remaining tasks can still run at their pre-calculated scaled frequencies. Moreover, for j shared recoveries, GSHR reserves a total CPU time equal to the sum of the first j largest tasks.

Let $R^j(T_1, .., T_n)$ denote the reliability (i.e. the probability of completing successfully the execution of tasks $T_1, .., T_n$) by using the slack of j recoveries if necessary. Further, let R_g be the reliability goal to achieve. Then, the **Energy-Optimal Reliability Configuration (EORC)** problem can be formally expressed as: find the optimal number j of recovery tasks and frequency assignments $f_1, f_2, ..., f_n$ to minimize:

$$E = \sum_{i=1}^{n} E_i(f_i) = \sum_{i=1}^{n} (P_{ind,i} \frac{c_i}{f_i} + C_{ef} c_i f_i^{\alpha-1}) \quad (5)$$

Subject to:

$$R^j(T_1, .., T_n) \geq R_g \quad (6)$$

$$\sum_{i=1}^{n} \frac{c_i}{f_i} \leq D' \quad (7)$$

where $D' = D - S_j$, and S_j is the total slack required for j recoveries.

Intuitively, with more recoveries, the target reliability R_g can be achieved more easily. However, the deadline constraint further complicates the problem, as each additional

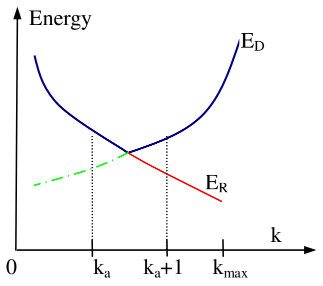


Figure 2: Energy-Optimal Reliability Configuration

recovery will reduce the available slack and limit the opportunities for DVFS to save energy. Figure 2 depicts these non-trivial trade-off dimensions for the simultaneous consideration of the reliability and deadline constraints. The curve E_D illustrates the minimum energy consumption that can be achieved when deploying k recoveries by ignoring the reliability constraint R_g. Similarly, the curve E_R gives the minimum energy consumption figure satisfying the target reliability constraint, but without taking the deadline constraint into consideration. Since the actual solution must satisfy both reliability and deadline constraints, the best energy performance that can be achieved with k recoveries corresponds to the union of the upper parts of these two plots, namely $max\{E_D(k), E_R(k)\}$. As a result, the optimal number of recoveries is either k_a or $k_a + 1$ as illustrated in the figure.

Before concluding this section, we sketch how the reliability of n tasks with j recoveries, namely $R^j(T_1, .., T_n)$, can be computed. The overall reliability when no recovery task is scheduled, is by definition the product of individual task reliabilities, i.e. $R^0(T_1, .., T_n) = \prod_{i=1}^{n} R_i(f_i)$ [15]. Similarly, considering that the recovery tasks are always invoked at f_{max}, the reliability with a single recovery is given by [15]:

$$R^1(T_1, .., T_n) = R_1(f_1)R^1(T_2, .., T_n) + (1 - R_1(f_1))R_1(f_{max})R^0(T_2, .., T_n) \quad (8)$$

In the general case, the reliability with j recoveries can be expressed recursively as:

$$R^j(T_1, .., T_n) = R_1(f_1)R^j(T_2, .., T_n) + (1 - R_1(f_1))R_1(f_{max})R^{j-1}(T_2, .., T_n) \quad (9)$$

The first term above corresponds to the scenario where task T_1 completes successfully and all j recoveries are made available to the remaining tasks. Similarly, the second term captures the scenario where task T_1 fails and subsequently is recovered with one recovery block. The remaining tasks will have $j - 1$ recoveries to complete their executions. While the details have to be omitted due to space limitations, we note that the value of $R^j(T_1, .., T_n)$ can be quickly computed by the well-known dynamic programming technique (with linear asymptotic complexity $O(j \cdot n)$).

4. COMPUTING ENERGY-OPTIMAL CONFIGURATIONS: ALGORITHM IRCS

While an exact and fast solution may be out of reach due to the multi-dimensional interplay of *deadline, energy* and *reliability* factors, in the following, we propose an algorithm called *Incremental Reliability Configuration Search (IRCS)* and establish its good performance through experimental evaluation. The basic idea of the IRCS algorithm is to iteratively determine the best frequency assignments for different recovery task allocation options, and select the best alternative at the end. That is, for a given number of recoveries k, IRCS reduces the frequencies of individual tasks as long as the overall reliability is no smaller than the target reliability, R_g. Initially, we set $f_i = f_{max} = f_\ell$ for all tasks. Then we determine the task to scale down to the next (lower) frequency level, by estimating the task that would yield large energy savings with relatively low reliability degradation in every step.

Algorithm 1 Incremental Reliability Configuration Search (IRCS)

```
1: For all tasks, compute $W_i^j$, $Q_i^j$, $S_i^j$ and $ERR_i^j$;
2: Sort all task execution time in decreasing order as $b_1, ..., b_n$;
3: $AC = D - \sum_{i=1}^{n} c_i$;
4: $k = max\{ j \mid \sum_{i=1}^{j} b_i \leq AC\}$ //compute the maximum number of recoveries.
5: Set $k_{opt} = 0$;
6: Set $f_i = f_\ell$ $(i = 1, ..., n)$;
7: Set $E = \sum E_i(f_i)$;
8: for $j = 0$ to $k$ do
9:    if $j > 0$ then
10:      $AC = AC - b_j$
11:   end if
12:   Assign-frequencies($\{S_i^m\}, AC$);
13:   Compute the total energy consumption $E_{total}$ under the new frequency assignments $\{f_i^*\}$
14:   if $E > E_{total}$ then
15:      Set $E = E_{total}$;
16:      Set $k_{opt} = j$;
17:      Set $f_i = f_i^*$ $(i = 1, .., n)$;
18:   end if
19: end for
20: return $k_{opt}$ and $f_1, ..., f_n$
```

Specifically, we first define $W_i^j = E_i(f_{j+1}) - E_i(f_j)$, which indicates the energy savings obtained when the task T_i's frequency is scaled down from f_{j+1} to f_j. Similarly, $Q_i^j = R_i(f_{j+1}) - R_i(f_j)$ indicates the reliability degradation following this frequency reduction. Now, we define $ERR_i^j = \frac{W_i^j}{Q_i^j}$ as the *energy-reliability ratio (ERR)* of task T_i when scaling down from frequency $j + 1$ to the frequency j. *ERR* is a measure of *utility* (i.e. energy savings per unit reliability degradation) that guides our search.

The IRCS algorithm (Algorithm 1) iterates over possible number of recoveries (Lines 8-19), selecting the best frequency assignment along the way. For this purpose, Algorithm 2 *Assign-Frequencies* is invoked to choose the task with the largest *ERR* value for the subsequent slow-down decision(s). *Assign-Frequencies* continues to reduce the task frequencies iteratively by this principle as long as the target reliability R_g is still satisfied. Note that, the input to Algorithm 2 is an array of task-level frequency-dependent slack usage factors $\{S_i^m\}$ and available total slack for DVFS (AC) after reserving the CPU time for the given number recoveries. $S_i^j = \frac{c_i}{f_j} - \frac{c_i}{f_{j+1}}$ denotes the CPU time needed to scale task T_i from the frequency f_{j+1} to f_j.

Algorithm 2 *Assign-frequencies*($\{S_i^m\}, AC$)

```
1: For all tasks, set frequency $f_i^* = f_\ell$ and $z_i = \ell - 1$;
2: Set $X = AC$;
3: while $X > 0$ and there exists $i$ such that $S_i^{z_i} \leq X$ do
4:     In the set $\alpha = \{T_i | S_i^{z_i} \leq X\}$, find task $T_y$ with maximum $ERR_y^{z_y}$ value;
5:     $f_y^* = f_{z_y}$;
6:     Compute the new system reliability $R_{new}$ by Eq. (9);
7:     if $R_{new} < R_g$ then
8:         Set $f_y^* = f_{z_y+1}$ and break
9:     else
10:        Set $X = X - S_y^{z_y}$;
11:        Set $z_y = z_y - 1$;
12:    end if
13: end while
14: return $f_1^*, ..., f_n^*$
```

5. PERFORMANCE EVALUATION

To evaluate the performance of our proposed solution, we implemented a discrete-event simulator in C programming language. In our simulator, we implemented the following four schemes:

- The *uniform frequency* **(UF)** scheme, which chooses the best *constant* frequency level for all tasks. UF simply evaluates all discrete frequency levels (whose number is typically a small integer) in terms of energy consumption and meeting the reliability goal, and selects the best solution.

- Our new *incremental reliability configuration search* **(IRCS)** scheme which tries to obtain best energy savings by scaling down the tasks step by step based on their energy-reliability ratio (ERR) values.

- The greedy *RA-PM* **(RAPM)** scheme [15] which aims to minimize the energy consumption and still maintain the system's original reliability. Notice that RAPM is designed to reach a reliability level no less than the system's original reliability, and cannot be configured to yield *arbitrary* reliability figures.

- The *Optimal* **(OPT)** scheme, which obtains the energy optimal reliability configuration by performing an exhaustive search in all possible frequency assignments for all the tasks. Even with small number of frequency levels, the number of distinct task frequency assignments grows exponentially, and this solution is not scalable. While not practical, OPT is included in our comparison as a yardstick algorithm.

In the simulations, transient faults are assumed to follow Poisson distribution with an average fault rate of $\lambda_0 = 10^{-6}$ at f_{max}, which is a realistic fault rate as reported in [18]. Moreover, the fault rate exponent d is set to 2. The six discrete frequency levels that we assume are modeled after Intel Xscale processor [12]. We use a cubic frequency-dependent power component P_d which is equal to unity at $f_{max} = 1.0$. The frequency-independent power component P_{ind} for each task is normalized with respect to P_d and is generated according to the uniform distribution in the range of $[0, 2]$.

For each task set (that contains 10 tasks), the worst-case execution time c_i for each task is randomly generated through a uniform distribution, such that the worst-case execution time under maximum frequency falls in the range of $[1ms, 10ms]$. All energy consumption results are normalized with respect to the *no power management (NPM)* scheme that executes all tasks without any frequency and voltage scaling at f_{max}. The original probability of failure PoF without any energy management (i.e. $1 - R_0 = 1 - \prod R_i(f_{max})$) is denoted by ρ_o. Each point in the presented figures is obtained by averaging the results obtained through 1000 different task sets.

First, we analyze the impact of available slack on the energy consumption, as shown in Figures 3 and 4. The frequency-independent power is set to $P_{ind} = 0.05$ and the available slack is represented by $L = D - C$, where $C = \sum c_i$. In Figure 3, the target reliability is set to original system reliability $R_g = 1 - \rho_o$. The energy savings of IRCS is extremely close to that of OPT and represents an improvement of up to 30% compared to RAPM. Notice that even UF is able to outperform RAPM, when using GSHR for recovery task assignments. The larger slack, the more more opportunities for both DVFS and recovery assignment; and all schemes achieve more energy savings with increasing L. At large L values, IRCS and UF can easily use low frequencies by deploying suitable number of shared recoveries. However, RAPM's options are more limited, as it is forced to allocate separate recovery tasks to different tasks.

Figure 4 repeats the evaluation for the case when a more strict reliability goal of $R_g = 1 - \frac{\rho_o}{1000}$ is imposed. In other words, in this case the system is supposed to be configured to reach a probability of failure level 1000 times smaller than the original case. RAPM simply cannot reach this very high reliability level, and thus is excluded from comparison. While similar trends are observed here, note that the energy consumptions of all schemes increase due to high-frequency executions, mandated by a more stringent reliability goal. The difference between IRCS and OPT becomes more visible in this scenario, however it does not exceed 10%.

Figure 5 further illustrates the impact of the reliability target R_g on system energy consumption, with the available slack $L = 1.1 \cdot C$ and $P_{ind} = 0.05$. Here, we vary the target PoF from 10^{-4} to 10^{-11}. As expected, the energy consumptions of all schemes get higher with increasing reliability objective (i.e decreasing PoF). This is due to the need for scheduling more recovery tasks to meet the target reliability, resulting in less available slack for DVFS. When $PoF < 10^{-9}$, the energy savings of IRCS is very close to that of the ideal OPT scheme. After that, OPT starts to perform slightly better, but as mentioned before, at very high computational cost. When $PoF = 10^{-11}$, all schemes are forced to run at the maximum frequency f_{max}, and also deploy recoveries, to achieve this high reliability level.

Finally, we investigate the effects of different frequency-independent active power P_{ind} on the system energy savings when $L = 1.1 \cdot C$ and the target reliability is the system original reliability $R_0 = 1 - \rho_0$. P_{ind} value for each task T_i is randomly generated from $[P_{ind,min}, P_{ind,max}]$ where $P_{ind,min}$ is set to 0. Figure 6 shows that energy consumptions of all schemes increase with increasing $P_{ind,max}$. This is because, as $P_{ind,max}$ increases, the frequency-independent energy consumption increases and further, the energy-efficient frequency thresholds for tasks become higher, which limits the DVFS opportunities. Consequently, when $P_{ind,max} \geq 0.2$, the energy consumptions of the IRCS and UF coincide

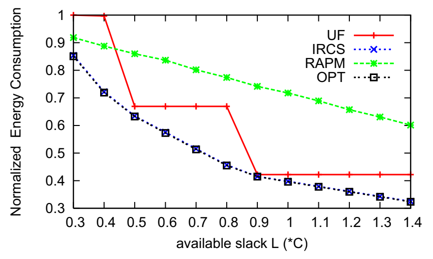


Figure 3: The impact of slack on energy when $R_g = 1 - \rho_o$

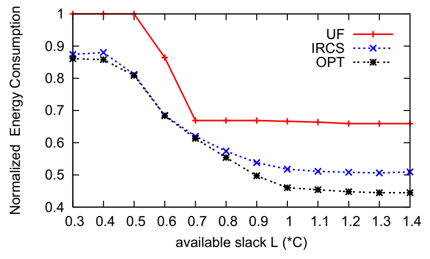


Figure 4: The impact of slack on energy when $R_g = 1 - \frac{\rho_o}{1000}$

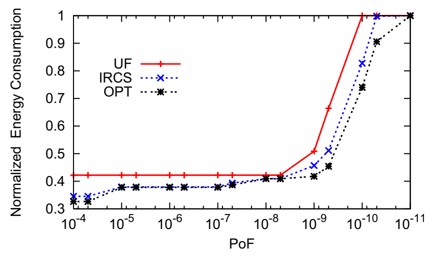


Figure 5: The impact of *PoF* on system energy consumption

with that of ideal OPT scheme. After this threshold, by executing all tasks at the energy-efficient frequency levels with the suitable numbers of shared recoveries, the original reliability can be preserved.

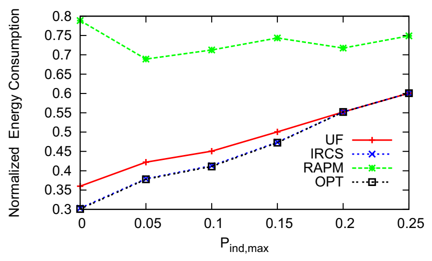


Figure 6: The impact of P_{ind} on system energy consumption

6. CONCLUSIONS

In this paper, we addressed the *energy-optimal reliability configuration (EORC)* problem for real-time embedded applications with the goal of minimizing system energy consumption while satisfying an arbitrary target reliability and deadline constraints. To this aim, we developed the *Generalized Shared Recovery (GSHR)* technique through which a small number of recovery tasks are shared by all the tasks. Our experimental evaluation indicates that for a broad range of reliability objective goals, our proposed solution IRCS delivers close-to-optimal energy savings.

7. REFERENCES

[1] H. Aydin. Exact fault-sensitive feasibility analysis of real-time tasks. *IEEE Transactions on Computers*, 56(10):1372 – 1386, 2007.

[2] H. Aydin, R. Melhem, D. Mossé, and P. Mejia-Alvarez. Power-aware scheduling for periodic real-time tasks. *IEEE Transactions on Computers*, 53(5):584 – 600, 2004.

[3] T. Burd and R. Brodersen. Energy efficient cmos microprocessor design. *Proceedings of the 28th Hawaii International Conference on System Sciences (HICSS)*, 1995.

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

[5] K. Choi, R. Soma, and M. Pedram. Fine-grained dynamic voltage and frequency scaling for precise energy and performance trade-off based on the ratio of off-chip access to on-chip computation times. In *Proc. Design, Automation and Test in Europe (DATE)*, 2004.

[6] A. Ejlali, B. M. Al-Hashimi, and P. Eles. A standby-sparing technique with low energy-overhead for fault-tolerant hard real-time systems. *Proceedings of International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, 2009.

[7] 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.

[8] R. Jejurikar and R. Gupta. Dynamic voltage scaling for systemwide energy minimization in real-time embedded systems. In *Proc. International Symposium on Low Power Electronics and Design (ISLPED)*, 2004.

[9] P. Pop, K. H. Poulsen, V. Izosimov, and P. Eles. Scheduling and voltage scaling for energy/reliability trade-offs in fault-tolerant time-triggered embedded systems. *Proceedings of International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, 2007.

[10] D. K. Pradhan. *Fault-tolerant computer system design.* Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1996.

[11] G. Quan and X. Hu. Energy efficient fixed-priority scheduling for real-time systems on variable voltage processors. In *Proc. the Annual Conference on Design Automation (DAC)*, 2001.

[12] R. Xu, D. Mossé, and R. Melhem. Minimzing expected energy consumption in real-time systems through dynamic voltage scaling. *ACM Transactions on Computer Systems*, 25(4), 2007.

[13] Y. Zhang and K. Chakrabarty. Energy-aware adaptive checkpointing in embedded real-time systems. In *Proc. Design, Automation and Test in Europe (DATE)*, 2003.

[14] B. Zhao, H. Aydin, and D. Zhu. Enhanced reliability-aware power management through shared recovery technique. In *Proc. International Conference on Computer Aided Design (ICCAD)*, 2009.

[15] D. Zhu and H. Aydin. Energy management for real-time embedded systems with reliability requirements. In *Proc. International Conference on Computer Aided Design (ICCAD)*, 2006.

[16] D. Zhu, R. Melhem, and D. Mossé. The effects of energy management on reliability in real-time embedded systems. In *Proc. International Conference on Computer Aided Design (ICCAD)*, 2004.

[17] J. Zhuo and C. Chakrabarti. System-level energy-efficient dynamic task scheduling. In *Proc. the Annual Conference on Design Automation (DAC)*, 2005.

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