Energy and Pre-emption Savings through Real-Time Race-To-Halt Algorithms

Muhammad Ali Awan

Stefan M. Petters

CISTER-TR-130507
Version:
Date: 05-30-2013

Energy and Pre-emption Savings through Real-Time Race-To-Halt Algorithms

Muhammad Ali Awan, Stefan M. Petters

CISTER Research Unit

Polytechnic Institute of Porto (ISEP-IPP)

Rua Dr. António Bernardino de Almeida, 431

4200-072 Porto

Portugal

Tel.: +351.22.8340509, Fax: +351.22.8340509

E-mail:

http://www.cister.isep.ipp.pt

Abstract

Energy consumption has become one of the major issues for modern embedded systems. Initially, power saving approaches mainly focused on dynamic power dissipation, while neglecting the static (leakage) power consumption. However, technology improvements resulted in a case where static power dissipation increasingly dominates. Addressing this issue, hardware vendors have equipped modern processors with several sleep states. We propose a set of leakage-aware energy management approaches that reduce the energy consumption of embedded real-time systems while respecting the real-time constraints. Our algorithms are based on the race-to-halt strategy that tends to run the system at top speed with an aim to create long idle intervals, which are used to initiate a sleep state. The effectiveness of our algorithms is illustrated with an exhaustive simulations that show an improvement of up to 8% reduction in energy consumption over an existing work at high utilization. The complexity of our algorithms is smaller when compared to state-of-the-art algorithms. We also eliminate assumptions made in the related work that restrict the practical implications of their algorithms. Moreover, a novel study about the sleep states relation with the number of pre-emptions is also presented with an extensive set of simulation results, where our algorithms reduce the experienced number of pre-emptions in all cases. Our results show that sleep states in general can save up to 30% of the overall number of pre-emptions when compared to the sleep-agnostic earliest-deadline-first algorithm.

Energy and Pre-emption Savings through Real-Time Race-To-Halt Algorithms

Muhammad Ali Awan Stefan M. Petters
CISTER Research Unit, ISEP-IPP Porto, Portugal
maan,smp@isep.ipp.pt

Abstract

Energy consumption has become one of the major issues for modern embedded systems. Initially, power saving approaches mainly focused on dynamic power dissipation, while neglecting the static (leakage) power consumption. However, technology improvements resulted in a case where static power dissipation increasingly dominates. Addressing this issue, hardware vendors have equipped modern processors with several sleep states.

We propose a set of leakage-aware energy management approaches that reduce the energy consumption of embedded real-time systems while respecting the real-time constraints. Our algorithms are based on the race-to-halt strategy that tends to run the system at top speed with an aim to create long idle intervals, which are used to initiate a sleep state. The effectiveness of our algorithms is illustrated with an exhaustive simulations that show an improvement of up to 8% reduction in energy consumption over an existing work at high utilization. The complexity of our algorithms is smaller when compared to state-of-the-art algorithms. We also eliminate assumptions made in the related work that restrict the practical implications of their algorithms. Moreover, a novel study about the sleep states relation with the number of pre-emptions is also presented with an extensive set of simulation results, where our algorithms reduce the experienced number of pre-emptions in all cases. Our results show that sleep states in general can save up to 30% of the overall number of pre-emptions when compared to the sleep-agnostic earliest-deadline-first algorithm.

I. Introduction

Embedded systems have become an integral part of our daily life. These devices are designed to perform a set of functions and interact with their environment. Some examples are air traffic management systems, car navigation, aircraft flight control system or common cell phones. Among the vast variety of these systems, devices with timing constraints are categorized as Real-Time (RT) embedded systems. These devices have additional timing constraints, which are required to be met on top of functional aspects for the overall system to be considered correct.

Apart from RT constraints, many embedded devices are nomadic and have limited energy supply. These devices rely on the batteries or intermittent power supply such as solar cells. Even if the embedded systems are connected to the regular power source, they have thermal issues such as data centres. Hardware vendors have provided system designer some special features such as dynamic voltage and frequency scaling (DVFS) and sleep states to optimize the energy efficiency of the system. DVFS scales the frequency while sleep states are based on race-to-halt mechanism. Race-to-halt mechanism executes the workload at high frequency to finish it earlier with the aim of shutting down certain parts of the hardware.

CMOS technology miniaturization has increased the leakage power tremendously. The increase is exponential as the process moves to finer technologies [1]. Leakage power consumption and its variability has been identified as a major concern in the International Technology RoadMap For Semiconductors 2010 Update under special topics [2]. Leakage power consumption comes from the sub-threshold current that flows through the transistors. A need to reduce the leakage current motivated the hardware vendors to put an extra effort to equip the modern embedded processors with several sleep states allowing a trade-off between transition overhead and power consumption in a sleep state. On the other hand, DVFS is designed to reduce dynamic power consumption by reducing the operating frequency and thus allowing a lower voltage to be used leading to higher than linear energy gains. Comparing the two approaches, DVFS reduces dynamic energy consumption at the cost of leakage energy consumption, while race-to-halt (RTH) does the opposite. The selection of either power saving mechanism (DVFS or RTH) is not clear cut as it depends on the available workload and the underlying hardware [3]–[5]. For instance, DVFS is suitable for the memory intensive workloads. On contrary, RTH favors CPU intensive workloads. Nevertheless, in the scope of this paper, we assume that either processor or the underlying hardware platform does not allow the frequency/voltage scaling to reduce the energy consumption. Such COTS (common off the shelf) processors are commonly used in embedded systems. For instance, ARM Cortex-A Series high performance processors [6] are readily used in tablets, smart-phones, auto infotainment, netbooks, digital TV and home network. Similarly, ARM Cortex-R Series also only use sleep states to save energy. An RTH mechanism may be used for such systems. It has to be noted, that this work does not claim general superiority over DVFS.

The increase in computing power also leads to a progressive integration of functionality into a single device. For example, a current mobile phone combines applications of soft real-time character (e.g. base station communication) with such of best-effort character (e.g. SMS). Additionally the different system components and software modules are

This work was financed by FEDER funds (EU) through the Operational Programme 'Thematic Factors of Competitiveness' - COMPETE, by National Funds (PT) through FCT - Portuguese Foundation for Science and Technology and the ARTEMIS-JU, under the projects RePoMuC (PTDC/EIA-EIA/112599/2009) and RECOMP (ARTEMIS/0202/2009).

potentially provided by different third party suppliers. Consequently such mixed criticality systems require temporal and functional isolation not only to protect critical applications from less critical ones, but also as a means to identify the offending application in the case of a misbehaving system.

State-of-the-art research in the energy management domain which exploits sleep states to save energy commonly assume additional hardware to run the energy management algorithm. This research effect considers the non-negligible time/energy overhead of sleep transitions and avoids additional hardware. The results show that our proposed approach saves a similar amount of energy and in some cases it saves extra energy upto 8% over the existing energy saving approaches. Moreover, we show the complexity of our algorithms is low when compared to the existing approaches.

This article summarizes and extends the work presented in [7]. The main contributions of the combined work are the following. 1) An energy efficient slack management approach to accumulate the execution slack. This accumulated slack is used online by the energy management algorithm. 2) We show that maximum feasible sleep interval determined offline by our method is always greater than or equal to minimum idle period identified by the state-of-the-art approaches. 3) An enhanced race-to-halt (ERTH) algorithm to minimize the leakage energy consumption for the mixed-critically uniprocessor systems, while avoiding the impractical assumptions made in the state-of-the-art. 4) The pessimism introduced in the ERTH is decreased with an improved race-to-halt algorithm (IRTH) at the cost of extra complexity to predict the future release information. 5) We also propose a complexity-wise light-weight race-to-halt algorithm (LWRTH). 6) Some improvements are also made in the leakage-control earliest-deadline-first algorithm (LC-EDF), to reduce the online overhead and implementation complexity. 7) The relation of sleep states with the pre-emption count is studied. Our results show that sleep states have positive impact on the number of pre-emptions in the average case. 8) Finally, a comparative study is done for different algorithms and their special features are identified that vary the sleep states and pre-emption count relation. The contributions $2, 4, 5, 7$ & 8 are over and beyond the work presented in [7].

The paper is organized as follows. Section II summarizes the limitations of the state-of-the-art followed by a system model in Section III. The break-even-time and schedulability analysis used to determine the sleep interval limit is discussed in Section IV. The following section presents the online slack management algorithm. ERTH, IRTH and LWRTH algorithms are proposed in section VI. Section VII discusses the offline and online overhead of ERTH, IRTH and LWRTH against the state-of-the-art approaches. The effect of sleep states on the number of pre-emption is discussed afterwards. The evaluation and conclusions are given in Sections IX and X respectively.

II. Related Work

To deal with an increase in leakage power consumption, Lee et al. [8] addressed the leakage-aware scheduling for periodic hard real-time systems. They proposed Leakage Control EDF (LC-EDF) and Leakage Control Dual Priority (LC-DP) algorithms for dynamic and static priority schemes respectively. The LC-EDF algorithm maximizes the idle interval by delaying the busy period to increase the duration of the sleep state. They assumed an external specialized hardware such as application specific integrated circuit (ASIC) or field programmable gate array (FPGA) to implement their algorithm. Baptiste [9] did a theoretical study of a non-DVFS system with unit sized real-time aperiodic tasks. He developed a polynomial time algorithm to minimize the energy consumption of static power and the sleep transition overhead of the system.

A combination of leakage-aware and dynamic voltage scheduling appears to be a promising way to reduce overall energy consumption. Irani et al. [10] proposed a 3-competitive offline and constant competitive ratio online algorithms for power saving while considering shutdown in combination with DVFS. Although the combination of shutdown and DVFS has its merits fundamentally, their approach requires further work to relax the assumptions in terms of DVFS power model. Besides requiring external hardware to implement their shutdown algorithm, they assume a continuous spectrum of available frequencies and an inverse linear relation of frequency with execution time. Niu and Quan's [11] scheduling technique also addressed the dynamic and leakage power consumption simultaneously on a DVFS enabled processor for hard-real time systems. They integrated DVFS and shutdown to minimize the overall energy consumption based on the latest arrival time of jobs, which is estimated by expanding the schedule for the hyper-period. However, this algorithm cannot be used online due to extensive analysis overhead. Previously, Jejurikar et al. [12] integrated DVFS with the procrastination algorithm, to minimized the total power consumption. The critical speed η_{crit} is computed that determines the lower bound on the processor frequency to minimize the energy consumption per cycle. Moreover, they showed the procrastination interval determined by their algorithm is always greater than or equal to the procrastination interval estimated by LC-EDF. Nevertheless, they did not relax on the requirement of additional hardware to support their shutdown approach.

Soon after, Jejurikar et al. [13] showed that procrastination under LC-DP originally proposed by Lee et al. [8] may cause some of the tasks to miss their deadlines. They proposed improvements in the original algorithm and also integrated their DVFS approach. However, they adopted the same assumptions of [8], [12]. Later on Chen and Kuo [14] showed that an approach given in [13] still might lead to some tasks missing their deadlines. They proposed a two phase algorithm that estimates the execution speed and procrastination interval offline, and predicts turn off/on instances online but also rely on extra hardware. Further work of Jejurikar and Gupta [15] reclaims the execution slack generated due to the difference between worst-case execution time (WCET) and actual execution time. They used procrastination scheduling and DVFS to minimize the overall energy consumption, and called their approach slack reclamation algorithm (SRA). The dynamically reclaimed slack is either used entirely for slowdown or distributed between slowdown and procrastination using slack distribution policy. This algorithm follows the same assumptions made by previous work [8], [12], [13]. However, it will also be used when evaluating our work.

Chen and Kuo [16] developed a novel algorithm distinct to greedy procrastination algorithms [8], [10]–[15] for procrastination interval determination. They showed that their algorithm can decrease the energy consumption by executing jobs at lower speeds than the critical speed η_{crit}, when the processor is decided not to be turned off in the procrastination interval. Chen and Thiele [17] proposed leakage-aware DVFS scheduling, where tasks execute initially with decelerating frequencies to accumulate the slack to initiate sleep state and towards the end execute with accelerating frequencies to reduce the dynamic power consumption. However, the work of Chen et al. [16], [17] still relies on continuous spectrum of available frequencies and external hardware. Considering previous history of events, predicting the future events using RT calculus [18] and doing the scheduling analysis with RT interfaces [19], Huang et al. [20], [21] estimated the procrastination interval for a device to activate the shutdown.

Santinelli et al. [22] proposed energy-aware packet and task co-scheduling algorithm EAS for the distributed RT embedded system consisting of a set of wireless nodes. The EAS algorithm generates the schedule till the next idle time in the schedule, and determines the frequency of the processor and the sleep interval such that the total energy consumption is minimized while efficiently utilizing the reserved communication bandwidth. The online complexity of this algorithm is high as system has to compute the demand bound function [23] online to determine the frequency and the switch off time. Wang et al. [24] determined the static schedule for the given set of dependent periodic tasks for homogeneous multiprocessors. In the first step they relax the dependencies of the tasks using coarse-grained task parallelization algorithm RDAG. The second phase determines the static schedule using a genetic algorithm (gene evolution) to minimize the energy consumption of the system assigning frequencies to the tasks and enforcing sleep intervals in the schedule. However, the work of Santinelli et al. [22] and Wang et al. [24] is proposed for different system models and hardware platforms compared to what we have assumed in this research.

One of the assumptions commonly made throughout the state-of-the-art is a requirement of the external specialized hardware to implement the procrastination scheduling. The aim of this work is to relax this assumption and propose a low complexity RTH algorithms for the sporadic task-model.

III. System Model

We assume a sporadic task model, with l independent tasks $\mathrm{T} = \{\tau_1, \tau_2, \cdots, \tau_l\}$. A task τ_i is described by $\langle C_i, D_i, T_i \rangle$, where C_i is the worst-case execution time (WCET), D_i the relative deadline and T_i the minimum inter-arrival time. The independent tasks release a sequence of jobs $j_{i,m}$. Each $j_{i,m}$ has a deadline $d_{i,m}$, a release time $r_{i,m}$ and an actual execution time $\hat{c}_{i,m}$. The system utilization $U = \sum_{i=1}^{l} \frac{C_i}{T_i}$.

We use the Rate-Based Earliest Deadline first (RBED) framework [25], which provides temporal isolation by associating with each task τ_i via enforced budgets A_i. At runtime the default value $a_{i,m}$ for a budget when releasing a job $j_{i,m}$ is A_i. However, the value for $a_{i,m}$ may be subject to manipulations; e.g. spare capacity assignment or consumption of budget during execution.

The temporal isolation of the RBED framework allows to mix hard, soft and best-effort type applications. The allocation of budget for soft real-time (SRT) and best-effort (BE) tasks in the original work is less than or equal to WCET ($A_i \leq C_i$). For hard real-time (HRT) tasks the budget is equal to the WCET ($A_i = C_i$), to ensure the timely completion of all jobs. The scheduler pre-empts every job when it has used up its allocated budget $a_{i,m}$. Thus a job exceeding its budget cannot affect the overall schedulability. In our system, we assume HRT and SRT tasks have a budget equal to their WCET and treated as RT tasks onwards, while a BE task may have budget less than or equal to its WCET time.

Our hardware model assumes N sleep states in the system, where each sleep state n is characterized with a power P_n, a transition overhead in terms of time t_n and energy E_n, as well as a break-even time (BET) t_n^e. The time t_n includes transition overhead time to sleep t_n^s, as well as the wake-up time t_n^w to transition out of sleep state i.e. $t_n = t_n^s + t_n^w$. Similarly, energy overhead E_n includes energy consumption during both transitions. Active and idle mode power are represented as P_A and P_L respectively. Each sleep state has a BET t_n^e (discussed in detail in the Section IV). We assume that the hardware platform or processor used does not support DVFS. Moreover, the use of external hardware is avoided to reduce its additional energy.

IV. Static Sleep Interval Limits

Each sleep state has a break-even-time associated to it. In the context of our approach the break-even time t_n^e is defined as follows.

Definition 1: The break-even time t_n^e is the maximum of the complete transition delay of going into and out of a sleep state, and the time interval during which energy consumption of a sleep state n becomes equal to the energy consumption of system running in idle mode.

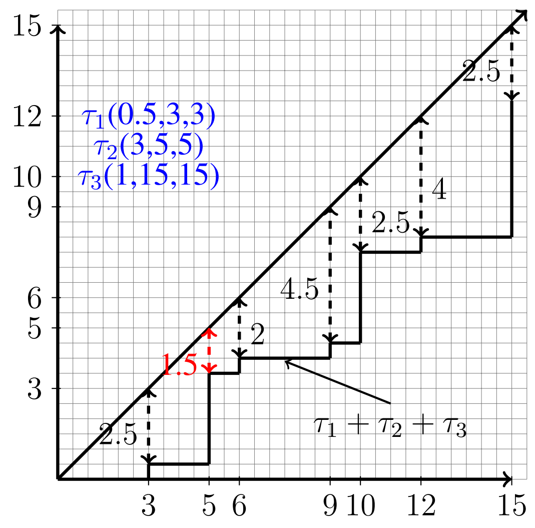


Figure 1. Demand Bound Function

Depending on the hardware characteristics, different ways to compute the break-even time are possible. Some of the examples are given in [26], [27]. The definition of t_n^e implies that the system will save energy if a sleep state is initiated for more than t_n^e. Therefore t_n^e gives a lower bound for the desired sleep interval. In the running system a value greater than t_n^e will be chosen as a minimum sleep interval. Intuitively it is preferable to choose fewer, but longer sleep intervals, when compared to more frequent sleep transitions for small intervals. While the overhead of the sleep transition has a major impact on the lower bound of the sleep interval, the schedulability of the system enforces an upper bound of the sleep interval. We define this upper bound on the sleep interval as static limit t_l for which the system can, under certain conditions (explained later in Section VI) be enforced to stay in a sleep state without causing deadline misses.

Definition 2: The static limit t_l describes the maximum time interval for which the processor may be enforced in a sleep state without causing any application to miss its deadline under worst-case assumptions.

The schedulability analysis of the EDF on uniprocessor [28], [29] is given in Theorem 1. However overall demand bound function for a T can be represented as $DBF_{\mathrm{T}}(L) \overset{\text{def}}{=} max_{L_0} df(L_0, L_0 + L)$ following the definition of Rahni et al. [23].

Theorem 1: A synchronous periodic task-set T is schedulable under EDF if and only if, $\forall L \leq L^*$, $df(0, L) \leq L$, where L is an absolute deadline and L^* is the first idle time in the schedule.

Formally the static limit is defined by exploiting the demand bound function (DBF). Assuming Theorem 1, a static limit for sleep interval t_l is given in Equation 1, where L is an absolute deadline and L^* is the first idle time in the schedule.

$$\forall L \leq L^*, \qquad t_l = \min\left(L - DBF(L)\right) \tag{1}$$

Theorem 2: Initiating a sleep state for the static limit t_l does not violate the EDF schedule if and only if

$$\forall L \leq L^*, \qquad DBF(L) + t_l \leq L$$

Where L is an absolute deadline and L^* is the first idle time in the schedule.

Proof: The sleep interval t_l can be interpreted as the highest priority task. In an EDF scheduled system it is equivalent to a task with deadline equal to the shortest relative deadline of any task in the system. As such the $DBF^*(L)$ is increased over $DBF(L)$ by t_l, following the definition in Equation 1 it follows that $DBF^*(L) \leq L$. ■

The state-of-the-art algorithms also compute such static limit on the sleep interval. The minimum sleep interval ℓ_{min} computed by LC-EDF [8] is given by Theorem 3. Note; that we can only consider the minimum limit to ensure the system schedulability and avoid external hardware. Jejurikar et al. [12] also identified a limit on such sleep interval Z_{min} given in Theorem 4. They proved in their work that $Z_{min} \geq \ell_{min}$.

Theorem 3: Any idle period in LC-EDF is greater than or equal to

$$\ell_{min} = \min_{k=1,2,\ldots,l} \left\{ \ell_k = \left(1 - \sum_{i=1}^{l} \frac{C_i}{T_i}\right) T_k \right\}$$

Theorem 4: The minimum idle period (Z_{min}) identified by the procrastination algorithm [12] is given as

$$Z_{min} = \min_{1 \leq i \leq l} \left\{ Z_i = \left(1 - \sum_{k=1}^{i} \frac{C_k}{T_k}\right) T_i \right\}$$

We claim and prove that the static limit t_l determined through DBF is greater than or equal to Z_{min} and ℓ_{min}. For instance, consider a task-set consists of three tasks with the following parameters: $\tau_1(0.5, 3, 3)$, $\tau_2(3, 5, 5)$ and

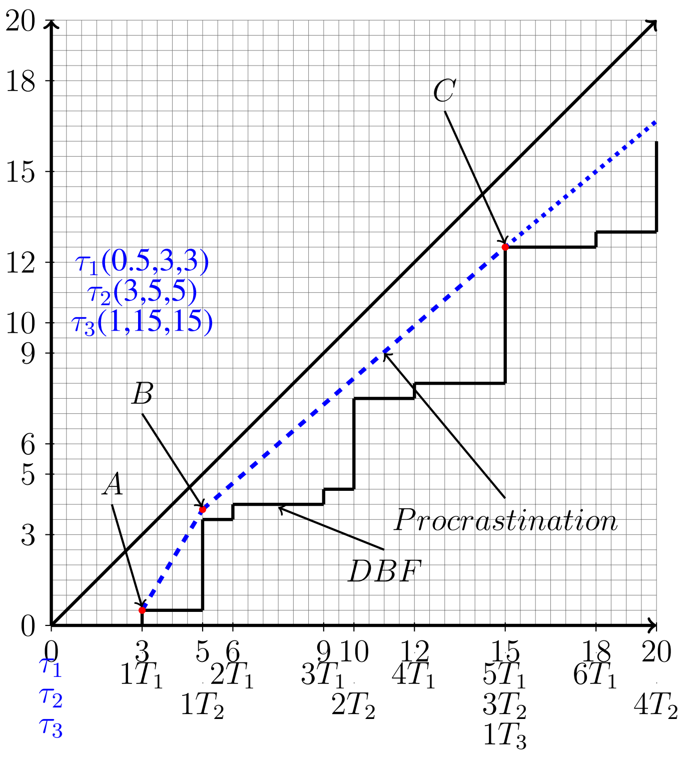


Figure 2. Demand Bound Function vs SRA

$\tau_3(1, 15, 15)$. The DBF for the given task-set is illustrated in Figure 1. For this example, $t_l = 1.5$ time units, $Z_{min} = 1.1667$ and $\ell_{min} = 0.5$, consequently, $t_l \geq Z_{min} \geq \ell_{min}$. Theorem 5 proves the claim that $t_l \geq Z_{min}$.

Theorem 5: The static limit t_l estimated through demand bound function is always greater than or equal to the minimum procrastination interval Z_{min} computed by the procrastination algorithm proposed by Jejurikar et al. [12].

Proof: Assume all the tasks are sorted in non-decreasing order of their periods/deadlines. The minimum procrastination interval Z_{min} determined through Jejurikar et al. [12] algorithm can be rewritten as given in Equation 2.

$$\min_{\forall \tau_i \in \mathrm{T}} \left\{ T_i - \sum_{j=1}^{i} \left(\frac{T_i}{T_j} \right) C_j \right\} \tag{2}$$

Also, the static limit computed in Equation 1 can be rewritten as shown in Equation 3, where L^* is the first idle time in the schedule assuming synchronous task releases.

$$\min_{\forall \tau_i \in \mathrm{T}, \forall_{1 \leq n_i \leq \left\lfloor \frac{L^*}{T_i} \right\rfloor}} \left\{ n_i T_i - \sum_{\forall \tau_j \in \mathrm{T}} \left\lfloor \frac{n_i T_i}{T_j} \right\rfloor C_j \right\} \tag{3}$$

To prove that $t_l \geq Z_{min}$, we have to show that

$$\min_{\forall \tau_i \in \mathrm{T}, \forall_{1 \leq n_i \leq \left\lfloor \frac{L^*}{T_i} \right\rfloor}} \left\{ n_i T_i - \sum_{\forall \tau_j \in \mathrm{T}} \left\lfloor \frac{n_i T_i}{T_j} \right\rfloor C_j \right\} \geq \min_{\forall \tau_i \in \mathrm{T}} \left\{ T_i - \sum_{j=1}^{i} \left(\frac{T_i}{T_j} \right) C_j \right\} \tag{4}$$

In order to prove the inequality given in Equation 4, we have to show that $\forall L \leq L^*$, the demand of the given task-set remains below or equal to the demand computed by the procrastination algorithm proposed by Jejurikar et al. [12] , where L is an absolute deadline. Jejurikar et al. [12] computed the demand of the system using utilizations of the tasks (i.e. on the deadline of each task). Their function is superimposed on the demand bound function. We interpolate their function by connecting the demand of task on its first deadline with its neighbors tasks's demand. Finally, the demand beyond the last period is extended with a line having a slope equal to the system utilization. For instance, consider Figure 2 that shows demand of the given example with both DBF and the procrastination algorithm proposed by Jejurikar et al. [12]. For Jejurikar et al. [12] algorithm, the demand of the tasks computed on their first deadline are represented with A, B and C points. A and B points are connected with a straight line to compare against all the deadlines in the

DBF happening in between these two points. Similarly, B and C points are connected, and the demand beyond C for procrastination algorithm is extended with a line having a slope equal to the utilization of the task-set.

Since the DBF needs to be checked at more instances than A, B and C in the procrastination algorithm, we need to consider constraints. The objective is to find the minimal distance with the unity line and the demand. For all intervals between successive points A, B and C, it is true that the smallest gap between the unity line and the demand within this interval of those can be found in either of the two delimiting points (for example, for interval $[A, B]$, the smallest gap can either occur at A or B). Since the system utilization $U \leq 1$, we also know that beyond the largest period, the largest gap can be found at the largest period point. In order to demonstrate that the gap computed with the DBF is always greater than or equal to that of procrastination algorithm [12] it is sufficient to show that the DBF test dominates at
a) First deadline of every tasks.
b) The demand computed by the DBF is always smaller than the connecting lines of the first deadline of all tasks.

Case a) To get the first deadline of every task, we set $n_i = 1$ in Equation 4 as shown in Equation 5.

$$\min_{\forall \tau_i \in \mathrm{T} n_i=1} \left\{ n_i T_i - \sum_{\forall \tau_j} \left\lfloor \frac{n_i T_i}{T_j} \right\rfloor C_j \right\} \geq \min_{\forall \tau_i \in \mathrm{T}} \left\{ T_i - \sum_{j=1}^{i} \left(\frac{T_i}{T_j} \right) C_j \right\} \quad (5)$$

$$-\sum_{\forall \tau_j} \left\lfloor \frac{T_i}{T_j} \right\rfloor C_j \geq -\sum_{j=1}^{i} \left(\frac{T_i}{T_j} \right) C_j \quad (6)$$

$$\sum_{\forall \tau_j} \left\lfloor \frac{T_i}{T_j} \right\rfloor C_j \leq \sum_{j=1}^{i} \left(\frac{T_i}{T_j} \right) C_j \quad (7)$$

$$\sum_{\forall \tau_j > \tau_i} \left\lfloor \frac{T_i}{T_j} \right\rfloor C_j + \sum_{\forall \tau_j \leq \tau_i} \left\lfloor \frac{T_i}{T_j} \right\rfloor C_j \leq \sum_{j=1}^{i} \left(\frac{T_i}{T_j} \right) C_j \quad (8)$$

$$0 + \cancel{\sum_{\forall \tau_j \leq \tau_i}} \left\lfloor \frac{T_i}{T_j} \right\rfloor \cancel{C_j} \leq \cancel{\sum_{j=1}^{i}} \left(\frac{T_i}{T_j} \right) \cancel{C_j} \quad (9)$$

$$\left\lfloor \frac{T_i}{T_j} \right\rfloor \leq \left(\frac{T_i}{T_j} \right) \quad (10)$$

Equation 10 shows for first case ($n_i = 1$) the inequality given in Equation 4 holds.

Case b: Suppose τ_{i-1} is the preceding task of τ_i. This case checks Equation 4 for all the deadlines that exist between T_{i-1} and T_i, i.e. ($\forall_{\tau_j \in \mathrm{T}} : T_{i-1} \leq n_i T_j \leq T_i$). Equation 11 is a line between two points (x_1, y_2) and (x_2, y_2). In DBF, x and y axis represent the time and demand respectively. Lets assume, the coordinates at the deadlines of τ_{i-1} and τ_i are $(x_1, y_1) = \left(T_{i-1}, \sum_{j=1}^{i-1} \left(\frac{C_j}{T_j} \right) T_{i-1} \right)$ and $(x_2, y_2) = \left(T_i, \sum_{j=1}^{i} \left(\frac{C_j}{T_j} \right) T_i \right)$ respectively. To find the equation between these two points, we put the corresponding coordinates of (x_1, y_2) and (x_2, y_2) into Equation 11 as shown in Equation 12.

$$y = \frac{y_2 - y_1}{x_2 - x_1}(x - x_1) + y_1 \quad (11)$$

$$y = \frac{\left(\sum_{j=1}^{i} \frac{C_j}{T_j} T_i - \sum_{j=1}^{i-1} \frac{C_j}{T_j} T_{i-1} \right)}{T_i - T_{i-1}} (x - T_{i-1}) + \sum_{j=1}^{i-1} \frac{C_j}{T_j} T_{i-1} \quad (12)$$

$$y = \frac{\left(\left(\sum_{j=1}^{i-1} \frac{C_j}{T_j} + \frac{C_i}{T_i} \right) T_i - \sum_{j=1}^{i-1} \frac{C_j}{T_j} T_{i-1} \right)}{T_i - T_{i-1}} (x - T_{i-1}) + \sum_{j=1}^{i-1} \frac{C_j}{T_j} T_{i-1} \quad (13)$$

$$y = \left(\sum_{j=1}^{i-1} \frac{C_j}{T_j} + \frac{C_i}{T_i - T_{i-1}} \right) (x - T_{i-1}) + \sum_{j=1}^{i-1} \frac{C_j}{T_j} T_{i-1} \quad (14)$$

$$y = x \left(\sum_{j=1}^{i-1} \frac{C_j}{T_j} + \frac{C_i}{T_i - T_{i-1}} \right) - \frac{C_i T_{i-1}}{T_i - T_{i-1}} \quad (15)$$

Now consider any deadline that lies in between the deadlines of τ_{i-1} and τ_i(i.e. between (x_1, y_2) and (x_2, y_2)). We show the demand (y-axis) of such deadlines will be below or on the line given in Equation 15. Lets says any deadline between (x_1, y_2) and (x_2, y_2) is $(x_m, y_m) = (n_k T_k, \sum_{\forall \tau_j : \tau_j < \tau_k n_k} \left\lfloor \frac{n_k T_k}{T_j} \right\rfloor C_j)$. Put the x coordinate of selected point (x_m, y_m) into Equation 15 and compare the resulting value of y coordinate with its y_m. If it is greater than or equal to y_m then we are below or on the line. The resulting expression is shown in Equation 16.

$$n_k T_k \left(\sum_{j=1}^{i-1} \frac{C_j}{T_j} + \frac{C_i}{T_i - T_{i-1}} \right) - \frac{C_i T_{i-1}}{T_i - T_{i-1}} \geq \sum_{\forall \tau_j : \tau_j < \tau_k n_k} \left\lfloor \frac{n_k T_k}{T_j} \right\rfloor C_j \tag{16}$$

As we know, the point (x_m, y_m) is in between T_{i-1} and T_i, therefore, the factor $\sum_{\forall \tau_j : \tau_j < \tau_k n_k} \left\lfloor \frac{n_k T_k}{T_j} \right\rfloor C_j$ can be rewritten as $\sum_{j=1}^{i-1} \left\lfloor \frac{n_k T_k}{T_j} \right\rfloor C_j$ as given below.

$$n_k T_k \left(\sum_{j=1}^{i-1} \frac{C_j}{T_j} + \frac{C_i}{T_i - T_{i-1}} \right) - \frac{C_i T_{i-1}}{T_i - T_{i-1}} \geq \sum_{j=1}^{i-1} \left\lfloor \frac{n_k T_k}{T_j} \right\rfloor C_j \tag{17}$$

$$\sum_{j=1}^{i-1} \frac{n_k T_k}{T_j} C_j + \frac{n_k T_k C_i}{T_i - T_{i-1}} - \frac{C_i T_{i-1}}{T_i - T_{i-1}} \geq \sum_{j=1}^{i-1} \left\lfloor \frac{n_k T_k}{T_j} \right\rfloor C_j \tag{18}$$

$$\sum_{j=1}^{i-1} \frac{n_k T_k}{T_j} C_j + \frac{C_i}{T_i - T_{i-1}} (n_k T_k - T_{i-1}) \geq \sum_{j=1}^{i-1} \left\lfloor \frac{n_k T_k}{T_j} \right\rfloor C_j \tag{19}$$

Obviously, $n_k T_k - T_{i-1}$ is greater than 0 as $n_k T_k > T_{i-1}$. Hence, all the deadlines such that $\forall_{\tau_j \in \mathrm{T}} : T_{i-1} \leq n_i T_j \leq T_i$ lie below the line represented by Equation 15.

As the difference computed between supply and the demand for all deadlines (case a and b) are greater than or equal to their corresponding difference computed through Jejurikar et al. [12] algorithm, therefore, the given theorem holds. ■

Similarly, we can also prove, a procrastination interval of each task computed with DBF will be greater than or equal to a procrastination interval computed through Jejurikar's method [12]. However, the objective of this research is not to propose another procrastination algorithm, rather to relax the assumption of external hardware required to implement such algorithms and reduce the online complexity. We compare the proposed algorithm with the procrastination scheduling and show the loss of energy saving of relaxing such assumption (external hardware) is within a range of $\approx 1\%$, while ignoring the energy overhead the external hardware would introduce.

V. Slack Management Algorithm

The processing time not used in a system is called slack. System slack can be categorized in two types, dynamic and static slack. The static slack exists due to spare capacity available in the system schedule. This spare capacity occurs as the system is loaded less than what can be guaranteed by the schedulability tests.

The dynamic slack occurs due to difference between worst-case assumptions made in the offline analysis and the actual online behavior of the system. It is further divided into two components based on two different worst-case assumptions. The first assumption is that each job of a task will execute for its WCET C_i. Due to the inherent pessimism in all WCET approaches, most if not all of the jobs in a real scenario finish their execution earlier than their C_i and by the chosen budget A_i, and thus generate extra slack. This kind of slack is termed as *execution slack* $\hat{S}_{i,m}$, and it is quantified by the difference in C_i and $\hat{c}_i$.

Similarly, the system is analyzed with the second worst-case assumption that each job of a sporadic task will be released as soon as possible i.e. released periodically with the minimum inter-arrival time. However, for truly sporadic tasks that rarely happens in hard real-time systems. Jobs of a sporadic tasks are released with a variable delay bounded by the minimum inter-arrival time. The slack generated due to sporadic delays is termed as *sporadic slack* and forms the second component of the dynamic slack. Naturally, all of the dynamic slack is generated online and can only be identified at run-time.

The execution slack and static slack are managed explicitly in our approach. Nevertheless, the effect of the sporadic slack is considered implicitly. Our slack management approach is based on the basic principles of [30]. The execution slack $\hat{S}_{i,m}$ available in the system at time instant t is represented by the tuple $S_t = \langle S_{t \cdot s}, S_{t \cdot d} \rangle$, where $S_{t \cdot s}$ corresponds to effective slack size and $S_{t \cdot d}$ corresponds to the absolute deadline of the slack. The term $S_{t \cdot d}$ defines the upper bound on the effectiveness of the slack S_t. In idle mode, the system consumes available slack [31].

Algorithm 1 Slack Management

```
1: On Every Scheduling Event
2: if ($S_{t\cdot d} \leq d_{i,m}$) then
3:     $a_{i,m} += S_{t\cdot s}$
4:     $S_{t\cdot d} = 0$
5:     $S_{t\cdot s} = 0$
6: end if
7: Slack Update On Job Completion
8: $S_{t\cdot s} += a_{i,m}$
9: $S_{t\cdot d} = max(S_{t\cdot d}, d_{i,m})$
10: if ($Ready\ Queue\ Empty$) then
11:     Consume slack $S_t$ first (This slack consumption accounts for a time spend without executing any workload
        irrespective of CPU state, i.e. idle or sleep mode)
12: end if
```

To preserve the system schedulability, we assume jobs with higher priority can only pass slack to jobs of same or lower priority. Whenever a job completes its execution it donates its slack to a slack container. A slack container stores and collates the slack identified at runtime. To reduce the complexity a single container is used for the slack management. Keeping several containers for the slack at different priority levels would add an extra online computational overhead in the slack management and is thus avoided.

Whenever execution slack is generated in the system by some job it is added to the slack container, and the later deadline is considered to be the effective deadline of the slack. Suppose a system receives an execution slack from a job $j_{i,m}$ of size SZ and deadline $d_{i,m}$. The deadline of the slack in the container $S_{t\cdot d}$ is updated by maintaining the later deadline ($S_{t\cdot d} = max\{S_{t\cdot d}, d_{i,m}\}$), while its size SZ is added to the slack size in the container $S_{t\cdot s}$ ($S_{t\cdot s} += SZ$).

As mentioned earlier only jobs with lower priority than slack can acquire the slack to preserve the system schedulability, therefore, on every scheduling event, the slack priority is compared against the current job priority. If the slack S_t has a higher or equal priority than the current job to be executed, the actual budget $a_{i,m}$ of the current job $j_{i,m}$ is incremented by $S_{t\cdot s}$; i.e. $a_{i,m} += S_{t\cdot s}$. The slack management in our approach will not pass slack to BE tasks. We assume BE tasks can borrow from its future jobs when their budget expires. Therefore, BE tasks are likely to consume the slack and we want to retain that for energy management purposes. When a slack S_t is allocated to any job $j_{i,m}$, the slack container is initialized to zero. The complete slack management algorithm is given in Algorithm 1.

The advantage of this approach is that slack generated at different priority levels are eventually accumulated implicitly with a very simplistic and transparent approach using just one container to hold the slack. The disadvantage of such approach is the poor distribution of the slack in the existence of long period tasks. The slack generate from such tasks will also decrease the priority of the already available slack and will results in poor slack distribution. The energy management algorithm proposed in the Section VI are not dependent on the slack management algorithm presented in this section (Algorithm 1). Any existing slack management algorithm can be integrated with minimal effort to the proposed RTH algorithms. Our slack management algorithm has low overhead (spatial/temporal) that makes it an attractive alternative.

VI. Energy Management Algorithms

We propose three different energy management algorithms to increase the energy efficiency of embedded systems using RTH strategy followed by a sleep state.

A. ERTH Algorithm (ERTH)

ERTH considers three different principles to initiate a sleep transition. These principles are based on the system status (i.e. either the system is idle or executing RT/BE task) and the capacity of the available slack in the system. This algorithm does not initiate a sleep state for less than static limit t_l to minimize the transitions overheads. The complete pseudo code of ERTH is given in Algorithm 2. The commons subroutines (such as Manage Slack, Get Slack, Set Sleep Time and Get Next Release Time) shared with other algorithms (extension of ERTH) are given in Algorithm 3.

Principle 1:: The principle one applies on the RT (SRT or HRT) task type. If any of the job of a RT task is at the head of the ready queue and is eligible for the slack $S_t \geq t_l$, a timer is initialized with static limit minus the wake-up transition-overhead time (i.e. $t_l - t_n^w$) and the system enters a sleep state until the timer expires. However, if the available S_t to the job is less than t_l, it is added to the budget of the job. The system performs a race-to-halt with the aim of collecting more slack in future.

Theorem 6: If the next job to execute in the ready queue $j_{i,m}$ is of type HRT or SRT, while the execution slack has a size greater than or equal to the static limit ($S_{t\cdot s} \geq t_l$) and the slack deadline is less than or equal to the absolute deadline $d_{i,m}$ of the HRT or SRT job $j_{i,m}$ ($S_{t\cdot d} \leq d_{i,m}$), then system can initiate a sleep state for a static limit of t_l without violating EDF schedulability.

Proof: Suppose the available S_t is considered as a task τ_{sleep} with a budget and deadline equal the t_l and $S_{t\cdot d}$ respectively. We need to prove τ_{sleep} is schedulable without an interruption in the presence of T. For this we split the potentially affected jobs of T in two parts which we address separately: 1)$\forall\tau_i$ not released yet. 2)$\forall\tau_i$ released but in ready queue.
Case 1 ($\forall\tau_i$ not released yet): This can be proven by contradiction. Suppose, the system schedule a τ_{sleep} for t_l and there is a synchronous arrival of all the tasks not yet released, and some of the τ_i missed their deadline. However from Theorem 2, all τ_i in the system can be delayed for an interval of t_l without any deadline misses, which is a contradiction. Therefore all τ_i not released yet will meet their respective deadlines.
Case 2 ($\forall\tau_i$ released and in ready queue): Due to the condition expressed in Principle 1, task τ_{sleep} has a deadline earlier than any τ_i in the ready queue. This task can be scheduled before its deadline and thus τ_{sleep} will not affect τ_i in the ready queue. We proved tasks in both cases do not violate the schedule, thus theorem holds. ■

Principle 2:: The principle 2 deals with BE task type. In case the job to execute is of type BE, we use Equation 20 to evaluate the possible sleep interval. The sleep interval computed through Equation 20 is the minimum of the available slack and the minimum gap ϱ. The latter (ϱ) is determined through Equation 21. Equation 21 computes the workload until the deadline of the slack and finds the time interval called minimum gap. This time interval ensures any deadline prior to the slack deadline is not violated if the schedule is delayed for such interval.

This approach has two main advantages. Firstly, the sleep interval estimated φ is always equal to or greater than t_l ($\varphi \geq t_l$) as we assume that φ is computed when $S_{t\cdot s} \geq t_l$. Secondly, it is useful, when t_l is very small (i.e. high system utilization). We do not assume knowledge about the previous release times of tasks, therefore a worst-case situation is assessed with Equation 21. The worst-case situation is when all higher priority tasks ($d_{i,m} \leq S_{t\cdot d}$) arrive just after system has initiated a sleep state. Jobs with deadlines after the deadline of the current job will not be affected, as the slack has a shorter or equal deadline to the current job. Another way to visualize the working of Equation 21 is through a demand bound function. Assume a synchronous arrival of all higher priority tasks at time instant t and compute the demand bound function within an interval of $[t, S_{t\cdot d}]$. The minimum gap ϱ is the one that has the smallest distance between budget demand and utilization bound.

The minimum gap identified by ϱ using Equation 21 does not relate to the schedulability of the lower priority tasks, as it may also contain the processing time reserved for those tasks. The amount of slack available in the system gives us an exact upper bound on the sleep duration. To avoid more complex schedulability checks, we assume a sleep interval is always less than or equal to the available slack even if the available gap ϱ is greater than the slack $\varrho \geq S_{t\cdot s}$. Conversely, if the gap ϱ is less than the system slack $\varrho < S_{t\cdot s}$, a sleep interval would be obviously equal to the duration of ϱ to ensure the schedulability of the higher priority tasks. Therefore φ finds the minimum between the available slack and the smallest-gap identified by ϱ ensuring overall system schedulability. Once a gap φ in the schedule is identified, the timer is set for an interval of $\varphi - t_n^w$.

Theorem 7: If the job to execute in the ready queue is of BE type and the execution slack is greater than or equal to the static limit ($S_{t\cdot s} \geq t_l$) with a deadline less than or equal to the absolute deadline of the BE job ($S_{t\cdot d} \leq d_{i,m}$), the system can initiate a sleep state for φ without violating any deadlines under EDF.

$$\varphi = \min\left(S_{t\cdot s}, \varrho\right) \tag{20}$$

Where

$$\varrho = \min_{k,m \in V(S_{t\cdot d})} \left\{ g_{k,m} - \sum_{j \in V(d_{k,m})} \left\lfloor \frac{g_{k,m}}{T_j} \right\rfloor \times C_j \right\} \tag{21}$$

$$g_{k,m} = d_{k,m} - t \tag{22}$$

$$\mathbf{V(x)} = \{i : r_{i,m} \geq t \wedge d_{i,m} \leq x\} \tag{23}$$

Proof: In this case the available sleep interval is not defined offline, rather computed online. To prove that a system can afford to hold the sleep state for an interval of φ, we segregated T into four parts. The schedulability of each part is proven individually.
1)$\forall j_{i,m}$ has not yet been released and $d_{i,m} \leq S_{t\cdot d}$
2)$\forall j_{i,m}$ has not been released and $d_{i,m} > S_{t\cdot d}$
3)$\forall j_{i,m}$ is in the ready queue
4)$\forall j_{i,m}$ has already completed
Let ϱ define the maximum available interval by which the higher priority jobs can be delayed at the current instant t. ϱ is computed by Equation 21 considering each deadline within an interval of $[t, S_{t\cdot d}]$. In principle it performs a limited demand-bound analysis for the defined interval to calculate the delay interval. Since there is a possibility to get a delay larger than the available $S_{t\cdot s}$, Equation 20 guarantees system is not delayed more than available slack capacity. Assume a sleep interval as a task τ_{sleep} with a deadline equal to $S_{t\cdot d}$. Equation 21 implies scheduling a τ_{sleep} for not more than ϱ does not affect the schedule of any $j_{i,m}$ that is not yet released and has a $d_{i,m} \leq S_{t\cdot d}$. Moreover, we restrict that

τ_{sleep} will not execute for more than $S_{t \cdot s}$ with Equation 20. This ensures that any $j_{i,m}$ not released yet with $d_{i,m} > S_{t \cdot d}$ will not be affected. Equation 23 exclude all $j_{i,m}$ such that $d_{i,m} \leq t$. Similar to Theorem 6, the schedulability of $\forall j_{i,m}$ in the ready queue is not affected as well, as they have a deadline later than that of τ_{sleep}. Any jobs already completed, are obviously unaffected. As none of the task in T miss their deadline, hence theorem holds. ■

Principle 3:: In idle state the system uses principle 3. As we know, t_l is computed offline considering the worst-case scenario in the schedule. Thus initiating a sleep state for $t_l - t_n^w$ during idle mode will not affect the schedulability in any circumstance. Moreover, the system is not allowed to prolong the sleep state beyond the static limit to preserve the schedulability. However, φ could be used to increase the sleep interval, it would also substantially increase the complexity of the algorithm.

Theorem 8: If the system is idle it can initiate a sleep state for the static limit t_l without violating the EDF schedulability, assuming the available slack S_t that may be less than the static limit t_l is consumed first.

Proof: The proof of the Theorem 8 follows the same reasoning of Theorem 2. ■

The timer value is set equal to $sleep\ interval - t_n^w$, where $sleep\ interval$ is determined based on which principle has initiated a sleep state. The sleep transition due to any of these principles restricts the system to wake up until the timer expires. This restriction applies to all higher priority tasks as well. We assume that all interrupts bar the timer interrupt are disabled on initiating a sleep state and re-enabled on completion of the sleep. In many CPUs separate interrupt sources can be used for this. As usual with such disabled interrupts, events occurring during the sleep interval are to be flagged in the interrupt controller for processing after the interrupts are re-enabled.

The ERTH algorithm is agnostic to future release pattern of the system. It assumes a critical instant on each sleep transition. As such, each sleep state interval is estimated assuming a synchronous release of all higher priority tasks. For instant, in calculation of φ, system assume synchronous release of all those tasks having deadlines earlier than the current job's deadline. Similarly, in principle 3 (idle mode), a synchronous release of all higher priority tasks is assumed at the instant of sleep transition. The critical instant occurs rarely, if ever, in reality. However, ERTH has to consider this

Algorithm 2 Enhanced Race-To-Halt Algorithm (ERTH)

```
1: Offline
2: Compute $t_l$
3: Find most efficient sleep state $n$ for $t_l$:
   $\forall$ Sleep States $N : t_l \geq t_n^e$
   Minimize $\{(t_l * P_n) + (t_n * (P_A - P_n))\}$
4: Let $\Phi$ be the sleep interval such that $\Phi = t_l - t_n^w$
5: Online
6: if ($System\ Idle$) then
7:    Manage Slack($t_l$)
8:    $Timer = \Phi$
9:    Mask-record interrupts and initiate Sleep
10: else if ($GetSlack(j_{i,m}) \geq t_l$) then
11:    if ($HRT/SRT\ Task$) then
12:       Manage Slack($t_l$)
13:       $Timer = \Phi$
14:       Mask-record interrupts and initiate Sleep
15:    else if $BE\ Task$ then
16:       Compute $\varphi$
17:       Manage Slack($\varphi$)
18:       Set Sleep Time($\varphi$);
19:    end if
20: else
21:    Race-To-Halt
22: end if
23: When Timer Expires
24: Unmask interrupts
25: if (interrupts) then
26:    Service the interrupts (Schedule the tasks arrived during sleep duration)
27: else
28:    $Timer = \Phi$
29:    Mask-record interrupts and initiate Sleep
30: end if
```

Algorithm 3 Common Routines for ERTH, IRTH and LWRTH

```
1: **Set Sleep Time**($\eta$)
2: $\forall$ Sleep States $N$:$\eta \geq t^e_n$
3: Minimize $\{(\eta * P_n) + (t_n * (P_A - P_n))\}$
4: $Timer = \eta - t^w_n$
5: Mask-record interrupts and initiate Sleep

6: **Get Slack**($j_{i,m}$)
7: if $d_{i,m} \geq S_{t \cdot d}$ then
8:     return $S_{t \cdot s}$
9: else
10:    return 0
11: end if

12: **Manage Slack**($\eta$)
13: if ($\eta \leq S_{t \cdot s}$) then
14:    $S_{t \cdot s} -= \eta$
15: else
16:    $S_t = 0$
17: end if

18: **Get Next Release Time**($r^n$)
19: $\forall i \in$ T
20: return $min(r^n_i)$
```

pessimistic condition to guarantee the system schedulability in hard RT systems. This pessimism results in a sub-optimal sleep interval.

B. Improved Race-To-Halt Algorithm (IRTH)

One can eliminate the pessimism of ERTH by knowing the future release information of the task-set. As we assume sporadic task model, it is not easy to predict exact future releases of all tasks. However, one can approximately predict the future based on past release-pattern of the tasks. In the sporadic system model, a new job of a task can only arrive after T_i. Therefore, by storing the past release information, we can approximate the future release time. This method can reduce the pessimism introduce in ERTH but cannot eliminate it entirely due to the sporadic task model employed.

This algorithm predicts the future release pattern of the task based on past release information. This can easily be implemented by maintaining an array of future release information r^n with a size equal to the number of tasks n in the system. When any job of a task arrives, it updates it future release time r^n_i by adding the T_i in its current job release time $r_{i,m}$ (i.e. $r^n_i = r_{i,m} + T_i$).

The Algorithm 4 presents IRTH. The three basic principles stays same when compared to ERTH. However, the sleep interval estimation varies in principle 2 (when the task to execute is of type BE) and principle 3 (idle mode). In idle mode, system finds the next earliest release r_{next} in the future from its future release information array r^n. This information assures there is no release in an interval $[t,\, r_{next})$, hence a sleep interval can be extended from t_l to $t_l + r_{next} - t$ without violating any deadline.

Theorem 9: In idle mode, the system can initiate a sleep state for a duration of $r_{next} + t_l - t$, without violating any deadline under EDF, given the earliest future releases of all tasks r^n is known at the time of initiating a sleep state.

Proof: Assume t is the current instant where the system became idle. Consider r_{next} is the next release of any task in T. r_{next} is assumed to be the critical instant, that leads to the longest busy interval (assuming synchronous releases) in the system (though that may or may not occur at this point). As there are no releases between r_{next} and t interval, system schedulability is not affected, however, we need to check for the schedulability of the task-set T in the duration of r_{next} and $r_{next} + t_l$. The schedulability of this interval follows directly from 2. Hence the schedulability of the overall system will be preserved under this sleep condition. ■

Similarly, in principle 2 future release information can improve on the sleep interval. As we know, principle 2 deals with BE task type and computes its sleep interval with Equation 21 which in turn is equivalent to limited demand bound function. It assumes critical instant at time t and considers all job releases having deadline less than or equal to $S_{t \cdot d}$. However, having predicted information about the future releases of the tasks, we can add an offset of $r^n_i - t$ to the first job of all those tasks having future releases in an interval $[t,\, S_{t \cdot d}]$ and not currently awaiting in the ready queue. We do not include jobs of the tasks awaiting in the ready queue, because by the definition of principle 2, they have deadlines

Algorithm 4 Improved Race-To-Halt Algorithm (IRTH)

```
1: Offline
2: Compute $t_l$
3: Find most efficient sleep state $n$ for $t_l$:
   $\forall$ Sleep States $N : t_l \geq t_n^e$
   Minimize $\{(t_l * P_n) + (t_n * (P_A - P_n))\}$
4: Let $\Phi$ be the sleep interval such that $\Phi = t_l - t_n^w$
5: Online
6: if $(System\ Idle)$ then
7:    Manage Slack($t_l$)
8:    $r_{next}$ =Get next release time($r^n$)
9:    Set Sleep Time($r_{next} - t + t_l$)
10: else if $(GetSlack(j_{i,m}) \geq t_l)$ then
11:    if $(HRT/SRT\ Task)$ then
12:       Manage Slack($t_l$)
13:       $Timer = \Phi$
14:       Mask-record interrupts and initiate Sleep
15:    else if $BE\ Task$ then
16:       Compute $\Omega$
17:       Manage Slack($\Omega$)
18:       Set Sleep Time($\Omega$)
19:    end if
20: else
21:    Race-To-Halt
22: end if
23: On release of $\tau_i$
24: Update $\tau_i$ next predicted arrival time in the future release array $r^n$ i.e. $r_i^n = r_{i,m} + T_i$
25: When Timer Expires
26: Unmask interrupts
27: if (interrupts) then
28:    Service the interrupts (Schedule the tasks arrived during sleep duration)
29: else
30:    $Timer = \Phi$
31:    Mask-record interrupts and initiate Sleep
32: end if
```

later than $S_{t \cdot d}$. The offset is only added if the r_i^n of the τ_i is greater than t. Otherwise, it is assumed to be 0. The offsets greater than 0 shifts the jobs deadlines accordingly – which may or may not shift the last job deadline of some tasks outside the interval $[t,\ S_{t \cdot d}]$. If some of the jobs deadlines move outside the interval, the demand requested by the system in the interval $[t,\ S_{t \cdot d}]$ is decreased. Which in turn increases the possibility to get larger sleep interval compared to pessimistic approach used in ERTH. The schedulability of system in principle 2 with this new amendment is proved in Theorem 10.

Theorem 10: If the task to execute in the ready queue is of BE type and the available slack is greater than or equal to the static limit $S_{t \cdot s} \geq t_l$ with a deadline less than or equal to the static limit $S_{t \cdot d} \leq d_{i,m}$, then sleep state could be initiated for a time interval Ω without violating any deadline under EDF, given the earliest estimated future releases of all tasks r^n is known at the time of initiating a sleep state.

$$\Omega = \min\left(S_{t \cdot s}, \vartheta\right) \tag{24}$$

Where

$$\vartheta = \min_{k \in V(S_{t \cdot d})} \left\{ g_{k,m} - \sum_{j \in V(d_{k,m})} \left\lfloor \frac{g_{k,m} - r_j^n}{T_j} \right\rfloor \times C_j \right\} \tag{25}$$

$$g_{k,m} = d_{k,m} - t \tag{26}$$

$$\mathbf{V}(\mathbf{x}) = i : r_{i,m} \geq t \wedge d_{i,m} \leq x \tag{27}$$

Proof: The scheduler estimates the sleep interval online considering the available dynamic parameters in the system. In order to prove the schedulability, we segregate T into following six parts. We individually prove the schedulability of each part.
1) $\forall j_{i,m}$ already completed
2) $\forall j_{i,m}$ released earlier than t and has $S_{t \cdot d} > d_{i,m} > t$
3) $\forall j_{i,m}$ released earlier than t and has $d_{i,m} > S_{t \cdot d}$
4) $\forall j_{i,m}$ that will be released after t with an initial offset of r_j^n and has $d_{i,m} \leq S_{t \cdot d}$
5) $\forall j_{i,m}$ that will be released after t with an initial offset of r_j^n and has $d_{i,m} > S_{t \cdot d}$
6) $\forall j_{i,m}$ that will be released after $S_{t \cdot d}$
The term ϑ given by Equation 25 estimates the worst-case response-time of all jobs in an interval of $[t, S_{t \cdot d}]$ having release times in an interval of $[t, S_{t \cdot d})$ and deadlines in an interval of $(t, S_{t \cdot d}]$. Moreover, it returns the feasible interval for the sleep state at time instant t. Consider a sleep state as a task τ_{sleep} with a deadline $S_{t \cdot d}$ and budget $S_{t \cdot s}$. The sleep state cannot be initiated for more than $S_{t \cdot s}$, as it might jeopardize the schedulability of the low priority tasks. With this restriction, scheduling the τ_{sleep} will not affect the jobs of category $1, 3, 5$ and 6 considering the EDF algorithm. Equation 27 eliminate all these jobs from the analysis. The principle 2 is only invoked when the task to execute in the ready queue has $d_{i,m} \geq S_{t \cdot d}$. This restriction is imposed by our slack management algorithm. Hence jobs with a category of 2 do not exist and are thus removed with this restriction from the analysis. The schedulability of the jobs in the category 4 is individually ensured with the Equation 25. Equation 25 can get a sleep interval larger than $S_{t \cdot s}$, but we are assuming a sleep task that is equal to $S_{t \cdot s}$. Therefore, its size is restricted to $S_{t \cdot s}$ with the use of Equation 24. As none of the tasks in T misses its deadline, the theorem holds. ■

C. Light-Weight Race-To-Halt Algorithm(LWRTH)

Though IRTH is promising, it also has an extra online overhead when compared to ERTH. (Online/Offline overheads are discussed in Section VII.) We also propose another algorithm that does not require any slack management scheme but performs slightly better than ERTH and marginally lower than IRTH. In this algorithm, system races-to-halt when there is a task in the ready queue and waits for the idle mode. In idle mode we initiate a sleep state using principle 3 of the IRTH algorithm. LWRTH has lower online complexity when compared to IRTH but is inferior (performance-wise) against it at high utilizations. However, LWRTH needs to maintain a list for the predicted future release information of tasks that adds extra online overhead and does not give us a full control over the sleep transition which might not be helpful in a thermal aware system design. For instance, if the system crosses the maximum temperature threshold in the middle of the execution phase, LWRTH has no way to slow down or stop its execution. However, thermal-aware constraints are intended for future research. Furthermore, LWRTH performs worse compare to ERTH when it comes to the number of pre-emptions for small task-set sizes (discussed in Section IX-C). The pseudo code of LWRTH is given in Algorithm 5.

VII. Offline vs Online overhead

The complexity of our algorithms is compared with LC-EDF and SRA, as they are with their use of dynamic priorities closest to our work. As it has been discussed in Section II, in LC-EDF, the system enters a sleep mode whenever it is idle.

Algorithm 5 Light-Weight Race-To-Halt Algorithm (LWRTH)

```
1: Online
2: if ($System\ Idle$) then
3:     $r_{next}$ =Get next release time($r^n$)
4:     Set Sleep Time($r_{next} - t + t_l$)
5: else
6:     Race-To-Halt
7: end if
8: On release of $\tau_i$
9: Update $\tau_i$ next predicted arrival time in the future release array $r^n$ i.e. $r_i^n = r_{i,m} + T_i$
10: When Timer Expires
11: Unmask interrupts
12: if (interrupts) then
13:     Service the interrupts (Schedule the tasks arrived during sleep duration)
14: else
15:     Set Sleep Time($t_l$)
16:     Initiate Sleep
17: end if
```

LC-EDF has a smaller number of sleep states when compared to EDF as it combines several small idle intervals to initiate sleep state for long interval. While in the sleep state, on each higher priority (shorter deadline) task arrival, the algorithm recomputes the new procrastination interval for that task, unless the system does not allow further procrastination. The online overhead of the LC-EDF algorithm depends on two main factors, 1) Number of times a sleep state is initiated in the system, 2) The overhead of each sleep transition. The first factor depends on the total number of idle intervals in the schedule because LC-EDF initiates a sleep in idle mode. However, the overhead of each sleep transition depends on the task-set size. The complexity of each sleep transition in LC-EDF is $O(l^2)$, where l denotes the task-set size.

The SRA algorithm [15] is developed on the same idea of LC-EDF. Instead of computing the procrastination interval offline, it determines this interval for each task offline. This approach also reclaims the execution slack from the system and use it to further procrastinate the sleep interval. In a nutshell, on every release of a task during the sleep interval, the scheduler computes the available execution slack and compares it with the offline computed procrastination interval of that task. The maximum of these two values is considered while deciding on the reinitialization of the timer. The complexity of the system to determine the available execution slack for the task is $O(l)$. In worst case l tasks can arrive during each sleep interval. Therefore, the complexity of the SRA algorithm is same as LC-EDF i.e. $O(l^2)$.

The major drawback of LC-EDF and SRA lies in their dependency on the external hardware to compute the procrastination interval online. The authors assume that their algorithm is implemented on the external hardware such as an ASIC or FPGA. The external hardware is required as the procrastination interval is readjusted on every new task release while the processor is in a sleep mode. The logic needed to implement such algorithms is not simple as well, as it needs to time stamp and store all interrupts arrived during the sleep interval. Such external hardware obviously has its own energy cost and partly negates what LC-EDF or SRA is aiming to achieve.

While on the other side, the ERTH proposed the more effective power saving algorithm with lower complexity. Initially, we discuss the complexity of ERTH. It is divided into three different categories based on its three different principles. Firstly, if the sleep transition is initiated through Principle 1, it requires just one comparison against the offline computed static limit t_l, i.e. $O(1)$. Secondly, sleep states initiated with Principle 2 requires the computation of φ in order to obtain the maximum feasible sleep interval. The major overhead lies in the computation of ϱ that could be obtained either offline or online. Offline Method: The interval for computing ϱ offline is no more than the longest T_i in the task-set. Therefore, the maximum available gap can be computed offline for each deadline and sorted in an increasing order by time. The runtime overhead is to search the sorted array of maximum available gaps for each given interval, which can be done in $O(ln(p))$, where p is the number of intervals. Online Method: The online complexity to compute ϱ depends on the number of jobs in an interval. We have used the former method to compute ϱ. The overhead of φ computation is justified when compared to its energy saving and benefits of using it. The major advantage is that it is computed only once before initiating the sleep state. Thirdly, in idle mode (Principle 3), sleep state is initiated for t_l interval without any check. Thus, sleep states initiated in idle mode do not have any online overhead.

Apart from its low complexity, the second advantage of ERTH is the existence of the fixed sleep-interval at the sleep-state initialization instant. Once the system initiated the sleep transition, no matter how many tasks arrive during the sleep mode, the system will wake up after a defined limit (when timer expires). Our schedulability tests ensure all the jobs will meet their deadlines. This mechanism simplifies the system implementation and eliminates a need for external hardware to compute the algorithm. Which in turn further reduce the complexity of the design, as external hardware require extra communication overhead and increases integration issues.

The online overhead of IRTH is similarly divided into three categories. If the sleep state is initiated by a RT task (Principle 1), its overhead is same as in ERTH Principle 1, i.e. $O(1)$. However, in idle mode (Principle 3), its complexity increases, as the system has to search for the earliest possible future release in an array of r^n. There are two ways to manage it. Firstly, we can keep a sorted array of r^n and use the first value when system initiates a sleep transition. Thus the complexity of maintaining the array on each job arrival is $O(ln(l))$. However, when system initiates a sleep the overhead is low i.e. $O(1)$. Secondly, r^n can be stored with respect to the task-ID and on each sleep invocation system traverse r^n to find the minimum value. In this case complexity to update an array of r^n on each job invocation is $O(1)$, however, each sleep transition has a complexity of $O(l)$. In our observation, the number of sleep transitions are fewer when compared to the number of jobs invocations. Therefore, we used the second approach. Thus the complexity of each sleep transition in IRTH through Principle 3 is $O(l)$. In Principle 2 of IRTH, we wanted to exploit the online information of future releases (r^n). Therefore, it is difficult to find the sleep interval offline, and we have to estimate it online on each sleep invocation. To compute the complexity of a sleep transition in principle 2, we assume $\Theta = \frac{T_{max}}{T_{min}}$, where T_{max} is the maximum and T_{min} is the minimum inter-arrival time in the task-set. Then the maximum complexity of each sleep transition in Principle 2 is $O(\Theta l)$, as in worst-case system has to check the all possible job releases within T_{max}.

To cope with additional online complexity of IRTH, we have devised LWRTH. LWRTH only initiates sleep states in idle mode. It relies on the future release information array to maximize the energy efficiency. Therefore, each sleep transition happening in LWRTH has a complexity of $O(l)$. This algorithm does not need any slack management algorithm, and moreover its online complexity to initiate a sleep transition is also low when compared to ERTH and IRTH. A system designer needs to perform a careful evaluation, while selecting among the available algorithms. IRTH clearly has the highest complexity when compared to ERTH and LWRTH but the energy saving is also greatest among them. The complexity comparison of ERTH and LWRTH is difficult. On on side, ERTH does not require to maintain a list of the future release information, while IRTH needs such information which needs to be updated on every task's release. On the other hand, LWRTH has lower sleep transition overhead when compared to ERTH and does not use slack management.

Suppose a, b and c are the number of sleep transitions in Principle 1, 2 and 3 respectively and n is the total number of sleep transitions in the system. Then overall online complexity of ERTH, IRTH and LWRTH to initiate a sleep transition is summarized in Equation 28, Equation 29 and Equation 30 respectively.

$$a\,O(1) + b\,O(ln(p)) + c\,O(0) = b\,O(ln(p)) \quad (28)$$

$$a\,O(1) + b\,O(\Theta l) + c\,O(l) = b\,O(\Theta l) + c\,O(l) \quad (29)$$

$$n\,O(l) \quad (30)$$

A. Extensions in LC-EDF

Apart from this, a minor modification is also proposed for LC-EDF that simplifies its implementation and reduces its online complexity. The improved form is shown in Equation 31, where $\mathbf{W}(l)$ is the set of indices of all tasks in the ready queue, that re-evaluated LC-EDF, from its last activation. The total system utilization is normally known offline/or can be computed once on system mode change. Therefore, in sleep mode, the system just needs to accumulate δ_i during each sleep interval.

$$\Delta_l = T_l \times \left(1 - U - \sum_{\forall i \in \mathbf{W}(l)} \frac{\delta_i}{T_i}\right) \quad (31)$$

VIII. Effect of Sleep-States On the Number of Pre-emptions

A side effect of the use of the sleep states is that the release behavior of the system and subsequently the pre-emption relations between tasks are affected. In this section how the behavior in terms of number of pre-emptions of tasks in the system is changed at runtime is investigated.

A sleep state is initiated for an interval greater than break-even time, as the overheads of transitioning into and out of a sleep state are still substantial. The tasks releases during sleep interval give rise to two conflicting scenario's. On one side, execution of the tasks releases during the sleep-state interval are postponed and constrained to a smaller window for execution. One could easily perceive that the number of pre-emptions will rise, as delaying the tasks execution increase the likelihood of higher priority tasks releases. Therefore, in the presence of low priority tasks, higher priority tasks cause more pre-emptions. On the other side, the interrupts that occur throughout the sleep state interval are served on completion of the sleep interval. We consider that a task release is triggered by an interrupt. Therefore, tasks releases during sleep interval are collated and scheduled after the sleep state. Thus delaying new tasks arrival and waiting for the higher priority task releases decreases the number of pre-emptions. Thus these two considerations indicate positive or negative changes in the number of pre-emptions.

The number of pre-emptions poses a substantial overhead (time/energy) on the running system. For instance, on resumption of a task the system has to pay the penalty to reload the cache content displaced by pre-emption. Access to off-chip memory is generally very expensive when compared to on-chip caches or scratch-pad memory. Therefore, the system has to reserve time for pre-emption related delays, which in turn also decreases the system utilization. A decrease in pre-emption count not only increases the system utilization but also reduces the energy consumption.

We assume the pre-emption count represents the number of pre-emptions taking place when a actively executing task is being replaced before it has completed execution by a higher priority task to execute. Suppose we have synchronous releases of one high and one low priority task. The high priority task executes first, but the pre-emption is not counted, as the low priority task has not yet started its execution.

Considering the overhead of pre-emptions on the energy consumption and system utilization, it is indeed an important issue to resolve which approach performs better. If the number of pre-emptions decreases, the overall energy consumption actually decreases more than just energy saved with sleep transition, as we reduced the overhead of pre-emptions as well. Through extensive simulations we have shown in our results that, in the average-case, sleep states have a positive effective on the number of pre-emptions. The number of pre-emptions of all algorithms (SRA, LC-EDF, ERTH, IRTH and LWRTH) are analyzed and compared in our results section.

IX. Evaluation

A. Experimental Setup

In order to evaluate the effectiveness of the proposed algorithms, we have implemented all the algorithms (ERTH, IRTH, LWRTH, SRA and LC-EDF) in SPARTS (Simulator for Power Aware and Real-Time System) [32]. SPARTS is an open source simulator of a real-time device and its source code is available at [33].

Table I
OVERVIEW OF SIMULATOR PARAMETERS

Parameters	Values
Task-set sizes $\|T\|$	$\{10, 50, 200\}$
Share of RT/BE tasks $\xi = \{\xi_1, \xi_2\}$	$\{\langle 40\%, 60\%\rangle, \langle 60\%, 40\%\rangle\}$
Inter-arrival time T_i for RT tasks	$[30ms, 50ms]$
Inter-arrival time T_i for BE tasks	$[50ms, 1sec]$
Sporadic delay limit $\Gamma_x \in$	$\{0.1, 0.2\}$
Best-Case execution-time limit C^b	0.2
Sleep Threshold Ψ_x in	$\{1, 2, 5, 10, 20\}$

Table II
DIFFERENT SLEEP STATES PARAMETERS

No.	Power Mode	t_n (μs)	t_n^e(μs)	Power	E_n
1.	Doze	10	225	3.7	42
2.	Nap	200	450	2.6	950
3.	Sleep	400	800	2.2	1980
4.	Deep Sleep	1000	1400	0.6	5750
5.	Typical	0	0	4.7	0
6.	Maximum	-	-	12.1	-

To cover a wide range of different systems, different task-sets are evaluated from a large number of fine grained small tasks (200) to a small number of coarse grained tasks (10). The two different share distributions ξ_1 and ξ_2 are applied to the number of tasks in a given class, as well as the overall utilization of the respective task classes. Moreover, utilization allocated to a specific task class is distributed randomly among the tasks of this class. The actual individual utilization per task is generated such that the target share for each scheduling class is achieved. Starting from the utilization U_i and T_i for each task according to the limits in Table I, the WCET of each task is deemed to be $C_i = U_i * T_i$. It has to be noted that due to numerical rounding in the parameters used in our simulator to generate the task-set with a target utilization of x has a resulting utilization of $x - \epsilon$, where ϵ is a very small number.

Beyond those initial settings a two level approach is used for generating a wide variety of different tasks and subsequently varying jobs. Tasks are further annotated with a limit on the sporadic delay Δ_i^s in the interval $[0, \Gamma_x * T_i]$ and on the best-case execution time C_i^b in the interval $[C^b * C_i, C_i]$. However, not only tasks vary in their requirements, different jobs of the same task have also varying behavior dependent on system state and input parameters. This is modeled by assigning each $j_{i,m}$ an actual sporadic delay in $[0, \Delta_i^s]$ interval and an actual execution time in $[C_i^b, C_i]$ interval.

Though not a fundamental requirement of our proposed algorithms, we assume implicit deadlines $D_i = T_i$ for evaluation purposes. It is obvious that $D_i > T_i$ leads to greater saving opportunities, but does not provide greater insights. All random numbers are taken from a uniform distribution and unless explicit values are given, random numbers are used for all assignments.

A vast variety of CPUs are available in the market. They have diverse hardware architectures and consequently different power characteristics. In order to observe the effect of different types of hardware platforms on the proposed algorithms, we have generated different power parameters of the processor. In our system model active and idle time of the CPU remain constant for a specific task-set. The factor among the power model parameters that affects the energy gain of an algorithm is the overhead of the sleep transitions as we have normed the total energy consumption. However, the overhead of the sleep transition is modeled by the break-even-time of the sleep state. Therefore, we have altered the power model parameters to generate the distinct BET such that it is a multiple of the original BET by a factor of x. The different break-even-times are represented with Ψ_x called sleep threshold. The sleep threshold with a value of $x = 1$ denotes the BET of the original power model. The above mentioned varied parameters are summarized in Table I.

Overall system utilization is varied from 0.2 to 1 with an increment of 0.05. We have generated 1020 different task-sets configurations ($C^b, \Gamma_x, U_i, \cdots$ etc). For each task-set of a particular configuration the seed value of the random number generator is varied from 1 to 100. Therefore, in total we simulated 102 thousand task-sets of the above mentioned different parameters and each task-set is simulated for 100 seconds. The overhead of all the algorithms including procrastination algorithms (LC-EDF and SRA) is considered negligible. This is obviously a favorable treatment for LC-EDF and SRA, as the time/energy overhead of the external specialized hardware is substantial. The SPARTS simulator takes into account the effect of the sleep state transition delays and its energy/time overhead is included in our power model.

The power model used for our simulations is based on the Freescale PowerQUICC III Integrated Communications Processor MPC8536 [34]. The power consumption values are taken from its data sheet for different modes (Maximum, Typical, Doze, Nap, Sleep, Deep Sleep). We assume a core frequency of 1500 MHz and core voltage of 1.1V. As the transition overheads are not mentioned in their data sheet, we assumed the transition overhead for four different sleep states. The transition overhead of the typical mode is considered negligible. The overhead t_n^e for the four different sleep states are computed using 1 and shown in the Table II. The power values given in Table II sum up core power and platform power consumption. The interested reader is directed to the reference manual [34] for details.

B. Energy Consumption Results

Any change in the parameters from those described above in Section IX-A is explicitly mentioned in the individual experiment description. Each point in the figure present results averaged over 100 runs with different respective seed values as well as all different free parameters. As baseline, we have simulated ERTH without the use of sleep states (NS), i.e. system uses typical power when it is not executing any task otherwise consume P_A during normal execution of tasks. Energy consumption of LC-EDF or SRA without sleep state is identical to NS as the overall idle and active execution time remains same in both cases. All the results are normalized to the corresponding results of NS.

We have used the RBED framework for the integration of applications with different criticality levels. The RBED framework allows an overrunning job to borrow from its future invocations [30]. Therefore, we have simulated two different scenarios. In scenario 1, we assume that $A_i = C_i$ for both task classes (RT and BE task). Moreover, we also assume $\Gamma_{0.1}$ for all experiments in scenario 1, as the difference is marginal when compared to $\Gamma_{0.2}$. Nevertheless, the effect of variation in Γ is explained later in scenario 2. In scenario 2, we assume BE tasks often overrun beyond their allocated periodic budget A_i. The mean of the BE tasks actual-execution-time distribution is set to 85% of A_i in this scenario. However to ensure timely completion of RT tasks, we assume $A_i = C_i$. The borrowing mechanism [30] is also integrated in scenario 2 so that BE tasks can use their future budgets, if required.

1) Scenario 1 *($A_i = C_i$, $\forall$ task types):* The minimum sleep threshold value Ψ is set to 1 for the next six experiments. The total energy consumption of ERTH is compared against LC-EDF and SRA for a task-set size of 200 and a task distribution of ξ_1 in Figure 3. All the values are normalized to the corresponding values of NS. As it is evident, SRA performs comparable to ERTH except at hight utilizations. Moreover, ERTH outperforms LC-EDF for all, but particularly for higher utilizations. With an increase in system utilization, the maximum feasible idle interval (procrastination interval) computed by the LC-EDF algorithm shrinks. Our system model assumes multiple sleep states, while LC-EDF cannot use more energy efficient sleep states with corresponding higher overhead t_n^e since it will risk system schedulability. The SRA algorithm saves more energy when compare to LC-EDF. Firstly, the procrastination interval computed for each task in SRA is greater than or equal to the procrastination interval determined by the LC-EDF algorithm. This increase in procrastination interval over LC-EDF enables SRA to select efficient sleep state offline while ensuring system schedulability. Secondly, it also benefits from the execution slack reclaimed online. On the other hand, the efficient slack management algorithm described in Section V also enables ERTH to accumulate the slack S_t and still use more efficient sleep states at high utilization. However, at high utilizations (especially at $U = 1$), the savings of ERTH are still larger when compared to SRA. This is motivated by the following observations. As already mentioned in the experimental setup, the resulting utilization is less than the target utilization by a very small factor of ϵ due to numerical rounding of the parameters used to generate a task-set. The secondary effect is the diversity in periods of task-set that rarely aligns and hyper-period of the given task-set is very long. Therefore, at high utilizations, the use of the demand bound function yields an actually usable t_l due to the disparity of periods and deadlines. If one uses the utilization based approach SRA, analytically this leads invariably to small intervals, due to the loss of accuracy when abstracting the workload through its worst-case utilization. An example of this is reflected in the proof to Theorem 5. At $U = 1$, ERTH creates idle intervals to save energy due to execution slack in the system. For a distribution of ξ_2, system consume approximately 1% more energy when compared to ξ_1. In ERTH it is due to the lesser usage of principle 2, as the system has fewer BE tasks in ξ_2. The LC-EDF and SRA algorithms depend on the period of the tasks. Extra tasks with long periods resulting in greater opportunities to save energy, therefore, ξ_2 consumes slightly more energy when compare to ξ_1.

An interesting observation is noticed in the total energy consumption of LC-EDF that the fine-grained large task-sets consume more energy when compared to the coarse-grained small task-sets at the same utilization. Hence, LC-EDF is susceptible to the changes in the task-set size. The figure is not shown here but its main features are described in details. The variation in the total energy consumption is up to 4% at higher utilizations. The major reason of this variation in total energy consumption lies in the algorithm itself, which computes the procrastination interval. The procrastination interval is recomputed on every arrival of a job with deadline shorter than any of the currently delayed jobs. An increase in the task-set size means a higher probability of recomputing the procrastination interval. Each re-computation includes a nominal shortening of the procrastination interval and increasing the virtual utilization in the process. Therefore, the energy consumption marginally increases with an increase in task-set size. The effect of task-set variation is also analyzed for ERTH, IRTH and LWRTH. Oppose to LC-EDF, task-set variation does not affect the total energy consumption of the system with either of them. Consequently, ERTH and its variants are more robust, when it comes to task-set size variations in the system. The task-set size variation has negligible effect on the energy consumption of SRA, as it pre-computes the procrastination interval offline and exploits execution slack.

The overall-gain of ERTH and SRA over LC-EDF for three different task-set sizes is gauged in Figure 4 with a distribution of ξ_1. The formula used to compute the overall-again is $\frac{E_{LCEDF}-E_x}{E_{LCEDF}}$, where E_{LCEDF} is the total energy consumption of LC-EDF and E_x corresponds to the total energy consumption of SRA or ERTH. It is evident ERTH

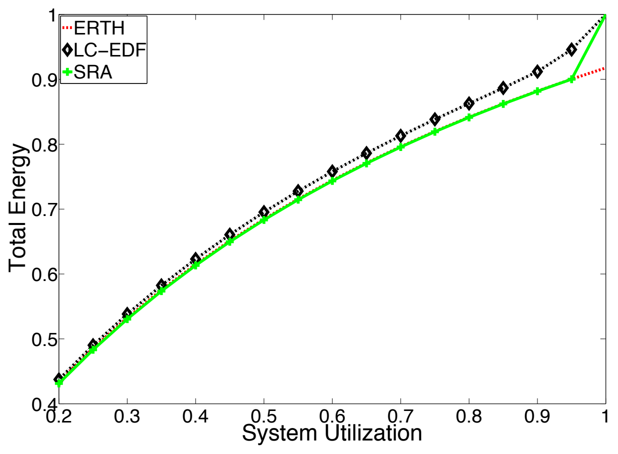


Figure 3. Normalized Total Energy Consumption (ξ_1 and $|\mathrm{T}| = 200$)

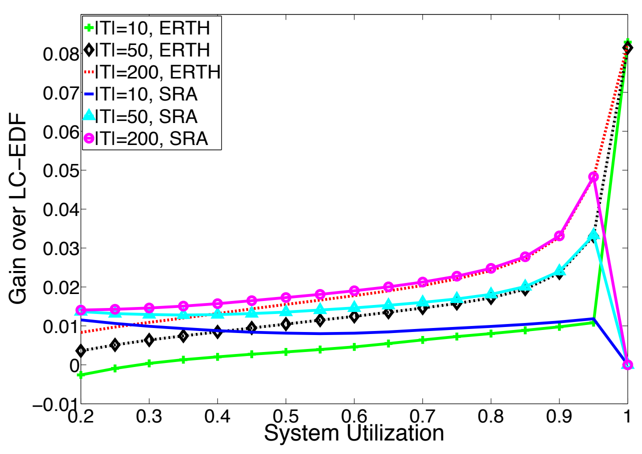


Figure 4. Gain of ERTH and SRA over LC-EDF for Different Task-Set Sizes

saves more energy compared to LC-EDF for larger task-set sizes. This happens due to dependency of LC-EDF on the task-set size as described above. However, SRA saves approximately 1% more energy when compared to ERTH at low utilizations. Its performance degrades towards high utilizations and the difference between SRA and ERTH slowly vanishes. The ERTH algorithm manages to saves this energy without the support of extra hardware. If the energy consumption of the external hardware is more than 1% of the saving, then ERTH is still a better approach in terms of energy saving due to less complexity comparing to SRA. The difference between two distributions ξ_1 and ξ_2 is small. Nevertheless, for all three different task-set sizes, the gain of ξ_2 dominates ξ_1 due to an increase in RT and decrease in BE tasks. One oddity exits at $U \leq 0.2$ for $|T| = 10$ as LC-EDF slightly performs better when compared to ERTH, but that difference is negligible. At such a low utilization, the system is consuming very little energy in any case.

The amount of execution performed by ERTH, SRA and LC-EDF is the same. Hence, the only difference arises from the difference of power consumption in the idle intervals of the schedule. Apart from the overall-gain that is shown in the Figure 4, we have also analyzed the gain of ERTH and SRA over LC-EDF only in the idle intervals. Simulation results are presented in Figure 5 and Figure 6 for two different distributions of ξ_1 and ξ_2 respectively. At high utilizations, the gain shown in Figure 5 is about a factor of 10 higher than can be seen in Figure 4. Another important observation is that the gain of the SRA algorithm is smaller with ξ_2 when compared to ξ_1. At high utilization, even the energy consumption of ERTH is reduced when compared to SRA. Hence, ERTH is favorable at high utilizations for a task-set containing small number of BE tasks.

The system consumes typical power (idle power) only in one scenario, i.e. when the available interval is not feasible to initiate a sleep state due to either break-even-time limitation or when the scheduler cannot guarantee the real-time constraint. Otherwise the used energy depends on the selected sleep state. Figure 7 compares ERTH with LC-EDF in terms of the normalized energy consumption in the idle interval. The idle energy consumption of ERTH and LC-EDF is normalized to the corresponding idle energy consumption of NS algorithm. The result indicates that LC-EDF performance degrades with an increase in system utilization. LC-EDF selects the single most efficient sleep state among the set of available sleep states based on its maximum-feasible-idle interval. As the system utilization increases, the maximum-feasible-idle interval length shrinks. Consequently, LC-EDF cannot select the more efficient sleep states due to their higher transition delay t_n^e which in turn leads to increased energy consumption. At $U = 1$, LC-EDF behaves similar to a system that is not using sleep states. Opposed to this ERTH can collate the available slack in the system, shows a smooth behavior for all utilizations. The SRA algorithm behaves similar to ERTH from $U = 0.2$ to $U = 0.9$ and afterwards it follows LC-EDF.

The disadvantage of the slack management algorithm proposed in this paper is the poor slack distribution. The improved slack management approach used in the SRA algorithm is also integrated with the ERTH algorithm for the fair comparison. The gain of the ERTH algorithm with improved slack management over ERTH with simplistic slack management approach proposed in this work in our current experimental set-up is negligible. The reason behind such a behavior is the fact that better slack distribution plays an important role for DVFS based algorithms, where the slack distribution among different tasks is important. However, when it comes to race-to-halt algorithms, slack accumulation is important rather than better slack distribution.

IRTH and LWRTH target the pessimism introduced in the ERTH at the cost of extra overhead. In order to quantify their effectiveness, we have analyzed the overall-gain of IRTH and LWRTH over ERTH. The corresponding results are illustrated in Figure 8 with a ξ_1 and $\Gamma_{0.1}$. Four important observations are evident from the results. Firstly, the gain decreases with an increase in task-set size. IRTH and LWRTH reduce the pessimism by utilising their past information

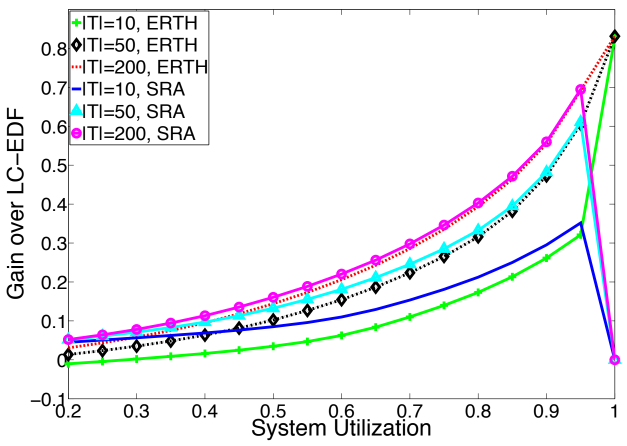


Figure 5. Gain of ERTH and SRA over LC-EDF in idle interval (ξ_1)

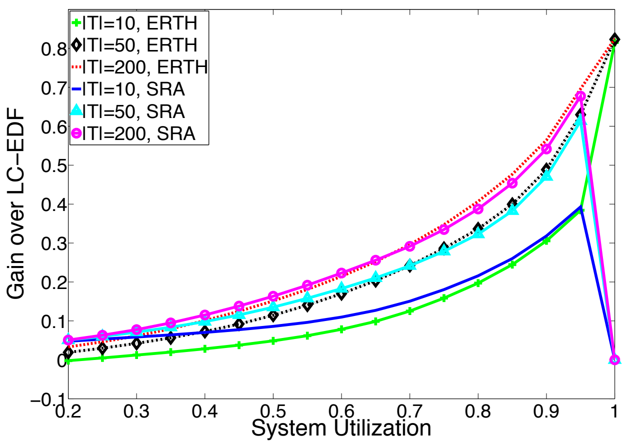


Figure 6. Gain of ERTH and SRA over LC-EDF in idle interval (ξ_2)

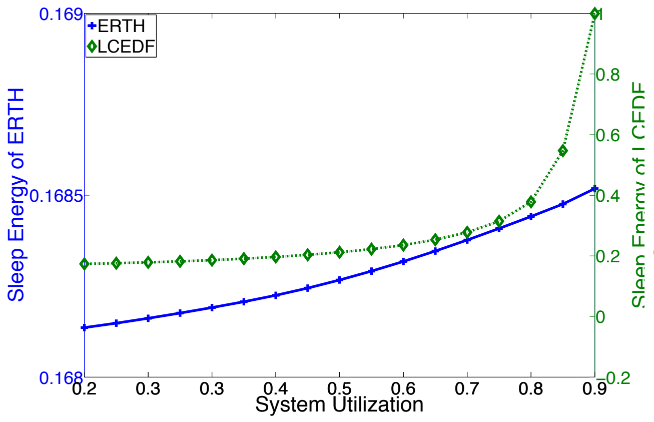


Figure 7. Normalized Sleep Energy Consumption (ξ_1 and $|\mathrm{T}| = 200$)

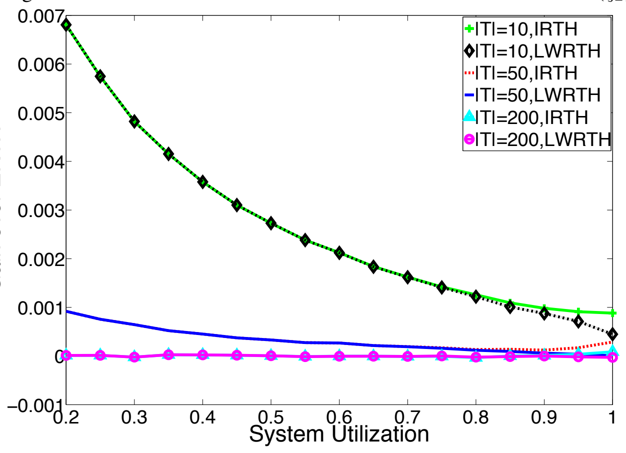


Figure 8. Overall-Gain of IRTH and LWRTH over ERTH for Different Task-Set Sizes (ξ_1)

to predict the future. The goal of these algorithms is to extend the sleep duration. Intuitively, one can argue with an increase in task-set size, future release information predicted is less helpful to extend the sleep interval. For a task-set size of 200, both IRTH and LWRTH behave very similar to ERTH. Secondly, with an increase in system utilization, especially for a task-set sizes of 10 and 50, the overall gain decreases. The increase of system utilization decreases the idle interval in the schedule and pushed the releases closer to each other. Hence, future release information becomes less important. Third observation is the difference between IRTH and LWRTH. IRTH exploits the execution slack explicitly in the system thus behaves superior to LWRTH at higher utilization. Finally, the gains are moderate over ERTH but worthwhile in mobile systems.

We have computed the average sleep-interval for all the algorithms. To determine such value we have divided the total sleep duration over the number of sleep transitions. Figure 9 and Figure 10 present the average sleep-interval against utilization with a distribution of ξ_1 for task-set sizes of 10 and 50 respectively. Similarly, Figure 11 and Figure 12 demonstrate the results with ξ_2 for task-set sizes of 10 and 50 respectively. All the values in these results are normalized to the maximum average sleep-interval of the SRA algorithm in the corresponding task-set size. The results presented in these graphs are consistent with the previously explained results in Figure 4, Figure 5, Figure 6 and Figure 8. The reasons described for different behaviors of all algorithms in terms of energy consumption also applies on the average sleep-intervals. The SRA algorithm has the highest average sleep-interval compared to other algorithms for a distribution of ξ_1 except at $U = 1$. With the same setting SRA has the maximum gain in energy consumption over LC-EDF as shown in Figure 4. LWRTH and IRTH behave the same at low utilizations but get diverted at high utilizations for both distributions (ξ_1 and ξ_2) (also see Figure 8). The average sleep interval of ERTH is around 10% lower when compared to IRTH and LWRTH for small task-set sizes for both distributions. However, for a large task-set size ($|\mathrm{T}| = 50$) ERTH, LWRTH and IRTH behaves same for ξ_1 and ξ_2 except at very high utilization, where IRTH has larger average sleep-interval due to less pessimism in analysis. The SRA algorithm has lower average sleep interval when compared to IRTH and LWRTH with a distribution of ξ_2 for both task-set sizes ($|\mathrm{T}| = 10$ and $|\mathrm{T}| = 50$) after $U \geq 0.7$. ERTH behaves better against SRA for large task-set size after $U \geq 0.7$ for ξ_2 (with same setting gain in energy consumption of

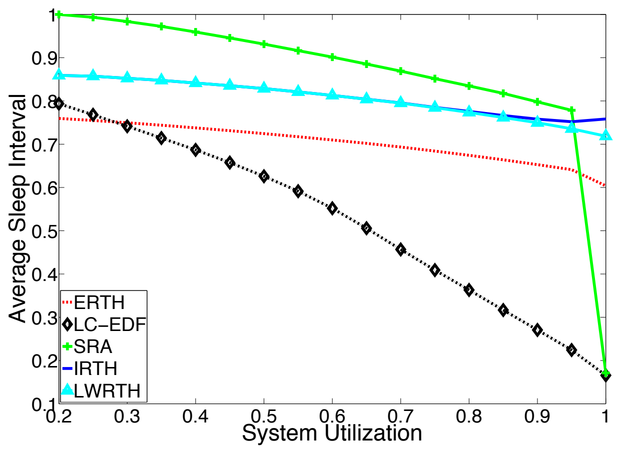


Figure 9. Normalized Average Sleep Interval ($|\text{T}| = 10$ and ξ_1)

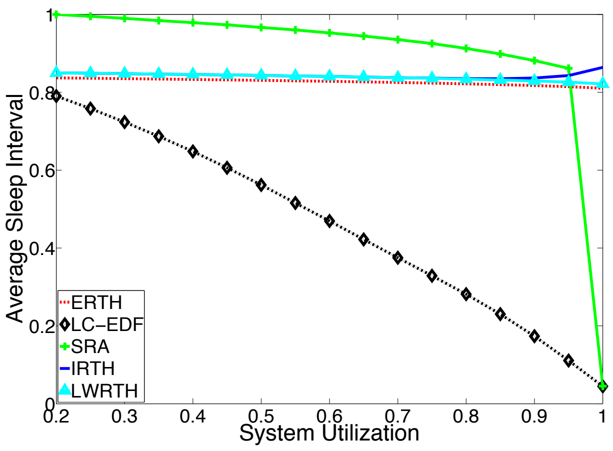


Figure 10. Normalized Average Sleep Interval ($|\text{T}| = 50$ and ξ_1)

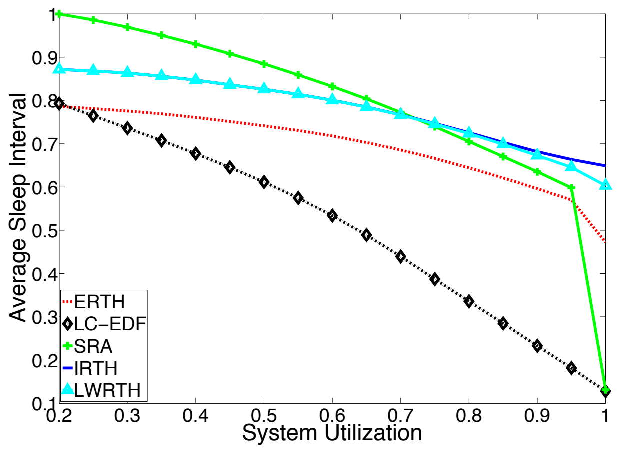


Figure 11. Normalized Average Sleep Interval ($|\text{T}| = 10$ and ξ_2)

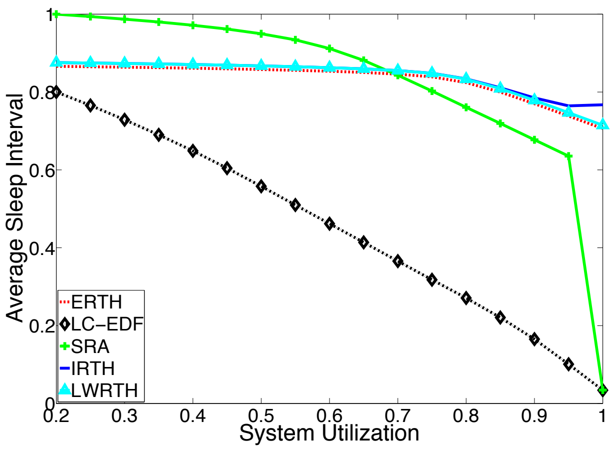


Figure 12. Normalized Average Sleep Interval ($|\text{T}| = 50$ and ξ_2)

ERTH is also higher when compare to SRA in ξ_2, see Figure 6). These results demonstrate that SRA does not behaves better than our algorithms with large number of RT tasks (i.e. for a distribution of ξ_2) at high utilizations. This is the same conclusion derived from the energy consumption of these algorithms.

To analyze the effect of different types of hardware platforms, the effect of a high sleep threshold Ψ that indicates the scaled value of t_n^e obtained by altering the power model parameters is studied for ERTH, IRTH, LWRTH, SRA and LC-EDF for two different distributions of ξ_1 and ξ_2. We analyzed the different scaling factors of the sleep threshold given in Table I. The scaling of Ψ corresponds to a scaling of all t_n^e. This is achieved by modifying the power model parameters such as sleep transitions overheads etc. Figure 13 presents the energy consumption of ERTH for different values of Ψ with $|T| = 50$ and ξ_1. Naturally, an increase in t_n^e is also reflected in higher overall energy consumption as depicted in Figure 13. IRTH and LWRTH have the similar results for the different values of Ψ.

LC-EDF suffers from a high dependence on different available sleep states. The effect of different sleep threshold values on LC-EDF is shown in Figure 14 with $|\text{T}| = 50$ and ξ_1. LC-EDF uses a single sleep state for each utilization and Ψ pair. Similar to ERTH the energy consumption of the system in LC-EDF also increase with an increase in the value of t_n^e. Abrupt variations on the same line of any sleep threshold refer to a switch to a different sleep state. An interesting observation in this graph is the drop of energy consumption for Ψ_{20} at $U = 0.5$ when compared to the energy consumption at $U = 0.45$, which is explained as follows with the help of Figure 15. LC-EDF can compute a bound on the maximum-feasible-idle interval in the schedule and states all the idle-intervals will be longer than this bound. An opportunistic approach of LC-EDF selects the most efficient sleep state considering a bound on the maximum-feasible-idle interval. Usually the actual-idle-intervals are longer than this bound. However, with an increase in system utilization, difference of actual-idle-intervals and t_n^e (as shown in Figure 15) becomes smaller. Other sleep states with low transition overhead has greater margin to save more energy when compared to the more efficient sleep state with higher transition delay. Therefore, at $U = 0.5$ when system switches to another less efficient sleep state it saves more compared to the sleep state selected at $U = 0.45$ for Ψ_{20}.

The effect of different sleep threshold on SRA is presented in Figure 16 for a distribution of ξ_1 and a task-set size

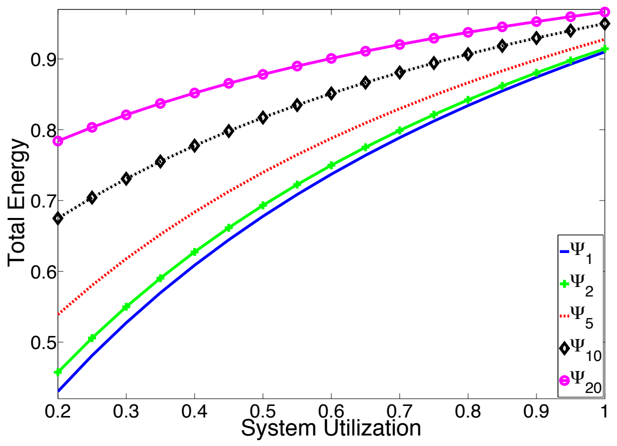


Figure 13. Effect of Sleep Threshold Change on Total Energy Consumption of ERTH ($|\mathrm{T}| = 50$ and ξ_1)

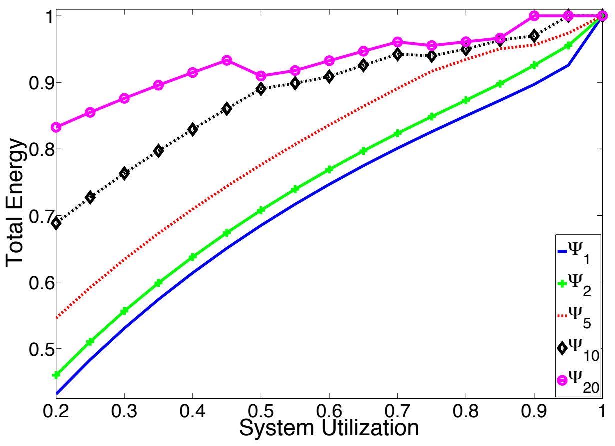
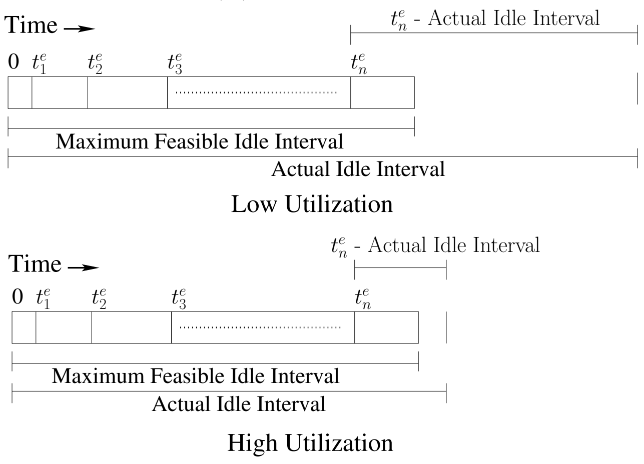


Figure 14. Effect of Sleep Threshold Change on Total Energy Consumption of LC-EDF ($|\mathrm{T}| = 50$ and ξ_1)



Time
t_n^e - Actual Idle Interval
0 t_1^e t_2^e t_3^e t_n^e
Maximum Feasible Idle Interval
Actual Idle Interval
Low Utilization
Time
t_n^e - Actual Idle Interval
0 t_1^e t_2^e t_3^e t_n^e
Maximum Feasible Idle Interval
Actual Idle Interval
High Utilization

Figure 15. Energy Drop on Same Threshold of LC-EDF Algorithm

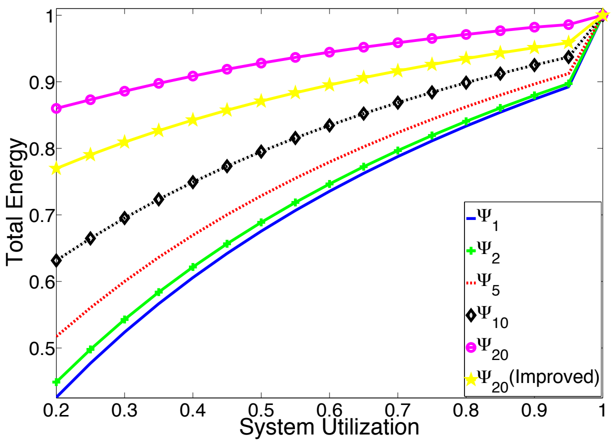


Figure 16. Effect of Sleep Threshold Change on Total Energy Consumption of SRA ($|\mathrm{T}| = 50$ and ξ_1)

of 50. It behaves similar to ERTH for all thresholds except Ψ_{20}. The energy consumption of the SRA algorithm scales up for Ψ_{20} when compared to ERTH. To ensure the scehdulability, the SRA algorithm selects its most efficient sleep state offline considering the minimum idle interval Z_{min}. However, the offline selected sleep state might not be a good choice as explained earlier for LC-EDF algorithm with the help of Figure 15. We provide one extension to the SRA algorithm to enhance its performance by selecting the appropriate sleep state online based on the predicted idle interval. When the system is in an idle mode, the next sleep duration is determined to greater than or equal to an interval $z_{sleep} = max(Z_1, R_1^F)$, where Z_1 is the maximum procrastination interval allowed on the arrival of highest priority task and R_1^F is the execution slack available to the highest priority task. The sleep state is selected for the sleep interval Z_{sleep} online. The simulations results for Ψ_{20} are shown in Figure 16 under a legend $\Psi_{20}(Improved)$. It clearly shows an effectiveness of the proposed modification. All the experiments presented in this paper are repeated for the all settings with this modification. The results remains same for all cases except for the very high threshold of Ψ_{20}. Therefore, this modification is useful for the hardware platforms having sleep states with high sleep transition overheads. Though it increases the online overhead of sleep state selection but cannot perform worst in terms of energy saving when compared to original SRA algorithm.

The effect of Ψ is also analyzed for different task-set sizes. The energy consumption of ERTH is presented in Figure 17 for different task-set sizes with Ψ_{10} and ξ_1. The high sleep threshold managed to created a minute difference between the energy consumption of $|T| = 10$ when compared to other task-set sizes at low utilizations. At such a low utilization tasks in a small task-set size are widely spread out and provide an extra opportunity to the system to use most efficient sleep states in conjugation with Principle 2. This experiment with the same values is repeated for LC-EDF as shown in Figure 18. The spread along the vertical axis among the different task-set sizes is higher when compared to ERTH, which affirms the strong dependency of LC-EDF on the task-set size as explained in the beginning of section IX-B1. The reason for the bumpy effect on the line of $|T| = 200$ is the same as already been explained earlier with Figure 14. The same experiment is also done for IRTH and LWRTH. The graphs has the same shape as ERTH with a slight larger

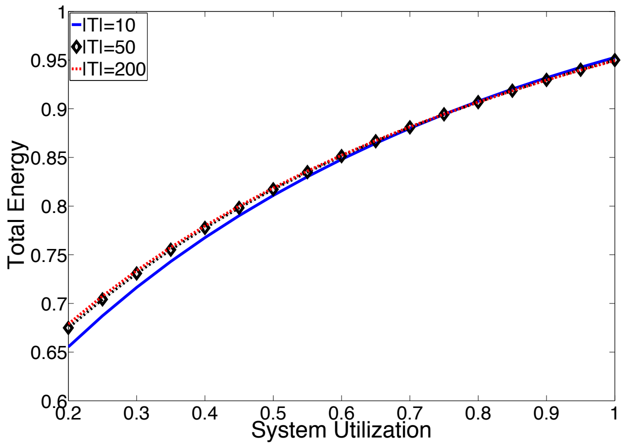


Figure 17. Effect of Sleep Threshold Ψ_{10} on ERTH with Different Task-set Sizes under ξ_1

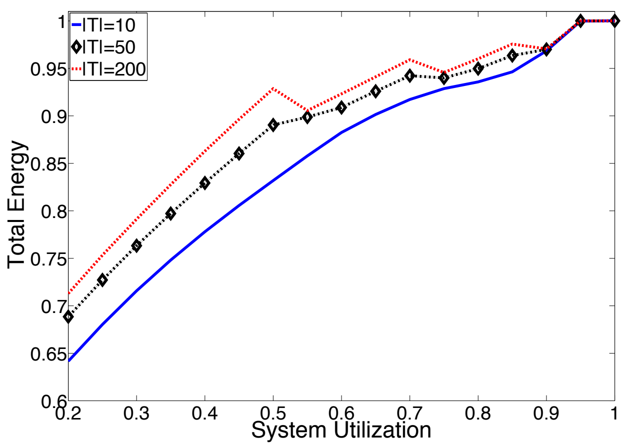


Figure 18. Effect of Sleep Threshold Ψ_{10} on LC-EDF Algorithm with Different Task-set Sizes under ξ_1

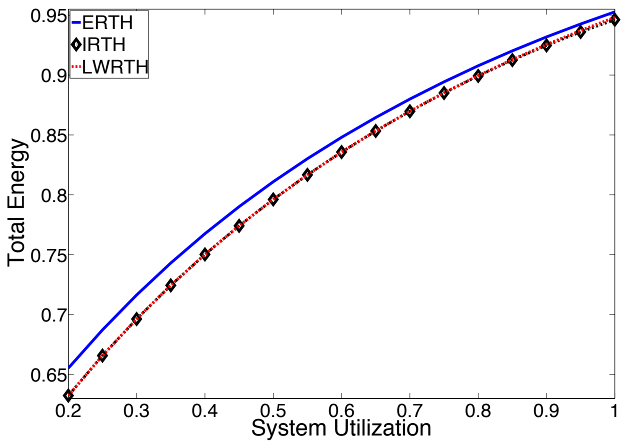


Figure 19. Total Energy Consumption of ERTH, IRTH and LWRTH at Ψ_{10} with $|T| = 10$ and ξ_1

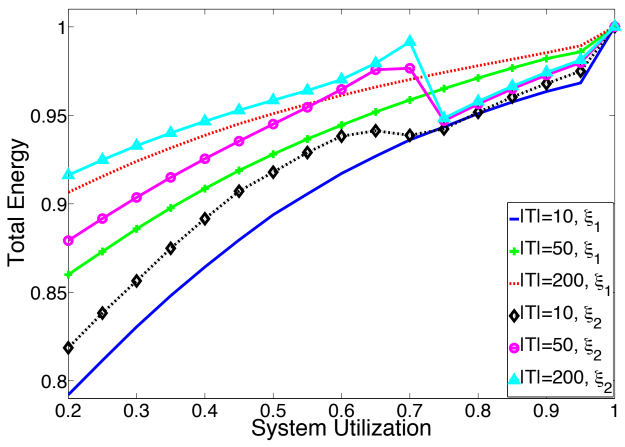


Figure 20. Effect of Distributions (ξ_1, ξ_2) on High Sleep Threshold with SRA (Ψ_{20})

gap between $|T| = 10$ and other task-set sizes. It demonstrates that high sleep threshold slightly favours small task-set size at low utilization.

The comparison of the energy consumption of ERTH, IRTH and LWRTH is illustrated in Figure 19 with $|T| = 10$, Ψ_{10} and ξ_1. The increase in t_n^e enhance the potential of IRTH and LWRTH to save more energy compared to ERTH. The future release information is useful at high sleep threshold values and helps to pick the more efficient sleep states. The curve of LWRTH tends to rise at $U = 1$ when compared to IRTH but the difference is very small and negligible.

Figure 20 shows the effect of high threshold Ψ_{20} on different task-set sizes and two different distributions with SRA. First observation is the difference of energy consumption between different task-set sizes. Secondly, the dropped down of the energy consumption for a distribution of ξ_2 is due to the change of sleep states. The reason for such behavior is already explained in conjunction with LC-EDF's similar behavior with the help of Figure 15. The performance of SRA algorithm suffers with a decrease in the number of BE tasks as the long period tasks allows longer procrastination interval.

Similarly, the effect of Ψ is also analyzed for a distribution ξ_2 with all task-set sizes on all algorithms (ERTH, IRTH, LWRTH, SRA and LC-EDF). The only difference it makes is the increase in the total energy consumption. This is motivated in the reduced share of BE tasks. It reduces the opportunity in ERTH, IRTH and LWRTH to use the Principle 2, and thus results in a extra energy consumption. SRA and LC-EDF algorithms (as mentioned earlier) depend on the periods of the tasks. Therefore, fewer tasks with longer periods decrease the opportunity to save energy consumption, hence ξ_2 results in more energy consumed when compared to the ξ_1.

2) Scenario 2 $(RT \Rightarrow (A_i = C_i), BE \Rightarrow (A_i \leq C_i))$*:* In scenario 2, BE tasks are allowed to occasionally require more than their allocated budget A_i. All algorithms i.e. ERTH, IRTH, LWRTH, SRA and LC-EDF have been extended and allowed to borrow from the budget of future job releases of the same task. While it was of little consequence in scenario 1, it has to be noted that in scenario 2 ERTH and IRTH do not allocate execution slack to BE tasks for two major reasons. Firstly, it reduces the priority of the execution slack by extending its deadline. By virtue of borrowing,

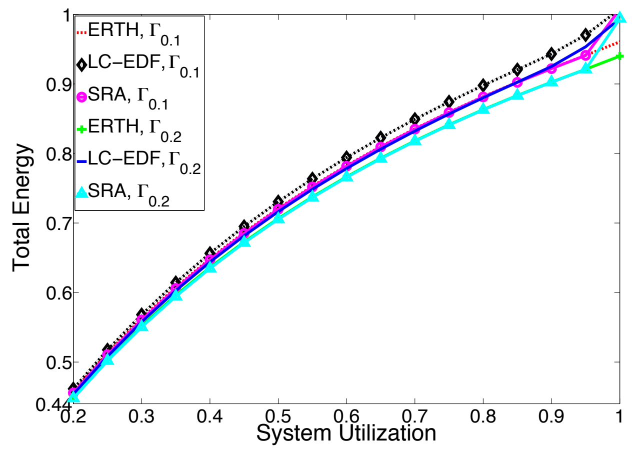


Figure 21. Normalized Total Energy Consumption with $|\mathrm{T}| = 200$ and ξ_1

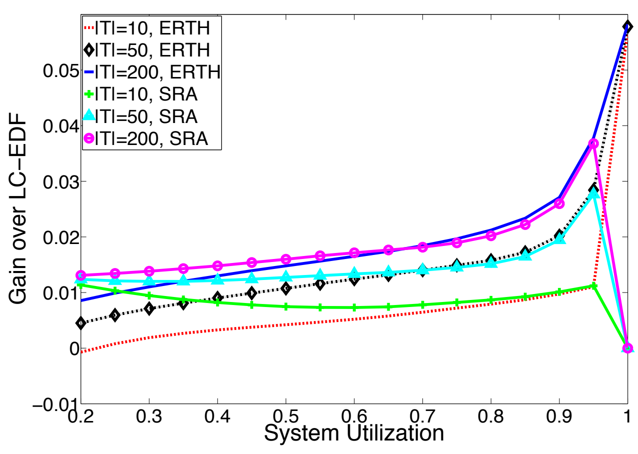


Figure 22. Overall-Gain of ERTH and SRA over LC-EDF with ξ_2 and $\Gamma_{0.1}$

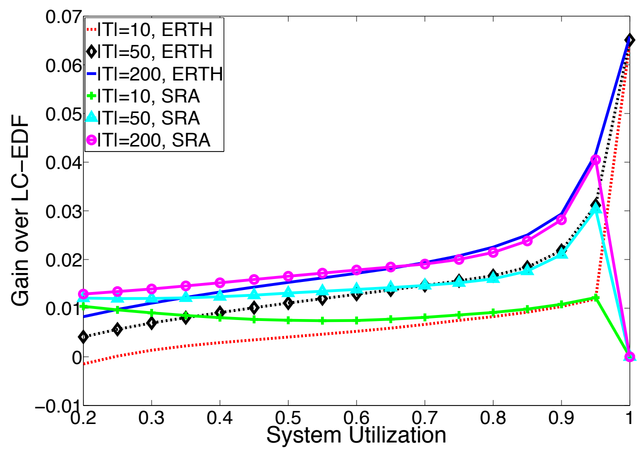


Figure 23. Overall-Gain of ERTH and SRA over LC-EDF with ξ_2 and $\Gamma_{0.2}$

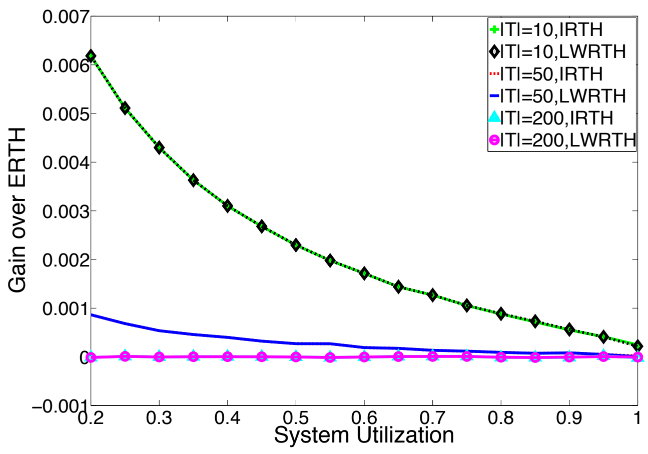


Figure 24. Overall-Gain of IRTH and LWRTH over ERTH (ξ_1 and $\Gamma_{0.1}$)

BE jobs usually have longer deadlines because borrowing mechanism extends the deadline of the overrun job by one period from its current deadline. Therefore, if slack is allocated to such a job, the deadline of the slack is also extended with it and hence, its priority reduces. As a consequence of this most RT tasks will not be able to initiate a sleep state with the available slack in the system and results in a poor slack distribution. Secondly, the complexity of Equation 20 or Equation 24 (depending which algorithm we are using, ERTH or IRTH) also increases. In both algorithms,if a system allocates an execution slack to BE tasks, it would need to compute for increased number of jobs, as the deadline of the slack is usually longer compared to a case when system does not allocate execution slack to BE tasks.

We have analyzed the total energy consumption of ERTH, SRA and LC-EDF in this scenario for two different sporadic delay limits ($\Gamma_{0.1}, \Gamma_{0.2}$) and two different distributions (ξ_1,ξ_2) with $|\mathrm{T}| = 200$. Figure 21 demonstrates the effect of a variation in the sporadic delay limit. The distribution for this experiment is fixed to ξ_1. For the sake of clear representation we chose to normalize all the values of Figure 21 to the corresponding results of NS with a distribution of $\Gamma_{0.1}$. $\Gamma_{0.1}$ and $\Gamma_{0.2}$ define an interval of 10% and 20% of T_i respectively for the sporadic delay to maneuver for a task T_i. The expansion of this interval means we are injecting more sporadic slack in the system when compared to the nominal utilization. The sporadic slack is dealt implicitly in our algorithms. Therefore, energy consumption of the system is less in $\Gamma_{0.2}$ when compared to $\Gamma_{0.1}$ as shown in Figure 21. Similarly, SRA and LC-EDF also have more room to initiate a sleep state as well. However, at higher utilization; extra sporadic slack does not help to save energy in LC-EDF or SRA. These algorithms (LC-EDF and SRA) calculate their maximum-feasible-idle interval based on the worst-case scenario i.e. each job of a task will be released as soon as possible with a difference of minimum inter-arrival. Therefore, at high utilization, the feasible-sleep interval is usually short and they cannot utilize the more efficient sleep states effectively. As a general rule ERTH performs superior to LC-EDF, especially at higher utilizations the difference is prominent. However, it performs comparable to SRA but consumes less energy at higher utilizations. The same experiment is repeated with ξ_2. The energy consumption of all algorithms decreases, the reason of which is explained in conjunction with the next experiment.

The energy consumption of two distributions (ξ_1,ξ_2) is studied with a fixed task-set size of $|\mathrm{T}| = 200$ and $\Gamma_{0.1}$. The resulting figure (not shown here) has a similar shape when compared to Figure 21 but the difference between ξ_1 and ξ_2 is slightly more pronounced. The energy consumption of SRA, LC-EDF and ERTH is reduced for ξ_2 when compared to ξ_1. The percentage of the BE tasks in ξ_2 is reduced to 40% and results in less borrowing. Therefore, the system has less energy consumption with ξ_2 when compared to ξ_1. Nevertheless, ERTH outperforms LC-EDF and comparable to SRA in both distributions (ξ_1, ξ_2), even with the borrowing mechanism integrated. The energy consumption of all algorithms decreases, when the same experiment is done with $\Gamma_{0.2}$ due to extra sporadic slack in the system. Moreover, we also observed that the energy consumption of IRTH and LWRTH is virtually similar to ERTH for the above mentioned two experiments. The borrowing effect dominates the total energy consumption and provides less room to maneuver for energy saving.

The overall energy consumption gain of ERTH and SRA over LC-EDF is analyzed for scenario 2 for three task-set sizes ($|\mathrm{T}| \in \{10, 50, 200\}$) with ξ_2 in Figure 22 and Figure 23 considering two different sporadic delay limits $\Gamma_{0.1}$ and $\Gamma_{0.2}$ respectively. Although sporadic slack is managed implicitly in all algorithms, ERTH outperforms LC-EDF, especially at higher utilization for large task-set sizes. The SRA algorithm performs better when compared to LC-EDF in all cases. For large task-set sizes, ERTH performs superior when compared to SRA at high utilizations and SRA performs better at low utilizations. The slight increase in gain of ERTH with $\Gamma_{0.2}$ over $\Gamma_{0.1}$ indicates an efficient implicit use of sporadic slack in ERTH. Nevertheless, LC-EDF performs slightly better when compared to ERTH only at $U = 0.2$ for a task-set size of 10 but the difference is negligible. LC-EDF can create large gaps for small task-set sizes at very low utilization and hence gains over ERTH at $U = 0.2$ with negligible margin. Similarly, if we vary the distribution to ξ_1 (not shown here), the results indicate a slight decrease in overall gain. Major difference lies at $U = 1$, where it varies approximately about 1% of overall gain. However, for smaller utilizations the difference is less pronounced. This is a function of the reduced number of BE tasks in ξ_2 and the consequently smaller amount of borrowing in the system.

Figure 22 demonstrates the overall-gain of ERTH and SRA over LC-EDF for the distribution ξ_2 and $\Gamma_{0.1}$ and comparing that to Figure 4 one can notice the reduced gains returned when borrowing. Generally, the gain of scenario 2 compared to scenario 1 is less at higher utilizations, but approximately the same at lower utilizations. The gain rises exponentially in Figure 4, Figure 22 after $U = 0.8$ for large task-set sizes. The overall energy gain of IRTH and LWRTH over ERTH is depicted in Figure 24 for ξ_1 and $\Gamma_{0.1}$. Compared to Figure 8, the overall gain has reduced in scenario 2. Moreover, IRTH and LWRTH behave identical when borrowing is enabled. Main reason is the extra execution requested by the BE task through borrowing i.e. an increase in effective utilization.

The normalized sleep state energy consumption of scenario 2 is similar to scenario 1. Moreover, the higher sleep threshold effect in scenario 2 is also identical to scenario 1 for IRTH, LWRTH, LC-EDF, SRA and ERTH with just one difference, i.e. energy consumption of the system increases in scenario 2. It happens due to an increased in execution-time requirement of the BE tasks that occasionally overrun and borrow from its future releases. To summarize, for different combinations of ξ and Γ, an increase in gain occurs in the following ascending order $(\xi_2, \Gamma_{0.2})$, $(\xi_2, \Gamma_{0.1})$, $(\xi_1, \Gamma_{0.2})$ and $(\xi_1, \Gamma_{0.1})$. This is caused by an increase in sporadic delay limit, sporadic slack of the system also increases, therefore the system saves more energy. Similarly, if we enable borrowing, the system consumes extra energy. Thus with the highest sporadic delay limit and minimum borrowing $(\xi_2, \Gamma_{0.2})$ the energy consumption is least in scenario 2, whilst with least sporadic delay limit and most borrowing $(\xi_1, \Gamma_{0.1})$ energy consumption is maximized.

C. Pre-emptions Related Results

A side effect of the use of the sleep states is a change in the number of pre-emptions. In order to find the sleep state relation with the number of pre-emptions, the pre-emptions for all algorithms (ERTH, IRTH, LWRTH, SRA and LC-EDF) are counted for different parameters. The experimental setup mentioned in Section IX-A and the parameters defined in Table I remain the same except some alterations in best-case execution-time limit C^b and sporadic delay limit Γ. The best-case execution-time limit C^b is varied from 0.25 to 1 with an increment of 0.25 (i.e. $C^b \in \{0.25, 0.5, 0.75, 1\}$). Similarly, the sporadic delay limit Γ is varied from 0 to 0.6 with an increment of 0.2 (i.e. $\Gamma \in \{0, 0.2, 0.4, 0.6\}$). For the representation purposes we have plotted only the two corner values for $\Gamma = (0, 0.6)$ and $C^b = (0.25, 1)$ as the results for the other two values lies in between these two curves and scales linearly. All the values in the following experiments are normalized to the number of pre-emptions with earliest deadline first algorithm (EDF). We have observed that the pre-emption count for LWRTH is virtually identical to IRTH, therefore, for presentation purposes only results of IRTH are shown hereafter.

1) Scenario 1*:* In this scenario, we assume all the tasks have $A_i = C_i$. The effect of best-case execution-time limit variation is shown in Figure 25 with $|\mathrm{T}| = 10$, $\Gamma_{0.2}$ and ξ_1. The results are plotted only for $C^b \in \{0.25, 1\}$, while the other two values of $C^b \in \{0.5, 0.75\}$ lie in between these two curves of the corresponding algorithm and scale linearly. First observation for small task-set size is the positive impact of $C^b = 0.25$ over $C^b = 1$ that holds for all utilizations with LC-EDF and ERTH, and only at high utilizations with SRA and IRTH (at low utilizations the opposite behavior of C^b for SRA and IRTH will be explained later in the discussion). The reason is quite clear, an increase in the value of C^b potentially decreases the range of execution slack that a task can provide online. Thus $C^b = 1$ means no execution slack

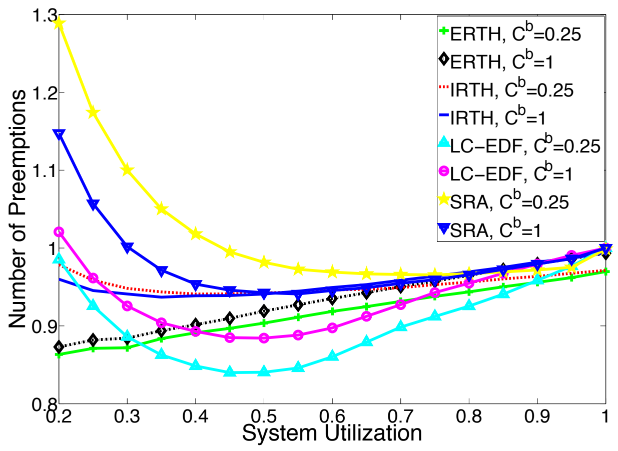


Figure 25. Effect of Variation in Best-Case Execution-Time Limit for $|\mathrm{T}| = 10$ $(\Gamma_{0.2}, \xi_1)$

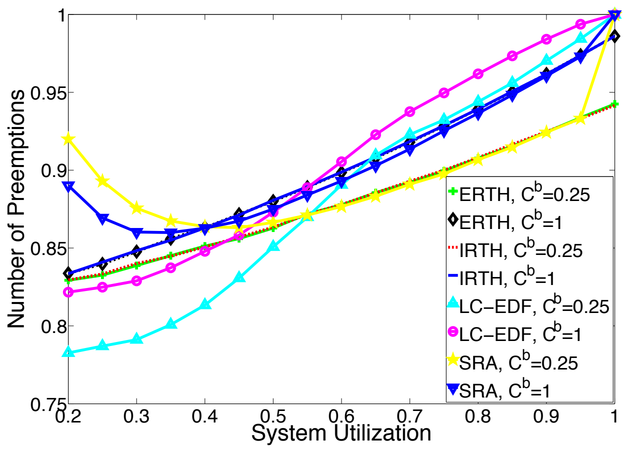


Figure 26. Effect of Variation in Best-Case Execution-Time Limit for $|\mathrm{T}| = 50$ $(\Gamma_{0.2}, \xi_1)$

in the system. Overall all scheduling algorithms showed a positive impact of sleep states on the number of pre-emptions, except for one case in LC-EDF at $U = 0.2$ and for SRA at $U \leq 0.45$. If we inject more execution slack, the system initiates more often a sleep state and hence lowers the number of pre-emptions.

As previously mentioned in Section VIII, a sleep state that delays the execution can increase the pre-emptions by pushing it closer to higher priority tasks. While at the same time, the delayed execution caused by a sleep state can combine the job releases to reduce the pre-emption count. With a small task-set size, jobs releases are anyway dispersed at low utilization. However, as the utilization increases execution increases and jobs execution run into each other and cause a rise in the number of pre-emptions. SRA and LC-EDF initiate a sleep state in idle mode, starts estimating the delay interval on the next job release and extend it as much as possible. This behavior causes widely spread low priority jobs at low utilization to come closer to high priority jobs and hence increase the pre-emption count. Moreover, at low utilization, in EDF the number of pre-emptions are smaller and the use of sleep states cannot help much to reduce them. However, as the utilization increases pre-emption count drops quickly for LC-EDF up to approximately a utilization of 0.5 and for SRA up to $U = 0.65$. These algorithms (SRA and LC-EDF) can collate enough tasks releases to compensate the effect of extra pre-emption due to the delay of execution. Nevertheless, at a further increase in utilization (beyond $U > 0.5$ for LC-EDF and $U > 0.65$ for SRA), the possibility to use sleep intervals also reduces for both LC-EDF and SRA, and consequently their ability to reduce the pre-emptions. Therefore, at $U = 1$, both algorithms cannot afford to initiate a sleep state and hence, the pre-emption count is exactly the same as EDF.

The ERTH algorithm distributes the sleep states uniformly in the schedule and never pushes the sleep interval beyond the static limit t_l, even if there is a possibility to prolong the sleep interval. Not extending the sleep state to its limit pays off at low utilization, as it never delays execution to increase the pre-emption count. However, ERTH ability to introduce sleep in the busy interval helps to save pre-emptions for even higher utilizations. Therefore, in ERTH, we have linear curve from low to high utilization for both C^b. The difference decreases towards high utilization due to a decrease in the execution slack and hence fewer sleep states in the system.

IRTH and SRA show an oddity at low utilisations, as $C^b = 1$ has fewer pre-emptions compared to $C^b = 0.25$. In IRTH algorithm, sleep states are increased by utilizing predicted future release information. Future release information is very useful especially to prolong the sleep interval for small task-set size at low utilizations. It can be easily motivated by the curve of Figure 8 that IRTH save more energy at low utilizations for a task-set size of 10 due to extensively long sleep intervals. Similarly, SRA sleep intervals are even greater than or equal to all the algorithms. As a side effect of long sleep intervals, they assemble a large amount of work for later execution. This delayed execution later on encounters high priority tasks and causes additional pre-emptions. However, if the encountered high priority tasks execute for their C_i, chances are higher that it might accumulate other tasks having priority higher than the backlog and less than the encountered high priority tasks. These intermediate priority tasks will not cause pre-emptions to a backlog. This effect causes the flip of $C^b = 0.25$ over $C^b = 1$ for low utilizations.

Large task-set sizes give a more smoother curve, as shown in Figure 26 for a task-set size of $|\mathrm{T}| = 50$ with a distribution of ξ_1 and sporadic delay limit of $\Gamma_{0.2}$. All the algorithms have fewer number of pre-emptions when compared to EDF and also savings are larger compared to a task-set size of 10. An odd behavior of SRA and LC-EDF is eliminated for small utilizations, as the probability of jobs being widely spread out is lower with the large task-set size. IRTH also behaves identical to ERTH, as future release information is less effective for large task-sets. Moreover, similar to previous case, curve for other two values $C^b \in \{0.5, 0.75\}$ lies in between $C^b = 1$ and $C^b = 0.25$ with no exception for corresponding algorithms.

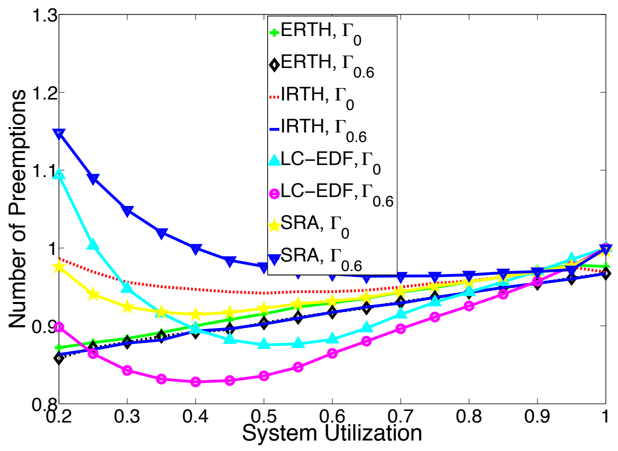


Figure 27. Effect of Variation in Sporadic Delay Limit for a task-set size of $|\mathrm{T}| = 10$ (ξ_1)

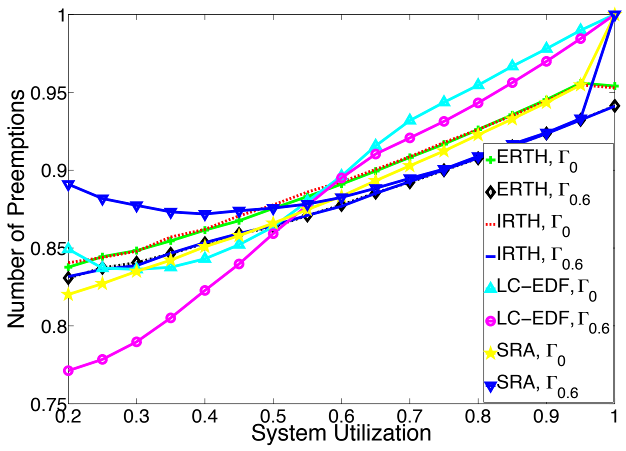


Figure 28. Effect of Variation in Sporadic Delay Limit for a task-set size of $|\mathrm{T}| = 50$ (ξ_1)

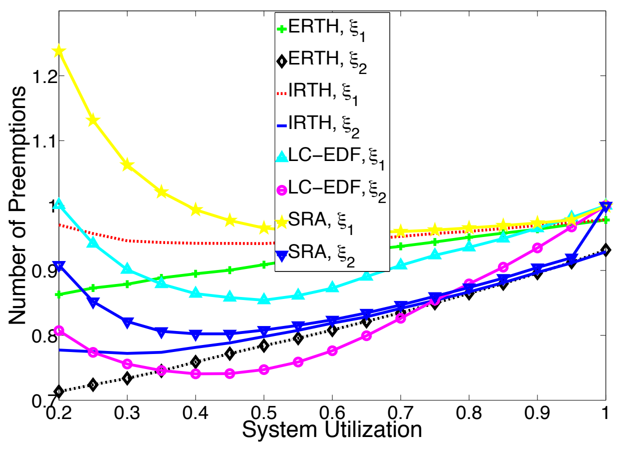


Figure 29. Effect of Variation in Distribution ξ for a task-set size of $|\mathrm{T}| = 10$ ($\Gamma_{0.2}, C^b = 0.5$)

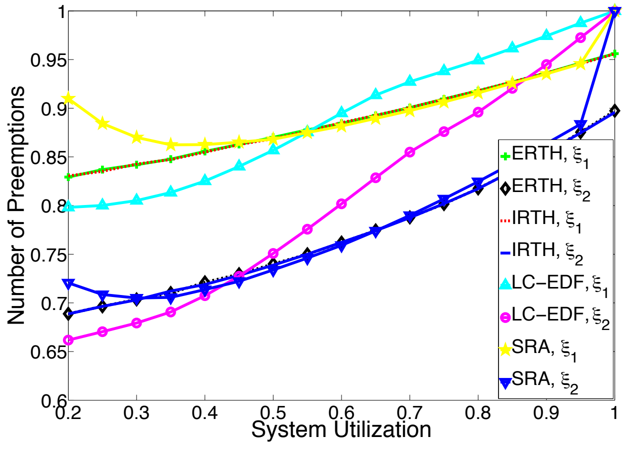


Figure 30. Effect of Variation in Distribution ξ for a task-set size of $|\mathrm{T}| = 50$ ($\Gamma_{0.2}, C^b = 0.5$)

The effect of variation in sporadic delay limit Γ is illustrated in Figure 27 for a task-set $|\mathrm{T}| = 10$ and a distribution of ξ_1. All algorithms consume sporadic slack implicitly. An increase in the sporadic delay limit causes an increase in sporadic slack and that can prolong and/or increase the number of sleep transitions. Similar to the execution slack, sporadic slack also helps to decrease the number of pre-emptions for all algorithms except SRA. ERTH and LC-EDF behave similar to the Figure 25, with a slight variation in the beginning and towards the end of utilizations. In IRTH, both sporadic delay limits ($\Gamma_0, \Gamma_{0.6}$) have access to same amount of future release information. Therefore, they also follow the same trend of saving on the number of pre-emptions with extra sporadic slack. However, the SRA algorithm that has the longest sleep intervals of all algorithms increases the number of pre-emptions when extra sporadic slack is available. This is motivated by the fact that widely spread out jobs in the EDF schedule are unlikely to preempt each other but SRA brings these jobs close together to the high priority jobs to such a degree that they result in an increased number of pre-emptions at low utilization. The other two sporadic delay limit ($\Gamma_{0.2}, \Gamma_{0.4}$ are bounded by $\Gamma_0, \Gamma_{0.6}$. A large task-set $|\mathrm{T}| = 50$, brings IRTH curves close to ERTH (Figure 28) because future release information becomes less important. Moreover, the pre-emption avoiding effect of LC-EDF at low and medium utilisations is reduced for larger task set sizes, when comparing to smaller task-sets, due to the higher probability of tasks cutting idle intervals short, as illustrated in Figure 28. Moreover, the SRA algorithm cause more pre-emptions with an increase in sporadic slack for low utilizations. However, for large utilizations it saves more pre-emptions with additional sporadic slack similar to other algorithms. In general, whenever there is a possibility to increase the length of sleep state (either through execution slack or sporadic slack) at low utilizations, SRA increase the number of pre-emptions. The effect of variation in the distribution ξ is demonstrated in Figure 29 for $|\mathrm{T}| = 10, \Gamma_{0.2}$ and $C^b = 0.5$. In general for all the algorithms, distribution ξ_2 saves more pre-emptions compared to ξ_1. BE tasks are more vulnerable to pre-emptions as they have longer periods along with their execution. Therefore, ξ_1 having more BE tasks results in more pre-emptions, when compared to ξ_2. The same observation holds for the large task-set size as shown in Figure 30.

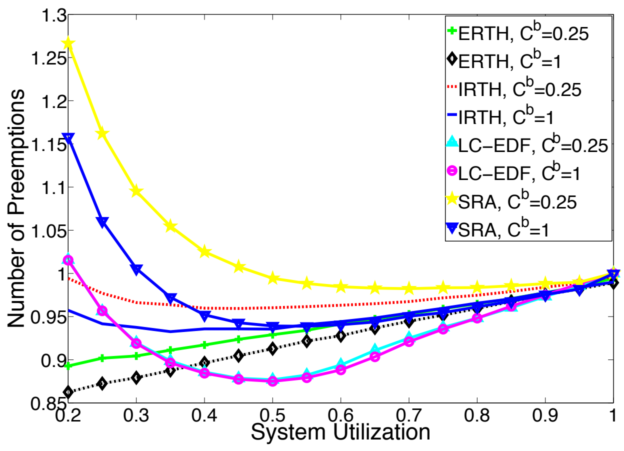


Figure 31. Effect of Variation in Best-Case Execution-Time Limit for a task-set size of $|\mathrm{T}| = 10$ $(\Gamma_{0.2}, \xi_1)$

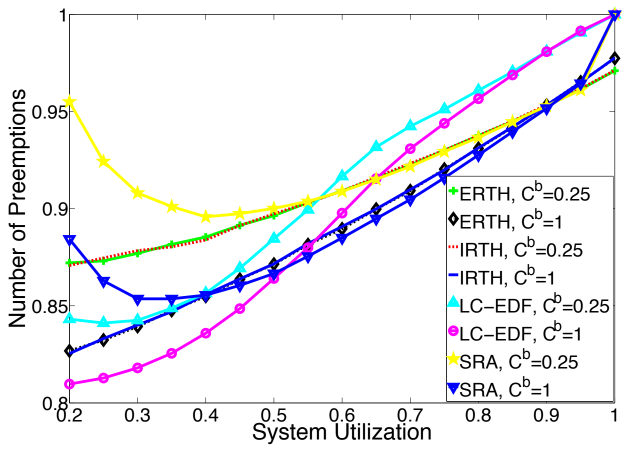


Figure 32. Effect of Variation in Best-Case Execution-Time Limit for a task-set size of $|\mathrm{T}| = 50$ $(\Gamma_{0.2}, \xi_1)$

2) Scenario 2: In this scenario, BE jobs occasionally require more than their respective budget and borrow from their future job releases. Figure 31 depicts the effect of variation in the best-case execution time limit C^b for a $|\mathrm{T}| = 10$, $\Gamma_{0.2}$ and ξ_1. One of the interesting observation that holds for all algorithms in general is that now $C^b = 1$ offers fewer pre-emptions when compared to $C^b = 0.25$. Because of the borrowing, BE tasks add a great deal of backlog in addition to a backlog assembled due to sleep transitions. Therefore, it increases the probability to encounter higher priority tasks. Similar to the case explained for IRTH ($U \leq 0.5$) in Figure 25, if the encountered higher priority tasks execute for their C_i, chances are higher that they will collect some of the tasks having priority in between backlog and the higher priority executing jobs. Thus $C^b = 1$ offers fewer pre-emptions compared to $C^b = 0.25$. Similar behavior is observed for a large task-set size of 50 as shown in Figure 32. Only exception is at $U \geq 0.9$ for particularly ERTH and IRTH. At such a high utilization, sleep states save more pre-emptions when compared to an increment in pre-emptions due to its backlog.

The variation in sporadic delay limit Γ is also observed for all task-set sizes. The results show an increase for low utilizations when compared to a system without borrowing. Moreover, SRA with borrowing in the system saves more pre-emptions with an increase in the amount of sporadic slack. Thus, the number of pre-emptions is higher for $\Gamma_{0.6}$ when compared to Γ_0. The variation in the distribution of task-set ξ also increase the number of pre-emption when we allow the borrowing in the system. BE tasks that overrun demand extra execution and hence more pre-emptions compared to the normal system without borrowing. Thus we have observed, when it comes to number of pre-emptions, ERTH performs superior to IRTH, LWRTH and SRA for small task-set sizes. Nevertheless, it equally performs comparable to IRTH, SRA and LWRTH if not better for large task-set sizes. The SRA algorithm that performs better in terms of energy consumption has the highest number of pre-emptions for at low utilizations and sometimes it is even more than EDF scheduler.

The overhead associated to the number of pre-emptions saved through the use of sleep states can help to reduce the worst-case execution time of the tasks. This effect further extends the slack in the system and consequently provide an extra opportunity to save more energy in the system or increase the system utilization.

X. Conclusions and Future Directions

This research effort presents the energy efficient algorithms based on the race-to-halt mechanism for dynamic priority scheduling while avoiding external specialized hardware commonly used to implement state-of-the-art strategies. The online complexity of the algorithms is reduced when compared to the related work. Our algorithms make explicit use of the execution slack as well as covering static and sporadic slack implicitly through an efficient slack management scheme at the cost of negligibly small overhead. Furthermore, our analysis shows that as a side effect of using sleep states, we can also save on the number of pre-emption on average. Therefore, the positive impact of race-to-halt on the pre-emption count helps to achieve the goal of minimal energy consumption and improves on required reservations. The detailed simulation results elaborate the pros and cons of the proposed algorithms over each other. We conclude that the best state-of-the-art algorithm saves marginally on the energy consumption when compared to our proposed algorithms. However, the most state-of-the-art algorithms use external hardware to implement their power management algorithms which comes with overhead in energy not included in our evaluation, so when considered the energy consumption of the state-of-the-art approaches may even be worse then for our algorithms. Moreover, our algorithms avoid more pre-emptions in general when compared to the most efficient energy management algorithm in the state-of-the-art (SRA).

In the future, we intent to extend this approach for multicores and target the multi-dimensional issues such as energy and thermal constraints simultaneously. Furthermore, the implication of this algorithm on static priority algorithms, self suspending tasks and shared resources is deemed for future work. A task-set containing both CPU and memory intensive workloads also provide an additional opportunity to integrate DVFS with this approach to minimize the overall energy consumption. Device power management in combination with CPU power management helps to globally optimize the energy efficiency of the system.

References

[1] ITRS, "International technology roadmap for semiconductors, 2011 edition, design," 2011.

[2] ITRS, "International technology roadmap for semiconductors, 2010 update, overview," 2010.

[3] E. Le Sueur and G. Heiser, "Slow down or sleep, that is the question," in *Proceedings of the 2011 USENIX Annual Technical Conference*, (Portland, OR, USA), Jun 2011.

[4] J. Donald and M. Martonosi, "Techniques for multicore thermal management: Classification and new exploration," in *Proceedings of the 33rd annual international symposium on Computer Architecture*, ISCA '06, pp. 78–88, 2006.

[5] E. Le Sueur and G. Heiser, "Dynamic voltage and frequency scaling: The laws of diminishing returns," in *Proceedings of the 2010 Workshop on Power Aware Computing and Systems*, Oct. 2010.

[6] ARM Ltd, *Cortex^{TM}-A Series*.

[7] M. A. Awan and S. M. Petters, "Enhanced race-to-halt: A leakage-aware energy management approach for dynamic priority systems," in *Proceedings of the 23rd Euromicro Conference on Real-Time Systems*, pp. 92–101, 2011.

[8] Y.-H. Lee, K. Reddy, and C. Krishna, "Scheduling techniques for reducing leakage power in hard real-time systems," in *Proceedings of the 15th Euromicro Conference on Real-Time Systems*, Jul. 2003.

[9] P. Baptiste, "Scheduling unit tasks to minimize the number of idle periods: a polynomial time algorithm for offline dynamic power management," in *Proceedings of the 17th ACM-SIAM Symposium on Discrete Algorithms*, (Miami, Florida), pp. 364–367, ACM, Jan. 2006.

[10] S. Irani, S. Shukla, and R. Gupta, "Algorithms for power savings," *ACM Transactions on Algorithms*, vol. 3, no. 4, p. 41, 2007.

[11] L. Niu and G. Quan, "Reducing both dynamic and leakage energy consumption for hard real-time systems," in *Proceedings of the International Conference on Compilers, Architecture and Synthesis for Embedded Systems*, (Washington DC, USA), pp. 140–148, ACM, 2004.

[12] R. Jejurikar, C. Pereira, and R. Gupta, "Leakage aware dynamic voltage scaling for real-time embedded systems," in *Proceedings of the 41st Design Automation Conference*, (San Diego), 2004.

[13] R. Jejurikar and R. Gupta, "Procrastination scheduling in fixed priority real-time systems," in *Proceedings of the Conference on Language, Compiler and Tool Support for Embedded Systems'04*, (Washington DC, USA), pp. 57–66, 2004.

[14] J.-J. Chen and T.-W. Kuo, "Procrastination for leakage-aware rate-monotonic scheduling on a dynamic voltage scaling processor," *SIGPLAN Notices*, vol. 41, pp. 153–162, June 2006.

[15] R. Jejurikar and R. Gupta, "Dynamic slack reclamation with procrastination scheduling in real-time embedded systems," in *Proceedings of the 42nd Design Automation Conference*, (Anaheim), pp. 111–116, 2005.

[16] J.-J. Chen and T.-W. Kuo, "Procrastination determination for periodic real-time tasks in leakage-aware dynamic voltage scaling systems.," in *Proceedings of the International Conference on Computer Aided Design*, pp. 289 –294, Nov. 2007.

[17] J.-J. Chen and L. Thiele, "Expected system energy consumption minimization in leakage-aware dvs systems," in *Proceedings of the International Symposium on Low Power Electronics and Design*, (India), pp. 315–320, ACM, 2008.

[18] L. Thiele, S. Chakraborty, and M. Naedele, "Real-time calculus for scheduling hard real-time systems," in *Proceedings of the 2000 IEEE International Symposium on Circuits and Systems, ISCAS 2000 Geneva*, vol. 4, pp. 101 –104 vol.4, 2000.

[19] L. Thiele, E. Wandeler, and N. Stoimenov, "Real-time interfaces for composing real-time systems," in *Proceedings of the 6th International Conference on Embedded Software*, EMSOFT '06, (New York, NY, USA), pp. 34–43, ACM, 2006.

[20] K. Huang, L. Santinelli, J.-J. Chen, L. Thiele, and G. C. Buttazzo, "Adaptive dynamic power management for hard real-time systems," in *Proceedings of the 30th IEEE Real-Time Systems Symposium*, (Washington, DC, USA), pp. 23–32, IEEE Computer Society, 2009.

[21] K. Huang, L. Santinelli, J.-J. Chen, L. Thiele, and G. C. Buttazzo, "Applying real-time interface and calculus for dynamic power management in hard real-time systems," *Journal of Real–Time Systems*, vol. 47, no. 2, pp. 163–193, 2011.

[22] L. Santinelli, M. Marinoni, F. Prosperi, F. Esposito, G. Franchino, and G. Buttazzo, "Energy-aware packet and task co-scheduling for embedded systems," in *Proceedings of the 10th International Conference on Embedded Software*, pp. 279–288, ACM, 2010.

[23] A. Rahni, E. Grolleau, and M. Richard, "Feasibility analysis of non-concrete real-time transactions with edf assignment priority," in *Proceedings of the 16th Conference Real-Time and Networked Systems*, p. NA, Oct. 2008.

[24] Y. Wang, H. Liu, D. Liu, Z. Qin, Z. Shao, and E. H.-M. Sha, "Overhead-aware energy optimization for real-time streaming applications on multiprocessor system-on-chip," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 16, pp. 14:1–14:32, Apr. 2011.

[25] S. A. Brandt, S. Banachowski, C. Lin, and T. Bisson, "Dynamic integrated scheduling of hard real-time, soft real-time and non-real-time processes," in *Proceedings of the 24th IEEE Real-Time Systems Symposium*, (Cancun, Mexico), p. 396, Dec. 2003.

[26] V. Devadas and H. Aydin, "On the interplay of dynamic voltage scaling and dynamic power management in real-time embedded applications," in *Proceedings of the 8th International Conference on Embedded Software*, (Atlanta, GA, USA), pp. 99–108, ACM, 2008.

[27] H. Cheng and S. Goddard, "Integrated device scheduling and processor voltage scaling for system-wide energy conservation," in *Proceedings of the 2005 Workshop on Power Aware Real-time Computing*, pp. 24–29, Sept. 2005.

[28] S. K. Baruah, L. E. Rosier, and R. R. Howell, "Algorithms and complexity concerning the preemptive scheduling of periodic, real-time tasks on one processor," *Journal of Real–Time Systems*, 1990.

[29] R. Pellizzoni and G. Lipari, "Feasibility analysis of real-time periodic tasks with offsets," *Journal of Real–Time Systems*, vol. 30, pp. 105–128, 2005.

[30] C. Lin and S. A. Brandt, “Improving soft real-time performance through better slack management,” in *Proceedings of the 26th IEEE Real-Time Systems Symposium*, (Miami, FL, USA), pp. 410–421, Dec. 2005.

[31] S. M. Petters, M. Lawitzky, R. Heffernan, and K. Elphinstone, “Towards real multi-criticality scheduling,” in *Proceedings of the 15th IEEE Conference on Embedded and Real-Time Computing and Applications*, (Beijing, China), pp. 155–164, Aug. 2009.

[32] B. Nikolic, M. A. Awan, and S. M. Petters, “SPARTS: Simulator for power aware and real-time systems,” in *Proceedings of the 8th IEEE International Conference on Embedded Software and Systems*, (Changsha, China), pp. 999–1004, IEEE, Nov. 2011.

[33] B. Nikolic, M. A. Awan, and S. M. Petters, “SPARTS: Simulator for power aware and real-time systems,” 2011. http://www.cister.isep.ipp.pt/projects/sparts/.

[34] FreeScale Semiconductor, *MPC8536E PowerQUICC III Integrated Processor Hardware Specifications*. Document Number: MPC8536EEC, Rev 3, Nov. 2010.