Energy-Aware Task Replication to Manage Reliability for Periodic Real-Time Applications on Multicore Platforms

Mohammad A. Haque, Hakan Aydin
Department of Computer Science
George Mason University
Fairfax VA 22030
mhaque4@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*—Energy and reliability management are important design constraints for real-time embedded systems. We consider the problem of achieving a given reliability target for a set of periodic real-time applications running on a multi-core system with minimum energy consumption. Specifically, we observe that the emerging multicore platforms provide ample opportunities to use task replication to achieve reliability targets and mitigate the negative impact of Dynamic Voltage Scaling (DVS) on the rate of transient faults leading to soft errors. However, while it allows using lower execution frequencies, replication may also increase overall energy consumption due to additional task copies. Our objective is to determine the level of replication and frequency assignment for each task, as well as task-to-core allocations, in such a way to achieve the target reliability levels with minimum energy consumption. We first identify the subtle interplay between the processing frequency, replication level, reliability, and energy consumption on DVS-enabled multicore systems. Then we show that the problem is intractable in the general case and propose our energy-efficient replication (EER) algorithm as an approximate solution. We also show how the framework can be extended to tolerate a given number of permanent faults affecting processing cores. We evaluate the performance of our proposed scheme through extensive simulations. The simulation results indicate that through our algorithm, a very broad spectrum of reliability targets can be achieved with minimum energy consumption through the judicious use of replica and frequency assignment.**

I. Introduction

Energy management is an important design constraint for any computer system. Apart from high energy costs, higher energy consumption results also in adverse environmental effects. Two very populer techniques are *Dynamic Power Management (DPM)* and *Dynamic Voltage Scaling (DVS)*. With DPM, system components are put to sleep/low-power states when they are idle [1], [4]. The main idea of DVS is to reduce the CPU voltage and frequency to achieve energy savings [20]. However, for real-time embedded systems, the applicability of DVS is limited by the timing constraints associated with the tasks. Recent studies also explored integrating DPM and DVS in optimal ways [5], [24].

Computer systems are susceptible to *faults*, leading to various *run-time errors*. Faults can be broadly categorized into two types: *transient* and *permanent* faults [14], [17]. Transient faults are much more common in practice, as they are caused by electromagnetic interference or cosmic radiation [3], [11]. Transient faults may manifest in the form of *soft errors* with incorrect computation results. Since transient faults are non-persistent, re-execution of the affected task or invocation of an alternate task are commonly used recovery techniques [14], [17]. Permanent processor faults, on the other hand, are caused by manufacturing defects or circuit wear-out. Permanent faults therefore can only be dealt with through *hardware redundancy* techniques (e.g., by deploying extra processors) in the system. Real-time systems are often deployed for safety-critical applications, where tolerance to faults is extremely important. However, energy management and fault tolerance are in general conflicting system objectives as the latter requires deployment of extra resources [6], [10] [16], [19], [22].

The consideration of reliability for systems employing DVS for energy is equally important, as the current research suggests the negative impact of DVS on the transient fault rate [7], [26]: as we decrease the supply voltage and frequency to save power, the transient fault rate (and the corresponding soft errors) increase exponentially. Therefore, energy management techniques must take into account the reliability degradation and make provisions accordingly. A set of techniques, called the *Reliability-Aware Power Management (RAPM)* [23], [27], exploit *time redundancy* available in the system for both energy and reliability management. These works consider the problem of preserving the system's *original reliability*, which is the reliability of the system executing tasks without any slowdown. The scheme in [27] resorts to *backward recovery* approach and assigns a recovery task for every task that has been slowed down, while the technique in [23] uses recovery blocks shared by all the scaled tasks.

More recently, the *reliability-oriented energy management* framework has been proposed for periodic real-time tasks running on a single processor [25]. The main objective is to achieve *arbitrary reliability levels*, in terms of tolerance to transient faults, with minimum energy consumption. Unlike RAPM studies, the target reliability level can be *lower* or *higher* than the original reliability. This flexibility is important,

978-1-4799-0623-9/13/$31.00

as some high-criticality tasks may require very high-levels of reliability, while for some other tasks a modest reliability degradation may be acceptable to save energy. The solution still focuses on single-processor systems and hence resorts to a limited number of shared recovery jobs that are invoked upon the detection of soft errors.

The main proposal of this paper is a *reliability-oriented energy management* framework for *multicore systems*. In the last decade, due to the advances in the CMOS technology, we witnessed the proliferation of systems with multiple processing cores. Systems with 2-4 cores are commonplace, and new systems such as Intel SCC (48 cores) [30] and Tile64 (64 cores) [31] are receiving increasing attention. We observe that on these emerging *many-core* systems, *task replication* is likely to become a quite viable option for reliability management. By scheduling multiple copies of the same task on multiple cores, the likelihood of completing at least one of them successfully (i.e., without encountering transient faults) increases significantly. Replication has several advantages as an effective reliability management tool. Firstly, very high reliability targets can only be achieved through task replication. Secondly, replication has the potential of tolerating permanent faults in addition to improving reliability in terms of tolerance to transient faults. Thirdly and arguably most importantly, it creates an additional and powerful dimension to reduce the energy consumption by executing multiple copies at *lower* frequencies, while achieving the same reliability figures.

In our framework, we consider the energy-efficient replication problem for preemptive, periodic applications running on a many- or multi-core system. Our setting is *reliability-oriented* in the sense that we consider minimizing energy to meet arbitrary task-level or system-level reliability targets. Specifically, for a given reliability target, our goal is to find the *degree of replication* (number of copies) and the *frequency assignment* for all tasks, such that the overall energy consumption is minimized, while ensuring that all timing constraints will be met. The main contributions of the paper can be summarized as follows:

- We present an extensive analysis to show the viability of replication as a tool for joint management of reliability and energy,
- We formulate the *Generalized Energy-Efficient Replication Problem (GEERP)* and show its intractability,
- We propose an efficient and approximate solution to GEERP and evaluate its performance through extensive simulations,
- We also outline how our framework can be extended to provision for *permanent faults* affecting at most Z processing cores.

The rest of the paper is organized as follows. In Section II, we present our workload, power, and reliability models. In Section III, we illustrate the applicability of replication as a tool for energy and reliability management on multicore systems. Then we first address the energy-efficient replication problem in the context of a single application in Section IV. In Section V, we present our proposed solution for the generalized settings with multiple tasks. The experimental evaluation is presented in Section VI. In Section VII, we show how our framework can be extended to tolerate permanent faults of up to Z processing cores. Finally, in Section VIII, we conclude.

II. System Model

A. Workload and Processor Model

We consider a set of N periodic real-time tasks $\Psi = \{\tau_1, ..., \tau_N\}$. Each task τ_i has worst-case execution time c_i under the maximum available CPU frequency f_{max}. τ_i generates a sequence of *task instances* (or, *jobs*) with the period of P_i. The relative deadline of each of these jobs is equal to the period value P_i, in other words, each job must complete by the arrival of the next job of the same task. The *utilization* of task τ_i, u_i, is defined as $\frac{c_i}{P_i}$. The *total utilization* U_{tot} is the sum of all the individual task utilizations.

The workload executes on a set of M identical cores. Each core can operate at one of the K different frequency settings ranging from a minimum frequency, f_{min} to f_{max}. We denote by F the set of available frequency settings. Without loss of generality, we normalize the frequency levels with respect to f_{max} (i.e., $f_{max} = 1.0$). At frequency f, a core may require up to $\frac{c_i}{f}$ time units to complete a job of task τ_i.

Our framework assigns k_i replicas (copies) to task τ_i, to execute on $k_i \leq M$ distinct processing cores, to achieve reliability targets. The entire set of $\sum k_i$ task copies are partitioned upon the multicore system. The preemptive Earliest-Deadline-First policy, which is known to be optimal on single processor [15], is adopted to execute tasks on each core.

B. Power Model

We consider a power model adopted in previous reliability-aware power management research [18], [25], [27]. The power consumption of each core consists of static and dynamic power components. The static power P_s is determined mainly by the leakage current of the system. The dynamic power P_d includes a frequency-dependent power component (determined by voltage and frequency levels), and a frequency-independent power component P_{ind}, driven by the modules such as memory and I/O subsystem in the active state. In DVS technique, the supply voltage is scaled in almost linear fashion with the processing frequency. Consequently, the power consumption of a core can be approximated by:

$$P_{active} = P_s + P_{ind} + C_e f^3 \quad (1)$$

Above, C_e is a system-dependent constant, reflecting the effective switching capacitance. When a core is not executing any task (idle state), its power consumption is primarily determined by the static power. We assume that the static power consumption can only be eliminated by the complete power-down of the core.

Existing research indicates that arbitrarily slowing down a task is not always energy-efficient [8], [12], [28], due to the frequency-independent power components. In other words, there is a processing frequency below which the total energy

consumption increases. This frequency is called the *energy-efficient frequency* and denoted by f_{ee}. f_{ee} can be computed analytically through the well-known techniques [12], [26].

C. Fault Model

Transient faults are typically modelled using an exponential distribution with an average arrival rate λ [21]. The fault rate λ increases significantly as frequency is scaled down when using DVS [7], [26]. The average fault rate at the maximum frequency is denoted by λ_0. The fault rate at frequency f can be expressed as [26]:

$$\lambda(f) = \lambda_0 10^{\frac{d(1-f)}{1-f_{min}}} \tag{2}$$

Above, the exponent d, called the *sensitivity factor* in the paper, is a measure of how quickly the transient fault rate increases when the system supply voltage and frequency are scaled. Typical values range from 2 to 6 [18], [26], [27].

The *reliability* of a task is defined as the probability of executing the task successfully, without the occurrence of transient faults [27]. The reliability of a single instance of task τ_i running at frequency f_i can then be expressed as [26], [27]:

$$R_i(f_i) = e^{-\lambda(f_i)\frac{c_i}{f_i}} \tag{3}$$

Conversely, the *probability of failure (PoF)* of a task instance of τ_i is denoted by:

$$\phi_i(f_i) = 1 - R_i(f_i) \tag{4}$$

At the end of execution of each task copy (replica), an *acceptance test* (or, *sanity check*) [14] [17] is conducted to check the occurrence of soft errors induced by the transient faults. If the test indicates no error, then the output of the task copy is committed to; otherwise, it is discarded. Therefore, in replicated execution settings of a given task, it is sufficient to have at least one task copy execution that passes the acceptance test.

III. Interplay of Energy, Reliability, Frequency, and Replication

Before presenting our detailed analysis, we start by illustrating how the level of replication and processing frequency jointly determine the reliability and energy savings. Consider a single instance of task τ_i running at frequency f_i. Its probability of failure is given by Equation (3) and is a function of the processing frequency f, λ_0 and the sensitivity factor d. Now consider two replicas of τ_i running at frequency f. The execution with replication will be unsuccessful only if both replicas encounter transient faults during their respective executions. Consequently the new reliability with two replicas is found as:

$$R_i' = 1 - (1 - R_i(f))^2$$

More compactly, its new probability of failure is given by:

$$\phi_i^{(2)} = (\phi_i(f))^2$$

In general, with k replicas, the probability of failure decreases exponentially with k:

$$\phi_i^{(k)} = (\phi_i(f))^k \tag{5}$$

We observe that the use of replication may be a powerful tool to mitigate the negative impact of the voltage/frequency scaling on the probability of failure, and improve energy savings through parallel execution. However, there are several non-trivial design dimensions that must be considered, including the energy cost of additional replica execution(s). As a concrete example, consider a single task with worst-case execution time $c_i = 100$ ms. Following [25], in this and subsequent examples used in the paper, we assume that transient fault arrival rate at the maximum frequency is $\lambda_0 = 10^{-6}$, and the system sensitivity factor d is 4.

Figure 1 shows how the execution frequency determines the achievable *PoF*, for different number (k) of replicas. The *PoF* values are given in normalized form, with respect to the probability of failure of a single copy running at maximum frequency. Note that the $y-$axis in the plot is in logarithmic scale. Hence, for a given number of replicas, the *PoF* increases (the reliability decreases) rapidly with decreasing frequency. However, for a given frequency, there is also an exponential improvement on reliability (decrease in *PoF*), with increasing number of replicas. This simple fact points to an interesting design spectrum: the same target *PoF* value can be achieved at different frequency and replication levels. For example, the *PoF* value achieved by 2 replicas of the same task running at f_{max} can be also yielded by 3 replicas running at the low frequency $f = 0.2$. Therefore, along with the frequency, adjusting the number of replicas is clearly an additional and powerful mechanism to achieve the target *PoF* figures.

We note that there is a lower bound on the number of replicas required to achieve a certain *PoF* target, ϕ_{target}. In particular, high reliability levels necessitate the use of multiple replicas. Using Equation (5), we can easily determine the minimum number of replicas (k) needed to achieve the target *PoF* at a given frequency level f:

$$\phi_{target} \leq (\phi_i(f))^k$$

$$k \geq \left\lceil \frac{\log(\phi_{target})}{\log(\phi_i(f))} \right\rceil \tag{6}$$

On the other hand, the energy consumption of different replication/frequency levels yielding the same reliability level may vary significantly. Figure 2 shows the impact of replication on energy consumption for our example task, under various target (normalized) *PoF* values. In these experiments, for a given target *PoF* value ϕ_{target} and for each replication level (k), we compute the minimum energy consumption when we use the best common frequency f for all k replicas to achieve ϕ_{target}. The energy consumption is normalized with respect to a single replica running at the maximum frequency[1].

[1]We assume $P_{ind} = 0.1$ in the examples.

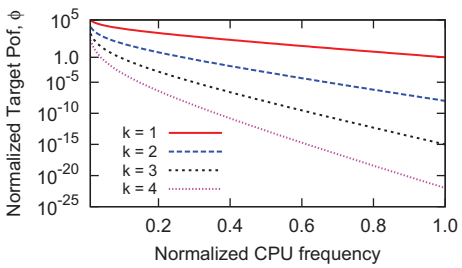


Fig. 1: Impact of the frequency on reliability

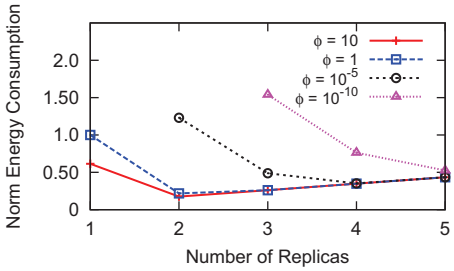


Fig. 2: Impact of replication level on energy

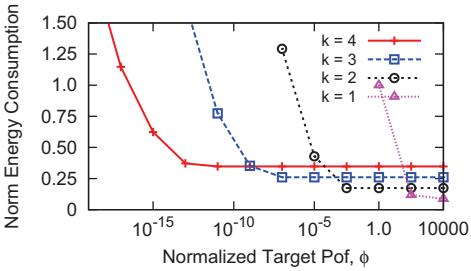


Fig. 3: Impact of target *PoF* on energy

A few key observations are in order. First, some very high reliability (very low *PoF*) targets can only be achieved with large number of replicas. Second, for a fixed reliability target, using a minimum number of replicas generally consumes high energy. This is due to the fact that the minimum number of replicas are typically executed at high frequency levels to meet the reliability target ϕ_{target}, giving high energy figures. On the other hand, as we start increasing the number of replicas, we can afford reducing the frequency of individual replicas and thus the total energy consumption starts to decrease. But as we deploy further replicas, after a certain point, the energy consumption due to additional replicas offsets the energy savings due to execution at low frequencies. As a result, the energy consumption starts increasing. This is also coupled by the fact that reducing the frequency below the energy-efficient frequency f_{ee} is not helpful, even if we can use additional replicas. Consequently, the energy consumption figures continue to increase beyond a certain threshold point. Finally, it is clear that the optimal number of replicas to minimize energy depends highly on the target reliability.

Figure 3 depicts the interplay of target probability of failure and energy consumption. Notice that, as we increase the target *PoF*, the total energy consumption decreases for a fixed number of replicas, as we are more tolerant of reliability degradation and we can afford running the replicas at lower frequencies. Once the frequency required to achieve the reliability target reaches the energy-efficient frequency, the energy curve flattens out. At that point, increasing the target probability of failure does not yield any energy savings. Moreover, since the *PoF* figures achievable by k replicas is a proper subset of those achievable by $k' > k$ replicas, one must consider the minimum number of cores needed for the ϕ_{target} under consideration – in our example, ϕ_{target} values smaller than 10^{-7} were simply not achievable by deploying 1 or 2 replicas.

We can summarize our key observations in this section as follows:

- In addition to the processing frequency, the replication can be used to manage reliability on DVS-enabled multicore systems, giving a broad design spectrum involving multiple dimensions, and in particular, *energy consumption*.
- The same target reliability can be typically achieved through multiple ways: using small number of replicas running at high frequencies or large number of replicas running at low frequencies. While the use of replication allows the system to use lower frequencies to mitigate the reliability loss, this may have a negative impact on the energy consumption due to the energy cost of running additional replicas. The configuration that minimizes energy consumption varies depending on the system parameters and target reliability.

IV. Task-Level Analysis

We first address our problem of finding the energy-optimal configuration (i.e., the number and frequency assignment of replicas) in the context of a single task to achieve a target reliability level. This single-task setting enables us to illustrate some additional non-trivial aspects of the problem as well as to present the terminology and definitions that will be instrumental for the eventual system-level analysis that will be carried out in Section V.

We start by observing that, for a given execution frequency, the minimum number of replicas needed is provided by Equation (6). Moreover, the number of available frequency levels is very limited in existing processors; for example, Intel Pentium M processor with 1.6 GHz maximum frequency supports only six frequency steps in the active state [29]. Consequently, one can easily compute the number of replicas needed, as well as the corresponding overall energy consumption for every available frequency setting. An obvious question is whether the energy numbers obtained as a function of decreasing replica frequency exhibit a certain (e.g., convex or concave) pattern. Unfortunately, the answer is negative, as the following example illustrates.

Example 1. Consider a task with execution time $c_i = 1000$ ms at the maximum frequency. The normalized *PoF* target for the task is given as 10^{-6}, requiring definitely more than one task copy. We assume that the system has 4 (four) normalized frequency settings, given as 0.4, 0.6, 0.7, and 1.0. For illustration purposes, we assume the frequency-independent power (P_{ind}) is set to 5% of the maximum dynamic power and the static power is negligible.

For each of the available frequency levels, we can compute the minimum number of replicas for the given reliability target and the corresponding overall energy consumption, obtaining the values shown in Table I.

TABLE I: Total Energy Consumption

Frequency	Replica	Energy
1.0	2	2.1
0.7	3	1.68429
0.6	4	1.77333
0.4	6	1.71

We notice that, as we decrease the CPU frequency the overall energy consumption of all the required replicas first decreases, then increases, before decreasing again. This suggests that the overall reliability-oriented energy consumption function, as a function of processing frequency, is neither convex nor concave. Consequently, the use of simple techniques like binary search or the techniques available from the convex optimization theory would not be applicable. This small example shows that, in the general case, one may need to consider the energy consumption for all the frequency levels to find the optimal configuration.

A. Energy-Frequency-Reliability (EFR) Tables

In general, as in Example 1, we can construct for every task τ_j a table with K rows on a system with K frequency levels. For each frequency setting, we compute the minimum number of replicas (through Equation (6)) to achieve its target reliability $\phi_{j,target}$ and the corresponding overall energy consumption. In preparation for our system-level analysis, we also include a column that indicates the total CPU time needed by all the replicas.

The entries given in each row corresponds to a separate *configuration* of the system. We denote the i^{th} configuration for a task by *RfConfig(i)*, which contains information about the frequency of each replica, number of replicas, total energy consumption, and total CPU time needed by all replicas. This *Energy-Frequency-Reliability (EFR)* table can be clearly constructed in time $O(K)$ for a given task.

Table II shows an example EFR table for a task whose execution time is $c = 100$ ms. The normalized *Pof* target is 10^{-6}. For simplicity, assume P_s and P_{ind} are negligible. Assuming the processor has 10 different frequency steps, we have 10 different configurations (*RfConfig(i)* $i = 1, \ldots, 10$).

TABLE II: An Example EFR Table

Frequency, f	Replica # r	Energy	CPU time
1	2	0.2	0.2
0.9	2	0.162	0.222222
0.8	*3*	*0.192*	*0.375*
0.7	3	0.147	0.428571
0.6	3	0.108	0.5
0.5	3	0.075	0.6
0.4	4	0.064	1
0.3	4	0.036	1.33333
0.2	5	0.02	2.5
0.1	6	0.006	6

Note that, in general, the number of *valid* frequency levels may be smaller than the number of available frequency levels. This is because frequencies below the energy-efficient frequency f_{ee} should not be considered. In addition, since the task τ_j would miss its deadline at the frequency levels below its utilization value of $u_j = \frac{c_j}{P_j}$, we do not need to consider frequencies $< max\{f_{ee}, u_j\}$.

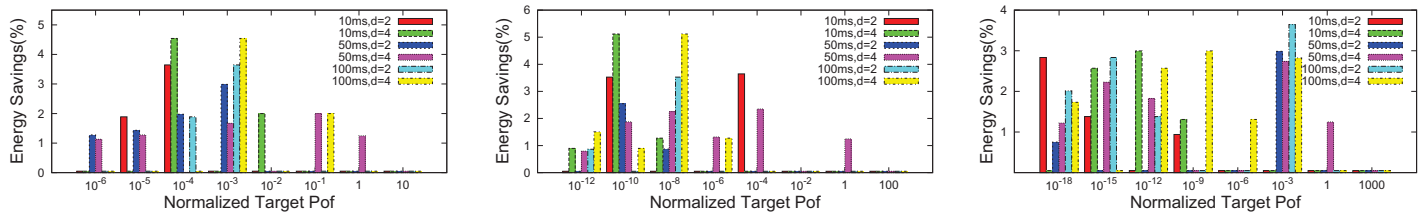
As we decrease the CPU frequency, the time required for executing each replica increases. The number of required replicas may remain the same or increase. As a result, the total CPU time consumption keeps increasing. However, in terms of energy consumption patterns, the trends are not always obvious. For example, looking at Table II and comparing the entries for $f = 0.9$ and $f = 0.8$, we observe an interesting phenomenon. Specifically, the energy consumption at the lower frequency configuration $f = 0.8$ is *higher* than that of the higher frequency configuration $f = 0.9$. Obviously, there is no benefit in using the frequency $f = 0.8$ as it consumes more energy while also using more CPU time (potentially affecting the feasibility of other tasks that may exist in the system), compared to the adjacent higher frequency level $f = 0.9$.

Clearly such *inefficient* frequency levels can be also removed from the table in a linear pass. Considering that the frequency levels below $max\{f_{ee}, u_j\}$ are not valid either, the trimming of the entire table can be achieved $O(K)$ time. After such a trimming, the energy consumption values in the table will be in decreasing order and the minimum energy configuration for the task will be at the last row of the table. While choosing this minimum energy configuration is ideal for the task, the feasibility and reliability requirements of other tasks may not allow the use of this frequency. This issue will be further analyzed in Section V.

B. Impact of Frequency Assignment to Replicas

The reader may have observed that, so far, we implicitly assumed *uniform* frequency assignment to the replicas of a given task in reliability-oriented energy management. In fact, in general, assigning uniform frequencies is not always optimal. This is due to the nature of the reliability function given through Equations (2) and (3), which is neither convex nor concave in the entire spectrum. As we decrease the frequency, task reliability first decreases slowly, then it falls very sharply, which is followed by a region where the reliability decreases very slowly again. Thus, occasionally, it is possible to achieve a certain reliability target with one replica running at a relatively high frequency and another replica running at a very low frequency. Moreover, using a uniform frequency for that specific setting may violate the reliability constraint or result in higher energy consumption. In those settings, uniform speed assignment provides a sub-optimal solution. We illustrate this point with a concrete example.

Example 2. Consider a task with 10 ms execution time at maximum frequency on a 2-core system. We will execute two replicas of the task on two cores. Static power and the frequency-independent power are set to 5% of the maximum dynamic power. The target *PoF* is set to the *PoF* achieved by two replicas of that task running both at 0.75. Therefore, we can trivially satisfy the reliability target by executing both



(a) 2 replicas (b) 3 replicas (c) 4 replicas

Fig. 4: Energy savings obtained by exhaustive search compared to the uniform frequency assignment

replicas at speed 0.75. However, an exhaustive search suggest that, if we assign frequency for the two replicas to 0.9 and 0.37 respectively, the energy consumption will be 4.55% lower than the uniform setting while achieving the same reliability. Moreover, using the uniform frequency which is the mean of the indicated distinct frequencies (i.e., $f = \frac{0.9+0.37}{2} = 0.635$) would *not* satisfy the reliability target.

However, using an uniform speed for all the replicas avoids further complications on the problem which has already several non-trivial aspects. In addition, we ran extensive simulations under various system parameters. Our results suggest that, for realistic system settings, using uniform frequency asssignment almost always yields energy figures which are quite comparable to those provided by a computationally very expensive, but optimal, algorithm.

To give the details of our experiments, for a fixed reliability target, we first find the most energy-efficient setting where all replicas run at uniform frequency. Note that, for a single task, the number of replicas cannot exceed the available number of cores. The optimal search for frequency assignment technique, on the other hand, performs an exhaustive search for every degree of replication and varies the frequency separately for each replica at increments of 0.01 from $0.1(f_{min})$ to 1.0 (f_{max}). At the end of this exhaustive search, the optimal scheme chooses the best solution.

Figures 4a, 4b and 4c present a subset of our simulation results. In these experiments, we consider task execution times of 10, 50 and 100 ms. We also vary the sensitivity factor d of the system from 2 to 4. In Figure 4a we consider the degree of replication to be 2 and for a wide spectrum of reliability targets, we report the energy savings obtained by the optimal (exhaustive) search technique over the uniform frequency assignment technique. In Figures 4b and 4c we repeat the experiments for 3 and 4 replicas respectively. The experimental results suggest only up to 5% energy savings obtained by the exhaustive search technique, compared to the uniform assignment technique. In fact, for many combinations, the difference is even smaller than 5%. As we observed the same patterns in our additional experiments, in the rest of the paper, we employ the uniform frequency assignment technique due to its conceptual and practical simplicity.

V. System-level Analysis and Solution

In this section, we address the system-level problem that involves the consideration of all the tasks in the system, each with potentially different reliability targets. We first give additional details necessary to formulate and manage reliability of periodic tasks each with multiple task instances (jobs).

A. Reliability Formulation for Periodic Tasks

We consider a reliability formulation similar to the one used in [25]. The reliability of a periodic task is defined as the probability of successfully executing all instances of that task during the hyperperiod, which is defined as the least common multiple of all the periods. Specifically, if the task τ_i has h_i instances in the hyperperiod, the *PoF* of the task can be expressed as:

$$\phi_i = 1 - \Pi_{j=1}^{h_i}(1 - \phi_{i,j}) \tag{7}$$

where $\phi_{i,j}$ denotes the reliability of the j^{th} instance of task τ_j.

The *system reliability* is the probability of executing all instances of all the tasks successfully. Therefore, it can be easily computed as the product of individual task reliabilities. The system *PoF* is then given by:

$$\phi_{syst} = 1 - \Pi_{i=1}^{N}(1 - \phi_i)$$

In reliability-oriented energy management problem, the task-level reliability targets may be given as part of the problem input. However, if only the system-level target reliability is given, we first need to compute the task level reliability target from the given system level reliability target ϕ'_{syst}. In this case, we can use the technique called the *Uniform Reliability Scaling* in [25]. This technique scales up or down all original task reliabilities by the same factor to achieve the new system-level target reliability. Specifically, assume that when all instances of a periodic task during the hyperperiod are executed at f_{max} and there are no additional replicas, the task level *Pof* is $\hat{\phi}_i$. Then, the task-level target $\phi_{i,target}$ values are determined such that $\phi'_{syst} = 1 - \Pi_{i=1}^{N}(1 - \phi_{i,target})$, and

$$\forall_i \frac{\phi_{i,target}}{\hat{\phi}_i} = \omega \tag{8}$$

Above, ω is called the *(uniform) PoF scaling factor*. Clearly, small (large) ω values correspond to higher (lower) reliability objectives.

B. Problem Definition

We now address the problem for a generalized setting, where there are multiple periodic tasks in the system. With multiple tasks, feasibility (deadline guarantees) becomes a major concern. Therefore, many tasks cannot be scheduled according to the most energy-efficient configuration from the EFR table since the total CPU time of all the replicas may exceed the time available on existing number of cores. Consequently, some tasks may have to be executed in a different 'configuration'. Given the EFR tables, determining the configuration (i.e., the degree of replication and frequency assignment) for each task such that the overall energy consumption is minimized while meeting the reliability target is a non-trivial problem.

Generalized Energy-Efficient Replication Problem (GEERP): Given a set of periodic tasks and task-level reliability targets, determine the number of replicas to execute and the frequency assignment for each replica such that the energy consumption is minimized, while ensuring a feasible partitioning such that the deadline constraints are met and two replicas of the same task are not assigned on the same core.

To present the general optimization problem formulation, we first introduce the necessary notation. Let k_i be the number of replicas assigned to task τ_i and $E_i(f_i)$ be the energy consumption for each replica of τ_i running at frequency f_i. Γ_m denotes the set of all tasks for which a replica is assigned to core m. $\rho(i,j)$ represents the core where the j^{th} replica of τ_i is assigned. Then our problem is to find k_i and f_i values along with the replica-to-core allocation (partitioning) decisions $\{\rho(i,k_i)\}$, $i = 1,\ldots,N$, so as to:

$$\text{minimize} \quad \Sigma_{i=1}^{N} k_i \times E_i(f_i) \tag{9}$$

$$\text{subject to} \quad \forall_i \; f_i \; \in F \tag{10}$$

$$\forall_i \; k_i \; \leq M \tag{11}$$

$$\forall_m \; \Sigma_{\tau_i \in \Gamma_m} \frac{c_i}{f_i} \; \leq \; 1 \tag{12}$$

$$\forall_i \; (\phi_i(f_i))^{k_i} \; \leq \; \phi_{i,target} \tag{13}$$

$$\forall_i \forall_{j \neq k} \; \rho(i,j) \; \neq \; \rho(i,k) \tag{14}$$

Above, the constraint (10) ensures a legitimate frequency assignment for every task and the constraint (11) enforces that the number of replicas does not exceed the available number of cores. The constraint (12) ensures that a feasible partitioning is obtained for all the cores, using the well-known schedulability condition with preemptive EDF [15]. The constraint set (13) represents the task level reliability targets. Finally, the constraint (14) ensures that no two replicas of a same task are assigned to the same core.

In Section IV, we observed that the overall replica energy consumption function $k_i \times E_i(f_i)$ is neither convex nor concave. Therefore no standard optimization technique can be applied to solve this problem. Moreover, the problem can be easily shown to be NP-hard in the strong sense. If we consider the special case of tasks with identical periods (deadlines), target reliabilities equal to the original reliability levels (requiring only one copy of each task), and a system without any DVS capability (where all tasks are executed at constant speed), GEERP reduces to the problem of packing variable-size items on M bins – this is the classical bin-packing problem, which is known to be NP-hard in the strong sense [9].

We consider a two-step solution for GEERP. In the first step, we construct the EFR tables for all tasks, separately. In the second step, using the tables, we search for a solution configuration that can be feasibly partitioned, while obtaining as much energy savings as possible. Due to the intractability of the problem, we resort to an efficient heuristic-based solution that still satisfies all the constraints of the problem.

C. Algorithm Energy-Efficient Replication (EER)

In this section, we present our solution. First, using Equation (7), the algorithm first determines the job level reliability target for each periodic task, given the task-level reliability targets. Then as described in section IV-A the EFR tables are constructed. Assume that, the j^{th} configuration of τ_i is denoted by *RfConfig(i,j)*. In the rest of the paper, *f(i,j)*, *r(i,j)*, *E(i,j)* and *S(i,j)* denote respectively the frequency, the number, total energy consumption, and total CPU time of all replicas in *RfConfig(i,j)*. The specific quantities for *RfConfig(i,j)* can be obtained from the j^{th} row of the corresponding EFR table. From the tables, the algorithm first determines the minimum energy configuration for a given task.

The algorithm then tries to partition the workload for various configurations on M cores. We choose the *First-Fit-Decreasing (FFD)* heuristic [13] for partitioning the replicas among the available cores. However, we modified the classical FFD heuristic such that a different core is chosen for each replica of a task.

As the first attempt, the algorithm checks if it is possible to obtain a feasible partitioning where every task has its preferred (i.e., minimum-energy) replica-frequency configuration. If so, this is clearly the optimal solution for the entire problem. Otherwise, we check the other extreme, where every replica is forced to run at f_{max} to minimize the number and total CPU time of all the replicas. If there is no feasible solution for this case, the algorithm exits with an error report. Otherwise, the algorithm moves on to the next phase, which we call the *relaxation phase*.

In the relaxation phase, the algorithm starts with a feasible configuration where every replica runs at f_{max} and all tasks are marked as *eligible* for relaxation. Then in each step, based on the specific task selection heuristic (that will be discussed shortly), one eligible task is chosen and its frequency is reduced by one level according to its EFR table. If the resulting configuration is also feasible, the new configuration is committed to and the algorithm proceeds to the next step. Otherwise, the algorithm backtracks to the previous configuration and the chosen task is marked as *ineligible* for future relaxations. If a task reaches its minimum energy configuration

level in the EFR table, it is also marked as ineligible for additional slowdown. The algorithm stops when there is no more eligible task further relaxation. Algorithm 1 shows the pseudo-code of the algorithm.

Algorithm 1 Algorithm Energy-Efficient Replication (EER)

```
Construct the EFR tables for all tasks
for i = 1 to N do
/* assume j_i is the most energy-efficent frequency-level for
τ_i in the EFR table */
    CurConfig[i] ← j_i;
end for
Partition the workload in CurConfig with modified FFD
if ( feasible(CurConfig) ) then
    return CurConfig and the partitioning ρ
    exit
end if
for i = 1 to N do
    CurConfig[i] ← 1;
    eligible[i] ← true;
end for
Partition the workload in CurConfig with modified FFD
if ( !feasible(CurConfig) ) then
    return error; /* No feasible solution exists */
    exit
end if
while (∃j eligible[j] ) do
    Choose eligible task according to LEF, LPF or LUF
    /* τ_i is chosen for relaxation */
    CurConfig[i] ++;
Partition the workload in CurConfig with modified FFD
    if( !feasible(CurConfig) ) then
        CurConfig[i] −−;
        eligible[i] ← false;
    end if
end while
return CurConfig and the partitioning ρ;
```

In Algorithm 1, we first construct the EFR tables for all tasks (Section IV-A). Due to trimming of the *inefficient* frequency levels, the minimum energy configuration for each task can be found at the last row of the corresponding EFR table. We use *CurConfig[i]* to denote the current configuration of τ_i. For example, if the current configuration for τ_i is *RfConfig(i,j)*, we set *CurConfig[i]* to j. For all tasks, we first set the current configuration to be the most energy-efficient configuration from the corresponding EFR table. We then attempt to partition the tasks using the modified FFD. If the partitioning succeeds, we have obtained an optimal solution. Otherwise, we set the current configuration to the first entry of the EFR table for every task, which corresponds to using f_{max}. If there exists no feasible partitioning for that setting, the algorithm quits. Otherwise, the algorithm relaxes one eligible task at a time and reduces its frequency by one level, until

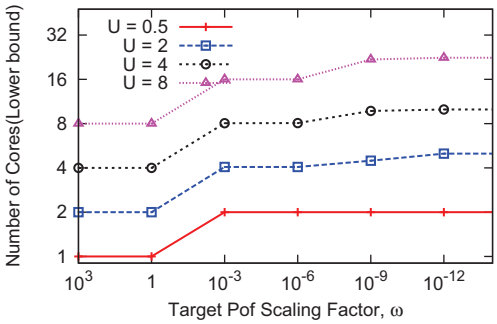


Fig. 5: Lower bound of number of cores

energy savings cannot be further improved while preserving feasibility.

Several heuristics can be applied to choose the task for relaxation in every iteration. For this, we considered three heuristics as described below.

Largest-Energy-First (LEF): In this heuristic we choose the task that will provide the largest energy savings when relaxed to the next level in the EFR table. For task τ_i, let the current configuration be the row j in the EFR table. Then, the task with the largest

$$\Delta E \;=\; E(i,j) - E(i,j+1)$$

value is selected according to this heuristic.

Largest-Power-First (LPF): We choose the task that provides the largest energy savings per unit time for the additional CPU time required for the next level in the corresponding EFR table. Therefore, task τ_i is selected to maximize:

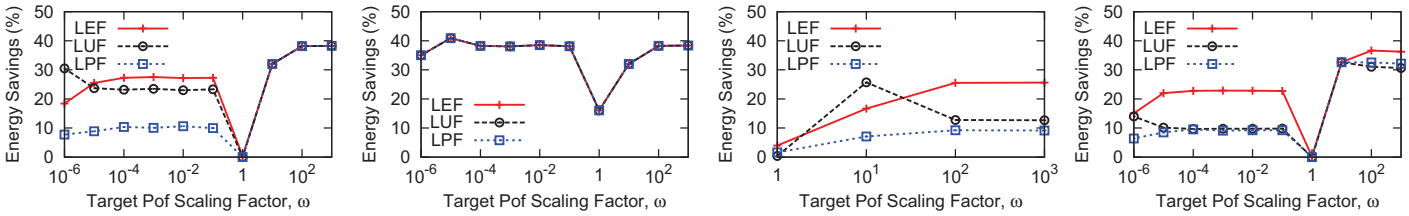
$$\frac{\Delta E}{\Delta S} \;=\; \frac{E(i,j) - E(i,j+1)}{S(i,j+1) - S(i,j)}$$

Largest Utilization First (LUF): This is a simple heuristic where the task with largest utilization value is chosen first.

We now analyse the complexity of the proposed solution. In the first phase, we construct the EFR tables for each task. As discussed in section IV-A, each table can be constructed in $O(K)$ time. Therefore, the running time of phase 1 is $O(NK)$ in the worst case. In the relaxation phase, we can have at most NK relaxation steps and for obtaining the partitioning the cost is $O(NM)$ in the worst case. Therefore, the overall running time of the algorithm is $O(N^2MK)$. Note that, for most practical systems K and and M are small constants. Also observe that, the algorithm is executed only once as a pre-processing phase.

VI. Performance Evaluation

In this section, we present our simulation results to evaluate the performance of our proposed scheme. We considered three different heuristics - LEF, LPF and LUF - for choosing the tasks for relaxation. As a *baseline* scheme, we considered the case where all replicas run at f_{max} and with the minimum number of replicas required to achieve the target reliability. We



(a) $U_{tot} = 1.2$, 4 cores (b) $U_{tot} = 1.2$, 8 cores (c) $U_{tot} = 3.2$, 4 cores (d) $U_{tot} = 3.2$, 8 cores

Fig. 6: Impact of target *Pof*

report the energy savings of our proposed scheme compared to the baseline scheme.

We constructed a discrete event simulator to evaluate the performance of our schemes. For each data point, we considered 1000 data sets with 20 tasks. The task utilizations are generated randomly using the UUnifast scheme [2]. Task periods are generated between 10 ms and 100 ms. The default number of speed steps in the system is 10. The static power and the frequency-dependent power are set to 5% of the maximum dynamic power consumption.

Lower bound on the number of cores. In general, the number of cores required to execute all the replicas may be significant for a given reliability target. A lower bound on this number may be obtained as follows. In general, one or more copies of each task is needed to achieve the reliability objectives. Let us denote the total utilization of the entire workload (all the replicas) by *effective utilization*. The number of cores required cannot be smaller than the *minimum effective utilization*, which is the effective utilization when all tasks/cores run at f_{max}. Also, all replicas of a given task execute on different cores. Let us denote by $R(i)$ the number of replicas needed by τ_i when executed at f_{max}, which can be readily obtained from the EFR table. Hence, the minimum number of cores required is the maximum of greatest number of replicas needed by any task and the minimum effective utilization:

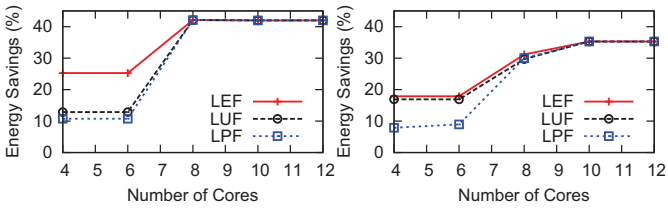
$$Lower\text{-}bound = \forall_i \max\{R(i), \lceil \Sigma_{j=1}^{N} R(j) \times \frac{c_j}{P_j} \rceil\}$$

Figure 5 shows the lower bound for the number of cores for different utilization values, as we vary the uniform *Pof* scaling factor ω from 10^3 to 10^{-15} under different total utilization (U_{tot}) values. We observe that, as we increase the reliability target (smaller ω), the number of cores required increases very fast. When ω goes below 1, each job requires at least two replicas to achieve the reliability target. So, at that point the minimum number of cores is doubled. As we require additional reliability or increase the load (utilization), the number of required cores further increases.

Impact of Target Reliability. We now consider the impact of target uniform *Pof* scaling factor ω on the system energy consumption. Figure 6 shows the energy savings for task sets with total utilization 1.2 and 3.2 running on 4- and 8-core systems. The energy gain reaches the minimum value for target $\omega = 1$. The reason is, when the target $\omega = 1$, the target reliability becomes equal to the reliability obtained trivially by the baseline scheme with exactly 1 replica for each task. In this case, our schemes can only achieve energy savings by running more than one replica at a considerably lower speed. However, this may require large number of replicas running at a very low speed, which adversely affects the feasibility. Therefore, only for very low load on 8 cores (Fig. 6b) LEF, LUF and LPF can achieve energy savings when $\omega = 1$. Otherwise, the baseline scheme provides the optimal solution and there is a significant drop in energy savings for $\omega = 1$. We observe that for all settings, as we increase ω beyond 1, the energy savings increase as we can afford running the replicas at lower frequencies. When the frequency reaches the energy-efficient frequency, the savings reach a stable level. On the other hand, when $\omega < 1$, the baseline scheme uses two replicas running at f_{max} and LEF, LPF and LUF can execute replicas at lower frequencies. Even though LEF, LPF and LUF may use more replicas compared to the baseline scheme, they still save energy thanks to execution at lower frequencies. As ω decreases even further, the required processing frequency increases and there is a slight drop in energy savings. Notice that, typically LEF provides higher energy savings than LPF and LUF.

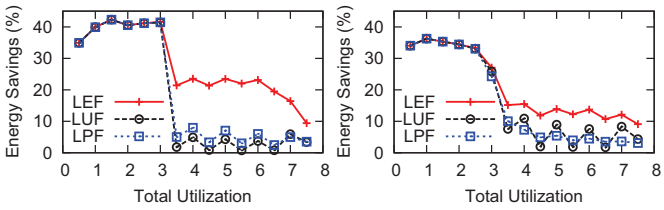
In Figure 6b, the system utilization is very low compared to the available cores. As a result, the minimum energy configurations are feasible. Therefore, the energy savings for all schemes converge. In Figure 6c, on the other hand, the system utilization is very high. Therefore, it is not possible to find a feasible solution for target $\omega < 1$. For Figure 6a and 6d, the utilization is moderate considering the number of cores. Therefore, there is no feasible solution for the most energy-efficient settings. However, some tasks can be feasibly relaxed to run at a lower frequency. Due to the difference in the order of task selection for relaxation, the heuristics differ in their performances.

Impact of the number of cores. Next, we evaluate the impact of the number of cores on the system performance. Figure 7a and 7b show the energy consumption for a task set with total utilization 1.5 for target *Pof* scaling factor set to ω 10^{-3} and 10^{-6}, respectively. Observe that increasing the number of cores allows greater slowdown of replicas and hence typically provides greater energy savings. We notice that initially small increase in the number of cores does not translate to energy gains, as it does not allow sufficient slow-down of jobs. Then,



(a) Target $\omega = 10^{-3}$ (b) Target $\omega = 10^{-6}$

Fig. 7: Impact of the number of cores ($U_{tot} = 1.5$, Effective utilization $= 3.0$)



(a) Target $\omega = 10^{-3}$ (b) Target $\omega = 10^{-6}$

Fig. 8: Impact of system load, 16 cores

as we can relax more tasks with added cores, the energy savings increase sharply. The energy savings for all schemes converge at a stable point, when choosing the most energy-efficient configurations become feasible for all tasks. Also notice that, the energy savings are greater for larger target ω as it allows more slack for relaxation.

Impact of the System Load. Figure 8 represents the impact of total utilization U_{tot} on the energy savings. For this experiment, we set the number of cores in the system to 16 and the target ω is set to 10^{-3} and 10^{-6} respectively. We vary the utilization from 0.5 to 8 and note the energy savings. We observe that, at very low utilization the energy savings is the highest, as we can find feasible partitioning for the minimum energy configurations. Also, for very small utilizations the energy savings first increase with added workload. Due to very low utilization, the replicas can still run at the same low frequency, while the baseline scheme keeps running at f_{max}. Since the dynamic energy consumption increases in squared fashion with respect to frequency, the energy consumption for the baseline scheme increases at a higher rate than our schemes. Therefore, the energy consumption increases slightly at first. Then, as the frequency of the replicas increase, the energy savings decreases slowly. When the minimum energy configurations become infeasible, the schemes start to differ and there is a sharp drop in energy savings, as the number of jobs that can be relaxed drops. As we continue increasing the system load, due to lower available slack, the energy savings become modest. Again, LEF outperforms LPF and LUF. Also notice that the energy savings is lower for smaller ω target, as it requires running more replicas and/or at a higher frequency.

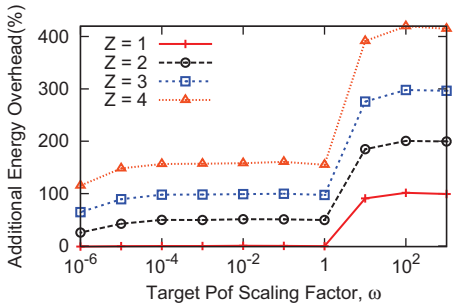


Fig. 9: Additional energy overhead of scheduling at least $Z+1$ replicas per task

VII. Extension to Tolerate a Fixed Number of Core Permanent Faults

The presented framework in this paper essentially aims at managing reliability in terms of tolerance to transient faults induced by dynamic voltage and frequency scaling. While less common, individual cores may also encounter *permanent faults* during task execution due to circuit wear-out or manufacturing defects [14]. Permanent faults may result in the unavailability of some processing cores and can only be dealt with redundant execution of tasks on different cores. The reader may have observed that the allocation of the replicas to different cores provides simultaneously an appropriate basis to tolerate the permanent faults of processing cores.

While the design and implementation of a comprehensive system to detect and recover from the permanent faults at runtime is a fairly complicated task ([14], [17]) that goes beyond the scope of this paper, below we sketch how our algorithm EER can be modified to allocate task replicas in such a way to tolerate up to Z permanent core faults. In order to tolerate Z permanent faults affecting any Z cores, we must schedule at least $(Z + 1)$ replicas for every task. Hence, during the construction of the EFR tables, for every frequency setting, we can choose the minimum of $(Z + 1)$ and the least number of replicas required to achieve the reliability target at that setting. Obviously, if $Z + 1$ is the larger quantity, then the energy consumption and CPU time entries for that configuration in the table will need to be updated accordingly. This modification makes sure that at least $Z + 1$ replicas of every task are scheduled on $Z + 1$ distinct cores, while still maintaining the reliability targets in terms of tolerance to transient faults.

We conducted simulations to assess the impact of additional requirement to schedule at least $Z + 1$ replicas for each task. Due to space limitations, we present a single plot that summarizes the main trends in Figure 9. Here, we vary the uniform *Pof* scaling factor ω from 10^3 to 10^{-6}. We report the additional energy overhead with respect to the case where we have no guarantees for permanent fault tolerance, i.e. $Z = 0$, as we increase Z from 1 to 4. We only present the results for the LEF heuristic. Notice that, when $\omega > 1$, the additional energy overhead is Z times the base energy consumption. This is because, in that case, the transient fault reliability target can be achieved with only one replica per task; but still we

need additional Z replicas for potential permanent faults. We observe that, as we decrease ω, the overhead decreases slightly. As we decrease ω, more and more replicas are required to achieve the given transient fault reliability target, and, we need to assign less additional replicas to handle permanent faults. For the same reason, we also notice that, the increase in overhead for increased permanent fault tolerance target is smaller when target $\omega < 1$ compared to the region where $\omega > 1$. The energy overhead is 0 for $Z = 1$ when the target $\omega < 1$, because in that region, the default behavior of the EER algorithm is to assign more than 1 replica to any task.

VIII. Conclusions

In this paper, we considered the reliability-oriented energy-management problem for preemptive periodic real-time applications running on a multi-core system. We showed how replication can be used to achieve the given task-level reliability targets that are expressed in terms of tolerance to transient faults. While replication allows the use of lower frequencies on different cores for a given reliability target to mitigate the negative impact of DVS on transient faults, it has also the potential of increasing energy consumption. We presented techniques to determine the degree of replication and the frequency assignment for each task while minimizing overall energy. Although the problem is intractable in the general case, our efficient heuristics are shown to satisfy the given reliability targets with considerable energy savings through simulations. We also a presented an extension that allows tolerating up to Z permanent faults on processing cores with minimum energy consumption.

Acknowledgement

This work was supported by US National Science Foundation awards CNS-1016855, CNS-1016974, and CAREER Award CNS-0953005.

References

[1] L. Benini, A. Bogliolo, and, Giovanni De Micheli. A Survey of Design Techniques for System-level Dynamic Power Management. *IEEE Trans. on VLSI Systems*, vol. 8, no. 3, pp. 299 - 316, 2000.

[2] E. Bini and G. C. Buttazo. Measuring the Performance of Schedulability Tests. *Journal of Real-Time Systems*, vol. 30, no. 1-2, pp. 129 - 154, 2005.

[3] X. Castillo, S. R. McConnel, and D. P. Siewiorek. Derivation and Calibration of a Transient Error Reliability Model. *IEEE Trans. on Computers*, vol. 31, pp. 658 - 671, 1982.

[4] V. Devadas and H. Aydin. Real-Time Dynamic Power Management through Device Forbidden Regions. In *Proc. of the 14th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'08)*, 2008.

[5] V. Devadas and H. Aydin. On the Interplay of Voltage/Frequency Scaling and Device Power Management for Frame-based Real-Time Embedded Applications. *IEEE Transactions on Computers*, vol. 61, no. 1, pp. 31 - 44, 2012.

[6] A. Ejlali, B. M. Al-Hashimi, and P. Eles. Low-Energy Standby-Sparing for Hard Real-Time Systems. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 31, no. 3, pp. 329 - 342, 2012.

[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*, vol. 6, pp. 10 - 20, November - December, 2004.

[8] X. Fan, C. Ellis, and A. Lebeck. The Synergy Between Power-Aware Memory Systems and Processor Voltage Scaling. *Lecture Notes in Computer Science Power - Aware Computer Systems*. Springer Berlin/Heidelberg, vol. 3164, pp 151 - 166, 2005.

[9] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* Freeman, San Francisco, 1979.

[10] M. A. Haque, H. Aydin and D. Zhu. Energy-Aware Standby-Sparing Technique for Periodic Real-Time Applications. In *Proc. of IEEE International Conference on Computer Design (ICCD'11)*, 2011.

[11] 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 System*, vol. 4, pp. 214 - 237, 1986.

[12] R. Jejurikar , C. Pereira and R. Gupta. Leakage Aware Dynamic Voltage Scaling for Real-Time Embedded Systems. In *Proc. of IEEE/ACM Design Automation Conference (DAC'04)*, 2004.

[13] D. S. Johnson, A. Demers, J. D. Ullman, M. R. Garey, and R. L. Graham. Worst-Case Performance Bounds for Simple One-Dimensional Packing Algorithms. *SIAM Journal on Computing*, vol. 3, no. 4, pp. 299 - 325, 1974.

[14] I. Koren and C. M. Krishna. *Fault-Tolerant Systems*. Morgan Kaufman, 2007.

[15] C. L. Liu and James W. Layland. Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment. *Journal of ACM*, vol. 20, no. 1, pp. 46 - 61, 1973.

[16] R. Melhem, D. Mossé, and E. Elnozahy. The Interplay of Power Management and Fault Recovery in Real-Time Systems. *IEEE Trans. on Computers*, vol. 53, pp. 217 - 231, 2004.

[17] D. Pradhan. *Fault Tolerant Computer System Design*, Prentice Hall, 1996.

[18] R. Sridharan and R. Mahapatra. Reliability Aware Power Management for Dual-Processor Real-Time Embedded Systems. In *Proc. of the 47th IEEE/ACM Design Automation Conference (DAC'10)*, 2010.

[19] O. S. Unsal, I. Koren, and C. M. Krishna. Towards Energy-Aware Software-Based Fault Tolerance in Real-time Systems. In *Proc. of the IEEE International Symposium on Low Power Electronics and Design (ISLPED 2002)*, 2002.

[20] M. Weiser, B. Welch, A. Demers, and S. Shenker. Scheduling for Reduced CPU Energy. *Mobile Computing of the International Series in Engineering and Computer Science*. Springer US, vol. 353, pp. 449 - 471, 1996.

[21] Y. Zhang and K. Chakrabarty. Energy-Aware Adaptive Check Pointing in Embedded Real-Time Systems. In *Proc. of IEEE/ACM Design, Automation and Test in Europe (DATE)*, 2003.

[22] Y. Zhang and K. Chakrabarty. Dynamic Adaptation for Fault Tolerance and Power Management in Embedded Real-Time Systems. *ACM Trans. on Embedded Computer System*, vol. 3, pp. 336 - 360, May 2004.

[23] B. Zhao, H. Aydin, and D. Zhu. Enhanced Reliability-Aware Power Management Through Shared Recovery Technique, In *Proc. of IEEE International Conference on Computer Aided Design (ICCAD)*, 2009.

[24] B. Zhao and H. Aydin. Minimizing Expected Energy Consumption through Optimal Integration of DVS and DPM. In *Proc. of IEEE International Conference on Computer Aided Design (ICCAD)*, 2009.

[25] B. Zhao, H. Aydin, and D. Zhu. Energy Management under General Task-Level Reliability Constraints. In *Proc. of 18th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2012.

[26] D. Zhu, R. Melhem, and D. Mossé. The Effects of Energy Management on Reliability in Real-Time Embedded Systems. In *Proc. of IEEE International Conference on Computer Aided Design (ICCAD)*, 2004.

[27] D. Zhu and H. Aydin. Reliability-Aware Energy Management for Periodic Real-Time Tasks. *IEEE Trans. on Computers*, vol. 58, no. 10, pp. 1382 1397, 2009.

[28] J. Zhuo and C. Chakrabarti. System-Level Energy-Efficient Dynamic Task Scheduling. In *Proc. of the 42nd IEEE/ACM Design Automation Conference (DAC)*, 2005.

[29] Intel Corporation. Intel Pentium M Processor Datasheet. http://download.intel.com/support/processors/mobile/pm/sb/25261203.pdf, 2004.

[30] Intel Corporation. Single-Chip Cloud Computer: Project. http://www.intel.com/content/www/us/en/research/intel-labs-single-chip-cloud-computer.html

[31] Tilera. Tile64 Processor.http://www.tilera.com/products/processors/TILE64