Enhanced Reliability-Aware Power Management through Shared Recovery Technique*

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

While Dynamic Voltage Scaling (DVS) remains as a popular energy management technique for real-time embedded applications, recent research has identified significant and negative impact of voltage scaling on system reliability. For this reason, a number of reliability-aware power management (RA-PM) schemes were recently proposed to preserve the system reliability when DVS is used. In this paper, we propose a new approach, called the *shared recovery (SHR)* technique, to minimize the system-level energy consumption while still preserving the system's original reliability. The main idea of the SHR technique is to avoid the offline allocation of separate recovery tasks to the scaled tasks by assigning a global/shared recovery block that can be used by any task at run-time. Our simulation results show that, compared to the existing RA-PM schemes, our scheme can achieve up to 35% energy savings. Further, this performance is shown to be comparable to the maximum energy savings that can be achieved by any algorithm. Interestingly, our extensive evaluation indicates that SHR offers also non-trivial gains over the previous algorithms on the reliability side. Further, a dynamic extension is proposed to improve energy and reliability management at run-time by reducing the size of the recovery block and re-using the slack that arises from early completions.

Categories and Subject Descriptors

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

General Terms

Algorithms, Performance

*This work is supported in part by US National Science Foundation grants CNS-0720647, CNS-0720651 and CNS-546244 (CAREER Award).

ICCAD'09 November 2-5, 2009, San Jose, CA

1. INTRODUCTION

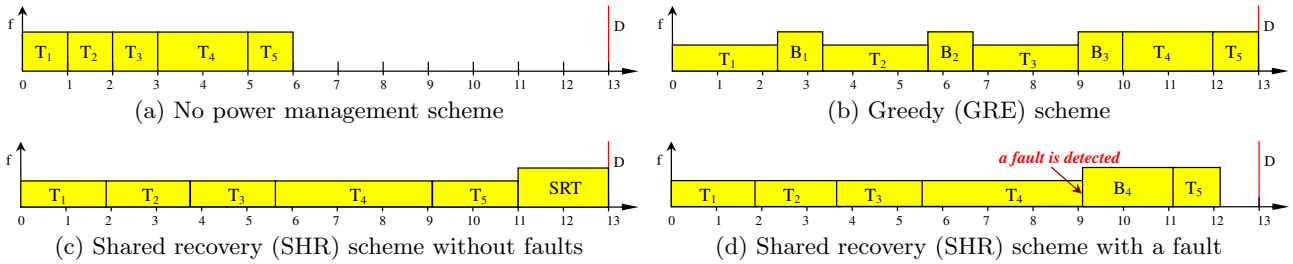
Dynamic voltage scaling (DVS) [19, 20] is one of the most important energy management techniques for computing systems with limited power. DVS achieves energy savings by scaling down the CPU supply voltage and frequency at run-time. Recently, many research studies explored the use of DVS to minimize energy consumption while meeting the applications' performance requirements, in particular for real-time embedded systems [3, 6, 13, 15, 17, 18, 27].

In another front, the *transient faults* that can occur at run-time have attracted the attention of the researchers for decades [5, 11], leading to the design of several fault tolerance solutions for safety-critical systems [16]. Further, recent research reported significant *reliability* degradation for systems that use the DVS feature [8, 25]. In fact, the *probability of failure* for an application executed at low voltage and processing frequency levels can increase by several orders of magnitude [8,25,26]. As a result, a number of recent research articles promoted the so-called *reliability-aware power management (RA-PM)* framework, where the aim is to minimize the system-wide energy consumption through DVS while *maintaining the original system reliability.*

The central idea behind RA-PM is to schedule a *recovery task* that may be invoked at run-time, in case that the application, whose execution frequency is scaled down through DVS, incurs a transient fault [22]. Specifically, when a transient fault is detected at the end of task execution, a recovery takes place in the form of re-execution at the maximum frequency before the task deadline. Based on this principle, several schemes are proposed for frame-based [23] and general periodic [24] tasks. A main feature of the existing RA-PM schemes is that the algorithms first reserve some portion of the system slack (CPU time) for potential recovery of the tasks that may be subject to transient faults when executed with voltage scaling. Only then, the remaining slack is used to determine the task-level low processing frequencies for energy management.

This research effort is motivated by the observation that the existing RA-PM schemes are *conservative*, in the sense that statically allocating the available slack for *multiple recovery blocks of a pre-determined set of tasks reduces the prospects for energy savings* by decreasing the available slack for DVS. Instead, our novel solution is based on allocating a **shared recovery block/slack** that can be used by *any* faulty task at run-time. This, in turn, improves energy savings.

To illustrate this point, we present a concrete example. Consider a frame-based real-time application that consists of



(a) No power management scheme

(b) Greedy (GRE) scheme

(c) Shared recovery (SHR) scheme without faults

(d) Shared recovery (SHR) scheme with a fault

Figure 1: The motivational example

five tasks and a common period/deadline of 13. The worst-case execution times of T_1, T_2, T_3 and T_5 under maximum frequency are given as 1 time unit each, while that of T_4 is 2. As can be seen in Figure 1(a), the system potentially has access to $13 - 6 = 7$ units of slack when all tasks execute at the maximum frequency. If we consider one of the existing RA-PM schemes (e.g. the greedy (GRE) scheme proposed in [23]), three *separate* recovery tasks (denoted by $\{B_i\}$) will be statically allocated to T_1, T_2 and T_3, leaving 4 units of total slack for voltage scaling of these three tasks and yielding the RA-PM schedule depicted in Figure 1(b). Applying the power model from [23] indicates that the energy consumption of the GRE solution is $0.74 \cdot E$ where E is the energy consumed when all tasks run at the maximum frequency (no power management (NPM) case). It can be also shown that since all tasks that are scaled down have recovery tasks, each task's reliability is no worse than the one in the NPM case (Figure 1(a)). One may further verify that the system reliability, which is the product of the individual task reliabilities [23, 24], actually improves in the GRE solution to a level of 99.99998% when using the fault rate model from [23].

On the other hand, our solution in this paper is based on assigning a *shared recovery block* that can be used for *any* fault detected in any scaled task. That is, instead of reserving slack for separate recovery tasks in advance, our new scheme (called *shared recovery (SHR)* scheme) will only provision for any single recovery that may be needed at run-time. The amount of CPU time reserved for this recovery will be set to that of the largest recovery/re-execution time that may be needed for any scaled task. Just like the original RA-PM schemes [22, 26], we assume that the recovery, if needed, will take place at the maximum frequency. As a result, with the new SHR scheme, the system will typically have access to a larger slack for slowdown and energy savings, after the CPU time reservation for the shared recovery. Re-visiting the motivational example, we note that the SHR scheme will reserve a recovery slack of only 2 time units, which is the maximum needed by any recovery. This essentially gives the system an opportunity to use 5 units of slack for, and further, manage *all* the five tasks through, DVS. It can be verified that the SHR solution given in Figure 1(c) will result in an energy consumption of $0.51 \cdot E$, an improvement of 31% over the traditional GRE scheme.

Obviously, another important objective is still to preserve the *original system reliability*. It is clear that as long as faults do not occur, each scaled task will have access to a recovery slack that is large enough for re-execution at its dispatch time, in our scheme. However, one can see that after a fault occurs and the shared recovery block has been executed, if all the remaining tasks are still executed at the originally-computed reduced frequency levels, their task-level reliability will not be preserved. For this reason, our SHR scheme switches to the maximum CPU frequency until (and only until) the end of the frame/period for the remaining tasks once a recovery takes place. In fact, in Section 2.3, we will formally prove that the SHR scheme preserves *the system's original reliability* by maintaining the task-level reliability.

Figure 1(d) shows a scenario where the shared recovery is dynamically allocated to the re-execution of T_4 at f_{max} after the detection of a transient fault in this task. As a result, T_5 is executed at the maximum frequency. Interestingly, in this example, the reliability achieved by SHR (computed as 99.999999%) turns out to be even better than that of GRE. This is essentially due to the fact that the GRE schedule in Figure 1(b) could not allocate any recovery tasks for T_4 and T_5, whereas SHR improved reliability for *all* the tasks through the shared recovery. While it is true that the GRE solution can recover from some multiple-fault scenarios (affecting, for example, T_1 and T_2), an important system design principle is to optimize more common cases. In our example, covering *all* single-fault scenarios turns out to be more important than covering some multiple-fault scenarios that can occur with very low probability. Our simulation results will further confirm this trend.

Our proposal also includes a *dynamic* extension to our base SHR scheme, that adjusts the size of the shared recovery and processing frequencies at run-time, by taking into account the early completions as well as large tasks that complete successfully without a fault at run-time. Our extensive experimental evaluation indicates our scheme offers not only significant energy gains over existing RA-PM schemes, but a markedly close performance to optimal (but reliability-oblivious) DVS schemes that reserve the entire slack for slow-down.

2. SYSTEM MODELS

2.1 Application Model

We consider a real-time embedded application that consists of a set Γ of n independent tasks $T_1, T_2, \ldots, T_n$. All the tasks have the common deadline D, which also corresponds to the *period* (or, *frame*) for periodic applications. Each task T_i is characterized by its worst case number of execution cycles c_i.

We consider a system with DVS capability where the clock frequency can vary from a minimum available frequency f_{min} to a maximum frequency f_{max} (normalized to 1.0). The execution time of task T_i under the frequency f_i is given

by $\frac{c_i}{f_i}$. As a result, for task T_i, c_i can also be viewed as its *worst-case execution time* when it is executed at the maximum speed $f_{max} = 1$.

2.2 Power and Energy Model

DVS exploits the fact that there is an almost linear relationship between the supply voltage and operating frequency [4], and the supply voltage is reduced alongside the processing frequency [14]. In this paper, we focus on DVS technique and adopt the system-level power model proposed in [25] and subsequently used in prior RA-PM research [22–24]. Hence, the system power consumption P is given by:

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

Here P_s is the static power, which includes the power to maintain basic circuits, and keep the clock running and the memory in sleep modes [9]. It can be removed only by powering off the whole system. P_{ind} is the frequency-independent active power and P_d is the frequency-dependent active power. P_{ind} is a constant independent of processing frequencies (i.e. the power consumed by off-chip devices such as main memory and external devices) and can be efficiently removed by putting systems into sleep states [1, 4]. As tasks may use different off-chip devices, in general, P_{ind} may show some variation from task to task. P_d is the power mainly consumed by the CPU, in addition to any power that depends on the frequencies [4]. $\hbar$ represents system states and indicates whether active powers are currently consumed in the system. Specifically, when the system is active, $\hbar = 1$; 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.

Since there exists an excessive overhead associated with turning on/off a system [7], we assume that the system is always active while our periodic application is running. As P_s is not manageable, we will ignore the sleep power P_s and concentrate on the frequency-independent active power P_{ind} and frequency-dependent active power P_d in our analysis.

Consequently, the overall *energy* consumption expression for a task with execution cycles c at the frequency f can be obtained by considering both the frequency-independent and frequency-dependent power components as in [22]:

$$E(f) = (P_{ind} + C_{ef} f^m) \cdot \frac{c}{f} = P_{ind} \cdot \frac{c}{f} + C_{ef} \cdot c \cdot f^{m-1} \quad (2)$$

Although DVS can reduce energy consumption because of the reduced frequency-dependent active power P_d at reduced processing frequencies, the application will take more time to complete at low frequencies. As a result, more energy may be consumed because of the prolonged device active times (due to frequency-independent active power P_{ind}). Therefore, considering the system-level power, lower frequencies may not be always best for energy savings and it is shown that there exists a minimum energy-efficient voltage/frequency pair [9, 13, 27]. In [26], the energy-efficient frequency for our power model is given as:

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

Hence, it is not energy-efficient to run any task at a frequency below f_{ee} even when the timing constraints allow; doing otherwise would result in higher energy consumption. In fact, if f_{ee} exceeds the maximum available frequency level f_{max}, then the system should not reduce its speed below f_{max} [2].

2.3 Fault and Reliability Models

During the execution of an application, a fault may occur due to various reasons, such as hardware failure, software errors and the effect of electromagnetic interference and cosmic ray radiations. Since *transient* faults occur much more frequently than *permanent* faults [5, 11, 12], in this paper, we focus on transient faults, especially the ones caused by cosmic ray radiations and electromagnetic interference.

Traditionally, transient faults have been modeled by Poisson distributions with the average arrival rate λ [21]. However, considering the effects of voltage scaling on transient faults [8, 25], the average rate λ will depend on system processing frequency and supply voltage. In our analysis and simulations, we focus on the exponential fault rate model proposed in [25] and again used in prior RA-PM research [22–25]:

$$\lambda(f) = \lambda_0 \, 10^{\frac{d(1-f)}{1-f_{min}}} \quad (4)$$

where the exponent d (> 0) is a constant, indicating the sensitivity of fault rates to voltage scaling, while 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 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}).

The *reliability* of a task is defined as the probability of completing the task successfully (i.e. without encountering errors triggered by transient faults) [22, 23, 25]. Assuming that the transient faults follow a Poisson distribution, the reliability of the task T_i with the worst-case number of cycles c_i at the frequency level f_i is [25]:

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

where $\lambda(f_i)$ is defined as in (4). The system's *original* reliability is the one observed when the voltage and frequency scaling is disabled (i.e when each task executes at the maximum frequency level f_{max}) [22–24, 26]. The overall reliability of a real-time system depends on the correct execution of all tasks in the application [26]. As a result, for n independent tasks in our application model, the system original reliability is given as $\prod_{i=1}^{n} R_i^0$, where $R_i^0 = R_i(f_{max}) = e^{-\lambda_0 c_i}$ represents the *original reliability* of T_i (also called original task-level reliability). Note that if a scheme maintains all the original task-level reliabilities, the system's original reliability will be automatically preserved.

Problem Description. In this work, our objective is to *minimize the system energy consumption as much as possible for multiple frame-based tasks through DVS while still preserving system original reliability and meeting the deadline constraints.* In what follows, we present the details of our new RA-PM scheme, which offers significant advantages over existing schemes. In accordance with the prior RA-PM research [22–26], we assume that the recovery takes the form of re-execution and is executed at the maximum frequency when a fault is detected.

3. THE SHARED RECOVERY TECHNIQUE

The reliability-aware power management (RA-PM) framework is based on maintaining the original reliability of the real-time embedded application even in DVS settings. The existing schemes [22–24, 26] achieve this objective by allocating, in the offline analysis phase, a separate recovery task with any task that will be executed through voltage and frequency scaling. However, both the recovery and DVS mechanisms essentially *compete* for the available *system slack*. Our novel solution is motivated by the observation that such an explicit, static and separate recovery task allocation in the offline phase severely limits the slow-down opportunities. In fact, to maintain the task-level reliabilities, it is sufficient to ensure that, at the dispatch time of any scaled task T_i, there will be sufficient time to execute a recovery in case that T_i incurs a transient fault. Moreover, the system can even reserve the CPU time only for *a shared recovery block* that may be used by *any* task if need arises: since transient fault probabilities are typically low, any CPU time reserved but not needed for the recovery can be immediately be made available to the use of the *next* scaled task. Consequently, the SHR scheme has the potential of maximizing the slack available for DVS while still maintaining the system reliability.

It is, however, important to determine the size of the CPU time reserved for the shared recovery carefully. Let us define the available system slack as $L = D - \sum_{i=1}^{n} c_i$. Observe that any task T_j where $c_j \geq L$ cannot be managed by any RA-PM solution, since there is simply no sufficient slack for possible recovery, should T_j be scaled and subsequently fail. As a result, only the tasks in $\Gamma_1 = \{T_i \,|\, c_i < L\}$ will be managed through DVS. Further, the shared recovery should be large enough to allow the re-execution of any scaled task, suggesting that $\alpha = max\{c_i \,|\, T_i \in \Gamma_1\}$ units of CPU time should be reserved. Now, the problem of determining the processing frequency assignments for the managed tasks to minimize the energy consumption can be formulated as a convex program:

$$\text{minimize} \sum_{T_j \in \Gamma_1} E_j(f_j) \tag{5}$$

$$\text{subject to:} \sum_{T_j \in \Gamma_1} \frac{c_j}{f_j} + \sum_{T_i \in (\Gamma - \Gamma_1)} c_i + \alpha \leq D \tag{6}$$

$$f_{min} \leq f_j \leq f_{max} (\forall j : T_j \in \Gamma_1) \tag{7}$$

where the inequality (6) encodes the deadline constraint and (7) gives the available speed constraints. Since $\sum(c_i) + \alpha$ in (6) is a constant, this problem can be seen to be an instance of the system-wide energy optimization problem that can be solved in polynomial-time (for example, in time $O(n^3)$ by the algorithm given in [2]). The offline frequency assignment phase of the SHR scheme can be found in Figure 2.

The online operation of SHR is relatively simple and presented in Figure 2 as well. The tasks are executed at the precomputed and optimal frequency levels as long as transient faults do not occur. Once a fault has been detected, the recovery is executed at f_{max}. Then, since the shared recovery slack may be partially or fully used for the current period, our scheme simply disables voltage scaling and executes all the remaining tasks at f_{max} until the next frame (period) boundary. Not only does this have only a minimal impact on the expected energy savings (because the transient faults occur rarely), but also, as we formally show below, makes sure that no task reliability is worse than its original reliability level (when executed without voltage scaling). One can easily verify that the online overhead of SHR is minimal.

Offline Phase:
1. Compute the available slack $L = D - \sum_{i=1}^{n} c_i$;
2. Determine $\Gamma_1 = \{T_i | c_i < L\}$;
3. Compute the optimal frequency assignments f_i^* for all tasks $T_i \in \Gamma_1$, by solving the non-linear optimization problem given by (5), (6) and (7);
4. Set $f_i^* = f_{max}$ for tasks $T_i \in (\Gamma - \Gamma_1)$.

Online Phase:
1. At every frame (period) boundary: set *flag = true* ;
2. At dispatch time of task T_i:
. - If (*flag = true*) set the CPU frequency to f_i^*
. else set the CPU frequency to f_{max};
3. At completion time of task T_i with $f_i < f_{max}$:
. - Check whether a transient fault has occurred
. - If the fault is detected, execute its recovery
. with f_{max} and set *flag = false*.

Figure 2: Steps of the SHR scheme

Before formally proving the reliability-preserving feature of SHR, we recall from [26] that the overall reliability of a task T_i executed at frequency $f_i < f_{max}$ is given by:

$$R_i(f_i) + (1 - R_i(f_i)) \cdot R_i(f_{max}) \tag{8}$$

when a recovery task of size c_i is guaranteed to execute at f_{max} in case of a transient fault incurred by T_i. The first term in (8) is the probability that the main scaled task will complete successfully. The second term captures the probability that the recovery executed at f_{max} will succeed when the main task fails (which can occur with probability $(1 - R_i(f_i))$.

LEMMA 1. *The SHR scheme preserves the system's original reliability in every period.*

PROOF. We will prove the statement by showing that SHR preserves the original reliability of each task individually, in each period. Consider the first task T_1 executed by SHR. If $f_1 = f_{max}$, its reliability is exactly $R_1^0 = R_1(f_{max})$, hence no reliability degradation occurs. On the other hand if $f_1 < f_{max}$, its reliability is given by (8) since SHR *must have assigned a shared recovery of size at least* c_i. But the expression (8) is guaranteed to be greater than $R_i(f_{max}) = R_1^0$ [26], implying that T_1's reliability does not decrease.

For the second task T_2, if T_1 fails and the shared recovery slack is used, T_2 and all the remaining tasks are executed at f_{max} until the period boundary, which maintains the original reliability of the entire application during that period. Otherwise, if T_1 does not fail, the recovery slack is not used and the same argument can be repeated for T_2 to demonstrate that a large enough recovery slack will preserve its reliability in case that it is executed at a low frequency level. Continuing in this way, we can show that the reliability of a task set executed by SHR is never worse than its original reliability while meeting the deadline. □

3.1 Experimental Evaluation

To assess the performance of the new SHR technique on both energy efficiency and system reliability, we designed a discrete-event simulator in C programming language. The following schemes are considered in the evaluation:

- The **Greedy (GRE)** algorithm (from [23]): In this scheme, the tasks are selected for reliability-aware power management in greedy fashion. That is, for the first task, a separate recovery is allocated and its frequency is reduced as much as possible (up to the energy-efficient frequency f_{ee}). The process of selecting tasks for management continues in this manner until all available slack is depleted.
- The **SUEF** algorithm (from [23]): In this scheme, first, the *slack usage efficiency factors (SUEFs)* of all tasks are computed and the slack is allocated in decreasing order of SUEF values. The SUEF value for task T_i running at the frequency f is defined as $\frac{E_i^0 - E_i(f)}{s_i(f)}$, where E_i^0 and $E_i(f)$ are the energy consumption of task T_i at f_{max} and f, respectively, and $s_i(f)$ is the total amount of slack needed (including the slack reserved for recovery) [23].
- The **Static Power Management (SPM)** solution that computes the optimal system-level DVS slow-down factors to minimize the overall energy consumption [2]. Note that SPM neither reserves any slack for recovery nor considers reliability preservation. SPM is included in our comparison simply because it yields the maximum energy savings that can be achieved by any algorithm – reliability-aware or not.

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 [10, 28]. We model a DVS-enabled CPU where the normalized processing frequencies can change from $f_{min} = 0.1$ to $f_{max} = 1.0$. We use a cubic frequency-dependent power component P_d which is equal to unity at f_{max}. The frequency-independent power component P_{ind} for each task is normalized with respect to P_d. All results are normalized with respect to the **No Power Management (NPM)** scheme that executes all the tasks without any frequency and voltage scaling at f_{max}. Note that the reliability level achieved by NPM corresponds to the original reliability of the system (Section 2.3).

Within a task set, the worst-case number of cycles c_i for each task is randomly generated through a uniform distribution, in such a way that the worst-case execution time under maximum frequency falls in the range $[1ms, 10ms]$. Each task set contains 10 tasks. Each point in the presented plots is obtained by averaging the values obtained through 1000 different task sets.

First, we evaluate the performance of the schemes from energy consumption point of view. Figure 3 shows the relationship between the available slack and the energy consumption. In these experiments, the exponent d in the fault rate function (4) is set to 2 and the frequency-independent power is 0.05. Let $C = \sum c_i$. The available slack is represented by $L = D - C$. Naturally, the larger the slack, the more opportunities for dynamic voltage scaling and energy reduction; and all schemes achieve lower energy consumption with increasing L. We observe that the SHR scheme can achieve significant (up to 35%) energy savings over the previous RA-PM algorithms, GRE and SUEF. This is because, unlike SUEF and GRE that allocate separate recovery tasks statically, SHR reserves only a minimal amount of slack as a shared recovery and is able to allocate much larger slack for energy-efficient slow-down. Moreover, we notice that as the available slack gets larger (e.g. when $L \geq 0.7$), the performance of SHR becomes remarkably close to that of the SPM, which is the upper bound for any DVS-based energy management algorithm.

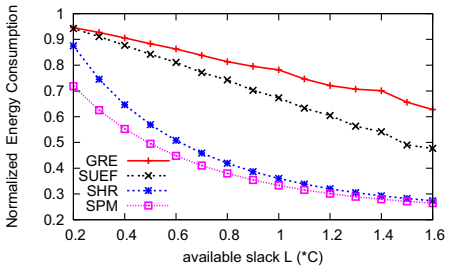


Figure 3: The impact of the available slack on system energy consumption

Figure 4 illustrates the effects of having different frequency-independent active power P_{ind} on the system energy consumption when the available slack $L = 0.8\ C$ and $d = 2$. The minimum value of P_{ind} is set to $P_{ind,min} = 0.05$. P_{ind} for each task is generated randomly and separately in the range $[P_{ind,min}, P_{ind,max}]$. The energy consumption figures of all the 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, limiting the voltage scaling opportunities (Section 2.2). Again we observe that the SHR scheme dominates over GRE and SUEF and further, its performance is extremely close to the ideal bound obtained by SPM.

Next, we investigate the system reliability dimension for these schemes. For convenience, we present the *probability of failure (PoF)* (defined as $1-$ *reliability*) achieved by all schemes. Again, all results are normalized with respect to the *PoF* achieved by NPM. Obviously, any reliability-preserving scheme should achieve a normalized *PoF* value that does not exceed 1.0. Figure 5(a) and Figure 5(b) show the reliability performance when the fault rate exponent d is 2 and 5, respectively. P_{ind} is set to 0.05 for all tasks. Consistent with previous research, the findings indicate that SPM is not able to maintain the original reliability; further, it experiences PoF increases of several orders of magnitude. As expected, SHR, GRE and SUEF are all able to maintain the system's original reliability. Moreover, the simulation results point to a somewhat counter-intuitive result: SHR has a clear advantage also on the reliability side through most of the evaluated spectrum, despite the fact that it uses a shared recovery block for all the tasks. The main reason behind this phenomenon is that GRE and SUEF choose to allocate statically the recovery tasks to individual tasks. As a result, except when the slack is very large, only a subset of tasks can be managed by SUEF and GRE while the remaining tasks do not receive any recovery. On the other hand, SHR allocates a single recovery block that can be used by *any* task in the case of a fault. In essence, SHR provides

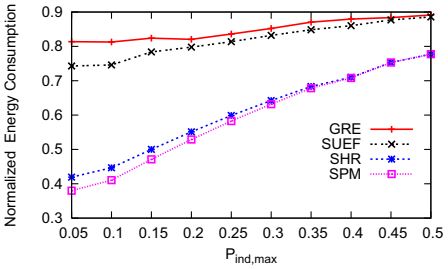
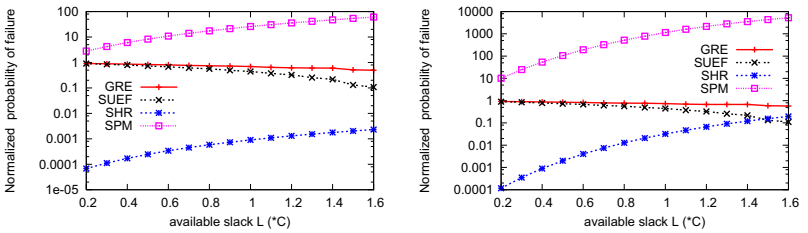


Figure 4: The impact of P_{ind} on the system energy consumption



(a) normalized probability of failure for $d = 2$

(b) normalized probability of failure for $d = 5$

Figure 5: The impact of slack on system reliability

a better protection for single-fault scenarios that are typically much more likely than some multiple-fault scenarios that SUEF and GRE provision for, at the expense of some entire tasks. With increasing slack, the *PoF* of SHR relatively increases as the system can use more aggressive voltage scaling. When $d = 5$, the fault rate is very sensitive to the voltage scaling and normalized *PoF* of SHR approaches (but never exceeds) 1.0.

4. DYNAMIC EXTENSIONS

While the basic SHR scheme provides a powerful mechanism to increase energy savings while maintaining the system reliability, some run-time optimizations can further improve the performance. In fact, there are two main opportunities that can be exploited at run-time:

- Initially, the SHR scheme is forced to reserve CPU time for the potential recovery of the largest task. Thus, as tasks complete within a frame, the system can dynamically reduce the amount of CPU time for shared recovery when faults are not encountered. For example, when the largest task completes successfully (without a fault), the system can further slow down the remaining tasks by using some part of the recovery block which is no longer needed. This can be repeated for other large tasks in the same frame.

- Typically, real-time embedded applications complete early, without consuming their worst-case number of cycles. In fact, many DVS frameworks [3, 15, 18] were proposed in the past to detect and use the *dynamic slack* that can arise from early completions to further reduce the CPU processing frequency. The same approach can be incorporated to our framework.

Thus, the **Dynamic Shared Recovery (DSHR)** technique is based on determining the initial frequency assignments as in the base SHR scheme presented in Section 3. At run-time, when a task completes early and/or without a fault, frequency assignments are updated by taking into account the time to deadline, the size of the recovery slack needed, and worst-case workload of the *remaining* tasks in that frame.

4.1 Experimental Evaluation

In this section, we evaluate the performance of the dynamic scheme experimentally. The experimental settings are essentially the same as those in Section 3.1; however, in order to model the variations in the actual workload, we use the ratio $\frac{WCC}{BCC}$, which denotes the ratio of the worst-case number of cycles (WCC) to the best-case number of cycles (BCC). For each task set, first the worst-case number of cycles for each task is determined as before. At run-time, the actual workload (number of cycles) of each task is determined between its BCC and WCC, using a uniform probability distribution. Clearly, the higher this ratio, the more the actual workload deviates from the worst-case. In the simulations, the fault rate exponent d is set to 2.

To obtain a fair comparison, though they were originally proposed only as static algorithms in [23], we extended GRE and SUEF algorithms to use dynamic reclaiming in case of early completions and not needed recovery tasks. These dynamic extensions are named as DGRE and DSUEF, respectively. Essentially, at every task completion point, GRE and SUEF are re-invoked to allow the re-adjustment of the processing frequency, just like DSHR. Finally, we implemented the scheme that invokes the static power management (SPM) algorithm to compute optimal frequency assignments using the knowledge of the *actual* workload in advance. This algorithm, named simply *Bound* in the following discussion, is not a practical scheme since it assumes the knowledge about the future workload; but its performance is used again as a natural bound on the energy consumption of any algorithm. Again, in the provided results, all results are normalized with respect to that of the NPM scheme.

First, we evaluate the energy savings of the schemes. Figure 6 shows how the energy consumption changes as a function of the available slack L when $\frac{WCC}{BCC} = 4$ and $P_{ind} = 0.05$. As before, the more slack available, the higher the energy savings. Notice that DSHR provides again clear advantages over DGRE and DSUEF. However, at very small slack values (e.g. when $L = 0.3\ C$), the difference between the clairvoyant algorithm *Bound* and DSHR is more significant since the former is able to adopt low processing frequencies with the pre-knowledge of the actual task workloads, while the latter is forced to base its offline decisions on the worst-case workload information, as every practical real-time algorithm. An interesting observation is the greatly enhanced performance of DGRE (compared to worst-case settings). In fact, when $\frac{WCC}{BCC} = 4$ not only it outperforms SUEF but also offers a performance reasonably close to DSHR. This is because, at this high workload variability settings, many tasks complete early, and as previous DVS research suggests [3,15], re-using the slack as early as possible typically pays off. This contrasts with DSUEF that chooses to re-allocate the slack according to slack usage efficiency factors, possibly excluding the next tasks that would immediate benefit. Note that when L is very large, DGRE, DSHR and *Bound* converge to

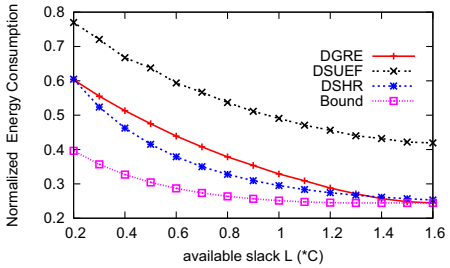


Figure 6: The impact of slack on system energy consumption

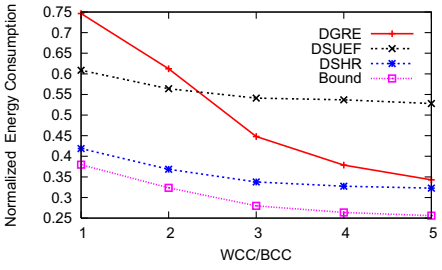


Figure 7: The impact of $\frac{WCC}{BCC}$ ratio on system energy consumption

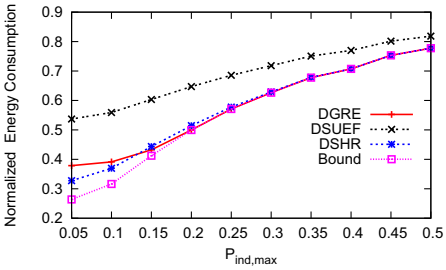


Figure 8: The impact of P_{ind} on system energy consumption

the same level since all frequencies are naturally limited by the energy-efficient frequency level.

Figure 7 illustrates the impact of the variability in the actual workload by the ratio of $\frac{WCC}{BCC}$ on the energy consumption when $L = 0.8 \cdot C$ and $P_{ind} = 0.05$. Again, as expected, the system energy consumption generally decreases with decreasing actual workloads (with increasing $\frac{WCC}{BCC}$ ratios), since more dynamic slack enables the system to scale down the frequency and then further improve the energy savings. Among all three schemes, DSHR achieves the best energy savings, which is close to the yardstick algorithm *Bound* by at most a margin of 7%. Further, we observe an interesting phenomenon: when $\frac{WCC}{BCC} \leq 2$, DGRE's performance quickly deteriorates. This is because, when the actual workload does not exhibit high variability, DSUEF's approach of assigning slack according to slack usage efficiency factors yields typically better results, as opposed the DGRE's strategy that consists in greedily making available the slack to the next task. However, with increasing $\frac{WCC}{BCC}$, DGRE performs better and better due to higher dynamic slack resulting from tasks' earlier completions.

Figure 8 shows the effects of frequency-independent active power on the system energy consumptions for $L = 0.8 \cdot C$ and $\frac{WCC}{BCC} = 4$. We observe that energy consumptions increase with increasing the value of $P_{ind,max}$ and that DSHR, DGRE and *Bound* converge to the same value when $P_{ind,max} \geq 0.2$. Again, the main reason is that higher P_{ind} values result in significantly high energy-efficient frequency values that force the system to run at high frequencies regardless of the actual workload.

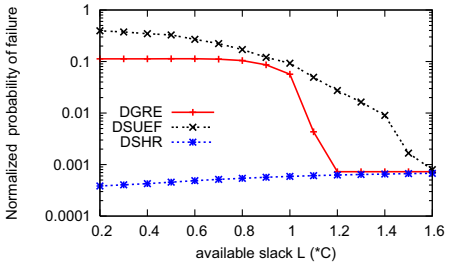


Figure 9: The impact of the system slack on the probability of failure

Next, the reliability performance of the schemes is evaluated. Figure 9 shows the impact of available slack on the probability of failure when $P_{ind} = 0.05$ and $\frac{WCC}{BCC} = 5$. We observe that, while DSHR is the best, now DGRE and DSUEF greatly benefit from higher slack values. Essentially, DGRE and DSUEF can allocate more recovery tasks dynamically by recycling the slack of unused recoveries at run-time, which is reflected in PoF values. Another interesting observation is that, just like the energy dimension, DGRE appears to outperform DSUEF because of re-assigning recovery tasks in greedy fashion. Also note that when $L \geq 1.1$, once again almost all the tasks are effectively executed at the energy-efficient frequencies by DGRE and DSHR, which gives similar PoF values.

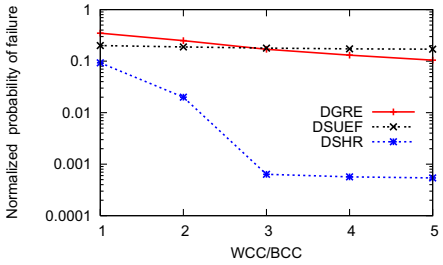


Figure 10: The impact of $\frac{WCC}{BCC}$ ratio on the probability of failure

Figure 10 illustrates the impact of the variability in the actual workload on the system reliability when $L = 0.8 \cdot C$ and all P_{ind} values are set to 0.05. In general, with fixed available slack, the probability of failure for all schemes decreases as we reduce the actual workload (i.e. as we increase the ratio of $\frac{WCC}{BCC}$) since more slack becomes available due to early completions of tasks. Because of its greedy nature (i.e. aggressive allocation of dynamic slack to the next task), DGRE's reliability tends to degrade with small $\frac{WCC}{BCC}$ ratios. Once $\frac{WCC}{BCC} \geq 3$, DSHR can execute almost all the tasks at the energy-efficient frequencies and also almost every task can have a recovery to be executed if a fault occurs.

5. CONCLUSIONS

In this paper, we proposed and evaluated a new RA-PM scheme, called the shared recovery (SHR) technique. In the offline phase, SHR allocates CPU time only for the largest recovery that may be needed by any task, essentially making available to the DVS mechanism the valuable slack that would be typically reserved for the recoveries of separate tasks in the previous RA-PM solutions. The energy savings of the new scheme are shown to approach those of the optimal (but not reliability-aware) DVS solutions. Further,

SHR is formally shown to preserve the original system reliability. The experimental evaluation indicates that SHR has also a clear advantage on the reliability dimension for most of the spectrum that we considered. Finally, we proposed a dynamic extension of the scheme, called DSHR, that is able to reclaim any dynamic slack that results from early completions or not needed recovery operations. This dynamic reclamation enables the system to further improve the energy savings, effectively approaching the performance of a potentially clairvoyant algorithm.

6. REFERENCES

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

[2] H. Aydin, V. Devadas, and D. Zhu. System-level energy management for periodic real-time tasks. In *Proc. IEEE Real-Time Systems Symposium (RTSS)*, 2006.

[3] 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.

[4] T. Burd and R. Brodersen. Energy efficient cmos microprocessor design. In *Proc. Hawaii International Conference on System Sciences (HICCS)*, 1995.

[5] 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.

[6] 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.

[7] E. M. Elnozahy, M. Kistler, and R. Rajamony. Energy-efficient server clusters. In *Proc. Workshop of Power Aware Computer Systems (PACS)*, 2002.

[8] 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.

[9] X. Fan, C. Ellis, and A. Lebeck. The synergy between power-aware memory systems and processor voltage. In *Proc. Workshop of Power Aware Computer Systems (PACS)*, 2003.

[10] 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.

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

[12] R. K. Iyer and D. J. Rossetti. A measurement-based model for workload dependence of cpu errors. *IEEE Trans. on Computers*, 33(6):518–528, 1984.

[13] 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.

[14] T. Pering, T. Burd, and R. Brodersen. The simulation and evaluation of dynamic power-aware scheduling for real-time applications. In *Proc. International Symposium on Low Power Electronics and Design (ISLPED)*, 2000.

[15] P. Pillai and K. G. Shin. Real-time dynamic voltage scaling for lowpower embedded operating systems. In *Proc. ACM Symposium on Operating Systems Principles (SOSP)*, 2001.

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

[17] 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.

[18] S. Saewong and R. Rajkumar. Practical voltage scaling for fixed priority rt-systems. In *Proc. IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2003.

[19] M. Weiser, B. Welch, A. Demers, and S. Shenker. Scheduling for reduced cpu energy. In *Proc. USENIX conference on Operating Systems Design and Implementation (OSDI)*, 1994.

[20] F. Yao, A. Demers, and S. Shenker. A scheduling model for reduced cpu energy. In *Proc. Annual Symposium on Foundations of Computer Science (FOCS)*, 1995.

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

[22] D. Zhu. Reliability-aware dynamic energy management in dependable embedded real-time systems. In *Proc. IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2006.

[23] 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.

[24] D. Zhu and H. Aydin. Reliability-aware energy management for periodic real-time tasks. In *Proc. IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2007.

[25] 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.

[26] D. Zhu, R. Melhem, D. Mossé, and E. Elnozahy. Analysis of an energy efficient optimistic tmr scheme. In *Proc. International Conference on Parallel and Distributed Systems (ICPADS)*, 2004.

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

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