On the Impact of Runtime Overhead on Energy-Aware Scheduling

Mario Bambagini[1], Marko Bertogna[2], Mauro Marinoni[1] and Giorgio Buttazzo[1]

[1]Scuola Superiore Sant'Anna, Pisa, Italy
[2]University of Modena and Reggio Emilia, Italy

***Abstract*—The real-time research community is often concerned with finding suitable assumptions to simplify the schedulability analysis of current real-time systems. This includes simplified power models, negligible scheduling overhead, negligible preemption cost, bounded cache misses and bus contention, etc. However, the actual behavior of a real application might be significantly different than that expected from a simplified system under the adopted set of assumptions.**

This paper investigates the impact that preemption cost might have on the energy consumption under a given energy-aware scheduling algorithm. A set of simulation experiments illustrate the influence of context switch and scheduling overhead on the actual energy consumed in a given system. Results show that, in certain conditions, the penalty due to the runtime overhead might be as large as the amount of energy saved using aggressive DPM and DVFS scheduling techniques. In this context, limited preemptive scheduling is proposed as a possible solution for limiting the main sources of overhead to fully exploit the benefits of power saving features of current computer architectures.

I. Introduction

Energy management is becoming more and more a crucial issue in modern embedded systems. Increasing lifetime, reducing energy bill, and decreasing the risk of temperature-related faults are only few of the motivations behind the growing interest in this topic. However, the reduction of power consumption leads to a decrease of performance, which could be critical for real-time applications where a set of time constraints need to be guaranteed.

Nowadays, the CMOS technology is the market leader and its power consumption could be roughly divided in dynamic and static components. The first one is due to the system activity and depends on the clock frequency, while the second one is a consequence of the static dissipation and could be reduced only switching off the gates. Over the years, hardware mechanisms have been developed to manage these two causes of consumption and a lot of techniques based on these mechanisms have been proposed to save energy. Two of the most widely used groups of techniques are *Dynamic Voltage and Frequency Scaling* (DVFS) and *Dynamic Power Management* (DPM). The DVFS approach trades energy with performance by decreasing the voltage and/or frequency of the processor to reduce the overall energy consumption. Since a frequency reduction increases the execution times of the computational activities, the objective of this technique is to find the slowest processor speed that still guarantees real-time constraints. On the other hand, DPM techniques aim at switching the processor in a low-power inactive state for the longest possible time, thus postponing the tasks execution as long as possible still guaranteeing the task real-time constraints.

With the growing importance of energy management, an increasing number of energy-aware scheduling algorithms have been proposed in the last years. Most of them consider either a fully preemptive or a non-preemptive model. In a fully preemptive system, if a newly activated task has a priority higher than the running one, a preemption occurs to move the running task in the ready queue and assign the processor to the new one. This implies a context switch overhead, which generally includes the time for suspending the running task and dispatching the new one, the time taken to flush the processor pipeline, and the cache-related preemption delay due to cache misses. Such an overhead is either neglected in the schedulability analysis (leading to an actual consumption higher than the predicted one) or overestimated by making pessimistic assumptions (so deteriorating the overall system performance). On the other side, non-preemptive scheduling can save a lot of runtime overhead and make the execution time more predictable, but it may introduce large blocking times in high priority tasks, degrading the schedulability of the system.

To mitigate the drawback of both approaches, limited preemptive scheduling has recently been proposed as a hybrid technique to decrease the number of preemptions without affecting the schedulability of the system. In this model, each task is view as a sequence of non-preemptive chunks and can be preempted only between chunks. Buttazzo, Bertogna, and Yao [1] have shown that limited preemptive scheduling increases the schedulability of fixed priority systems with respect to both fully preemptive and non-preemptive scheduling, even when preemption cost is neglected. The improvement is even more significant when considering preemption cost, due to the smaller number of context switches and the more predictable location of preemption points. Moreover, limited preemptive scheduling allows an implicit mutual exclusion management (when critical sections are encapsulated inside non-preemptive regions) and permits reducing stack memory requirements.

The objective of this paper is to investigate the impact of runtime overhead in energy-aware scheduling algorithms. In particular, simulation experiments show that the penalty associated with context switch overhead can significantly deteriorate the performance of energy-aware algorithms, introducing an additional energy consumption comparable to the amount of energy saved by aggressive DPM algorithms. Considerations will be also made on the scheduling overhead related to DPM techniques that require a significant number of operations at each idle interval. Finally, the impact of the overhead sources will be measured under different power models.

Preliminary results will be shown on the effectiveness of limited preemptive scheduling in reducing the minimum DVFS speed that can guarantee feasibility. Finally, some considerations will be presented on the integration of limited preemptive models with DPM techniques.

The remainder of the paper is organized as follows; Section II presents the state of art on energy saving algorithms; Section III introduces the system model in terms of tasks, energy consumption, and overhead sources; Section IV reports a set of simulation experiments aimed at showing the impact of runtime overhead on an energy-saving algorithm; Section V presents some ideas to decrease energy consumption by exploiting limited preemptive models; and Section VI concludes the paper.

II. Related work

This section presents the state of the art regarding energy management, first discussing DVFS algorithms, then DPM approaches.

One of the first papers about power management exploiting frequency scaling was due to Yao et al. [2]. The authors proposed an off-line algorithm that, given a task set, computes the minimum energy schedule under the Earliest Deadline First scheduling (EDF) algorithm [3] with a complexity $O(n \cdot \log^2(n))$. Then, they introduced two online methods to scale the speed according to the actual workload requiring $O(n)$ at every scheduling event and $O(n \cdot \log^2(n))$ at every task arrival, respectively. The analysis compares the efficiency of the algorithms with respect to different power models, but without taking switching and algorithms overhead into account.

Seong et al. proposed two algorithms. The first one (OLDVS) [4] accumulates the time generated by early terminations and exploits it to decrease the CPU speed so that the current task is completed at the same time at which it would have completed in the worst case. The second algorithm (OLDVS*) [5] divides each task in two parts: the first one is executed at a slower speed, while the second one is executed at a higher speed. The approach is based on the assumption that the probability of ending the task instance in the first part is significantly higher than finishing on the second part. Both algorithms do not take switching overheads into account. Bambagini et al. [6] extended the previous approaches by considering switching overheads. Moreover, they implemented the algorithms on a real embedded platform and showed that the more aggressive algorithm does not improve the energy consumption as expected, due to the actual power function.

Aydin et al. [7] proposed three algorithms with growing complexity. The first one computes the lowest CPU speed such that the task set is schedulable under the assumption that all tasks execute for their WCET. The second algorithm (DRA) keeps track of the times at which a task is going to be dispatched (computed off-line and stored in a dispatch queue). At runtime, if a task is dispatched earlier, the CPU is slowed down to prolong the execution until the original finishing time. The third algorithm (AGR) estimates the tasks completion times based on past instances and computes the lowest CPU speed to keep the task set feasible assuming that tasks execute for such estimates. However, since the estimations can be optimistic, the algorithm may speed the CPU up to recover from a task overrun. Neither this paper considers the overhead due to speed scaling, algorithm complexity or preemptions.

The problem of obtaining an optimal frequency from a discrete frequency range was discussed by Bini et al. [8]. The authors provided a method for computing the optimal speed off-line (that could be unavailable in a specific architecture) and introduced a speed modulation technique to achieve the required speed using two discrete values. The analysis selects the pair of frequencies that minimizes energy consumption also considering switching overheads. Since the algorithm works mostly off-line (as only the speed moluation is executed online) the algorithm overhead does not increase overall utilization. Despite its innovative contribution, such an off-line approach does not take advantage of tasks early terminations to further reduce consumption.

Some authors [9], [10] reported that online DVFS techniques that frequently scale the execution speed may lead to transient faults. This problem was also addressed by Zhao et al. [11], who proposed a recovery allowance and an additional recovery task to run in case of fault.

Another side effect of DVFS techniques was emphasized by Kim et al. [12], who noticed that such algorithms increase the number of preemptions, leading to a higher system utilization and, therefore, a higher energy consumption. This is because scaling the speed increases the computation times, exposing each task to a larger number of preemptions. To mitigate such a problem, they proposed two preemption control techniques integrated with a DVFS algorithm.

The raising impact of leakage power in modern architectures, highlighted by Kim et al. [13], is driving the research on power management toward DPM techniques.

Lee et al. [14] proposed two leakage control algorithms for procrastinating tasks execution as long as possible, both under fixed (LC-EDF) and dynamic (LC-DP) priority scheduling. Using a dual priority scheme [15], LC-DP computes the longest delay (*promotion time*) each task can suffer still satisfying its deadline. The drawbacks are that the critical speed is not taken into account (i.e., the system runs at the maximum speed) and the overhead introduced at runtime due to the higher complexity of the online analysis is high ($O(n^2)$).

Jejurikar et al. [16] proposed an approach (CS-DVS-P) based on critical speed analysis and task procrastination working for periodic tasks under EDF. First, an off-line DPM algorithm computes the maximum amount of time each task can spend in the sleep state within its period. Then, at run-time, sleep management is delegated to an external controller that switches the system off for the corresponding pre-computed time. Jejurikar and Gupta [17] extended the previous method to consider early terminations and fixed priority scheduling [18]. More precisely, the approach proposed in [18] differs from the first one for the computation of the maximum sleep time for each task. These values are obtained exploiting the dual priority of [14]. Since most of the computation is done offline, the online complexity reduces to $O(1)$. However, a dedicated external hardware is required to run the algorithm, increasing the power consumed, and no preemption overhead is considered.

Chen and Kuo [19] showed that the DPM part of the algorithm in [18] may lead to deadline misses, and proposed some solutions to avoid such a problem. The first method (OSS) simulates the execution of periodic tasks to compute the idle time available until the next deadline. Then, such a time is used to postpone the task activations and switch the system into the sleep state. An improved version of the algorithm (VOSS) further increases the sleep intervals making use of the *virtual blocking*, which is the maximum blocking that tasks can suffer. Neither the algorithm complexity ($O(n \cdot \log(n))$ at every idle time) nor preemption overhead were taken into account into such analysis.

Marinoni et al. [20] proposed a different approach: instead of computing a single speed off-line and postponing task execution on line, they compute the sleep interval and the speed at every idle time, to guarantee the feasibility within the next busy period. In addition, they also considered bandwidth allocation constraints that force the processor to be active during the assigned communication intervals. Altough such proposal aims at finding the optimal configuration at every idle time, the introduced complexity is pseudo-polynomial.

Awan and Petters [21] proposed to accumulate task execution slack to switch the processor off during such intervals under EDF, considering tasks with different criticality and processors with several low-power states and different break-even times. However, tasks are always executed at the maximum speed. The slack accounts both dynamic and static spare capacity. The static slack is due to an utilization lower than the maximum obtainable and the dynamic slack is ascribable to task early terminations. The same authors [22] observed that task procrastination algorithms significantly reduce the number of preemptions, leading to a lower system utilization and a larger slack for sleep states. Procrastinating tasks produces a synchronous activation of some tasks, which are then executed following the priority order without preemptions among them.

Irani et al. [23] addressed the problem concerning the estimation of the distance between the resulting schedule and the optimal one. More precisely, they proposed an α-approximation leakage-aware algorithm which produces a schedule α times worse than the optimal from the energy consumption point of view considering a continuous set of speeds.

Niu and Quan [24], [25] proposed an algorithm that, working with periodic tasks, computes at run-time the latest starting time for each task to guarantee the feasibility considering the jobs activated within the hyperperiod. Even though the proposed solution obtains good results, it introduces a considerable computation overhead that makes it unsuitable to actual real-time systems.

Huang et al. [26], [27] proposed an off-line analysis that combines DPM and Real-Time Calculus to estimate tasks arrivals, compute the CPU idle intervals and then, at run-time, modulate between active (at maximum speed) and sleep states. The main idea is similar to [8], but applied to DPM-based architectures. This approach leads to sleep intervals that are generally smaller and more frequent than those obtained by procrastination algorithms.

Rowe et al. [28] presented a technique that harmonizes task periods to cluster task execution such that processor idle times are lumped together. More precisely, the algorithm introduces the harmonizing period, so that the scheduler notifies task arrivals at integer multiples of such a period. If there is no task to execute, the system can put itself in sleep state until the next period. Although the algorithm allows entering a sleep state every time there is an idle period, it is less effective than postponing computation times, and it has a smaller schedulability bound.

Wang and Mishra [29] investigated DVS and dynamic cache reconfiguration in hard real-time systems for minimizing the overall energy consumption considering every single component, such as processor, cache and bus.

To the best of our knowledge, only Maxim et al. [30] have addressed the problem of exploiting a limited preemption model to further reduce energy consumption and the number of preemptions. However, their preliminary work does not consider the critical speed, switching overhead and DPM approaches.

III. System Model

We consider a set Γ of n fixed priority tasks, $\tau_1, \tau_2, \ldots, \tau_n$ executing upon a single processor platform with preemption support. We assume that tasks are indexed in decreasing priority order (i.e., if $0 < i < j \leq n$, then τ_i has higher priority than τ_j). The processor can vary the clock frequency f by selecting one of the available frequencies in a discrete set $\{f_1, \ldots, f_m\}$, ordered by ascending values. In the following, the normalized speed s, defined as $s = f/f_m$, will be used as a more convenient parameter.

Each sporadic task τ_i is characterized by a worst-case execution time (WCET) $C_i(s)$, which is a function of the speed, a relative deadline D_i and a period T_i. The WCET of τ_i depends on the actual speed of the processor and is computed as $C_i(s) = C_i^{NP}/s$, where C_i^{NP} denotes the time to execute τ_i in a non-preemptive mode at the maximum speed ($C_i^{NP} = C_i(s_m)$). Relative deadlines can be smaller than, equal to, or greater than periods. All parameters are assumed in $\mathbb{N}^+$. Each task generates an infinite sequence of jobs, with the first job arriving at any time and subsequent arrivals separated by T_i units of time.

A. Power model

The power model derived by Martin et al. in [31] represents a general expression of the power consumption of an active processor as a function of the speed:

$$P(s) = K_3 s^3 + K_2 s^2 + K_1 s + K_0. \quad (1)$$

The K_3 term is the coefficient related to the consumption of those components that vary both voltage and frequency. The second order term (K_2) describes the non linearity of DC-DC regulators in the range of the output voltage. The K_1 coefficient is related to the hardware components that can only vary the clock frequency, whereas K_0 represents the power consumed by the components that are not affected by the processor speed.

Note that the energy needed to execute a job is the product of the power and the execution time at the selected speed; moreover, a higher speed reduces the execution time, but increases the power consumption. Hence, the quantity that it is important to minimize is the energy consumption of each clock cycle $E_{clk}(s) = P(s)/s$. Considering the shape of $P(s)$ as a function of s, as reported in Equation (1), a critical speed s^* that minimized $E_{clk}(s)$ can be found [19].

As an example, two generic processors are considered with ten speeds uniformly distributed from 0.1 to 1.0 and power functions $P^{(dvfs)}(s) = 0.9s^3 + 0.1$ and $P^{(dpm)}(s) = 0.9s + 0.1$. The first one models a DVFS-sensitive architecture, as scaling down the speed (until $s^* = 0.4$) is energy-convenient. The second one represents a DPM-sensitive architecture, as running at a speed slower than the maximum one ($s^* = 1.0$) is not suitable from an energy point of view. In these cases, the curves representing the power functions and the energy per clock functions are reported in Figure 1 and Figure 2, respectively.

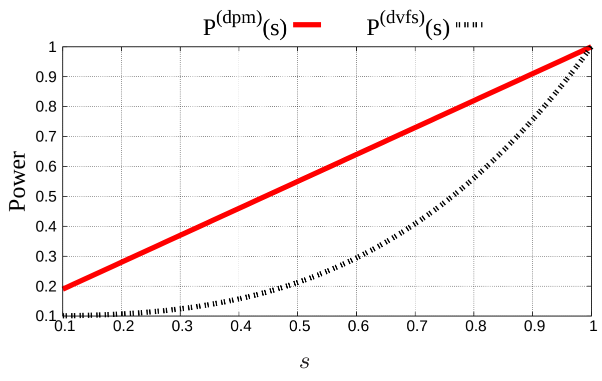

Fig. 1. $P^{(dvfs)}(s)$ and $P^{(dpm)}(s)$ functions.

Stating which model better describes the current architecture trend is not easy. In fact, although DPM-like models seem to be dominant in real processors (but less studied in the literature) [6], [21], a different trend has been reported by Kandhalu et al. in [32], considering the power consumed by a full board: a Motorola Xoom platform [33] equipped with a NVidia Tegra 2 [34]. In order to derive general results, in this paper we consider both power models.

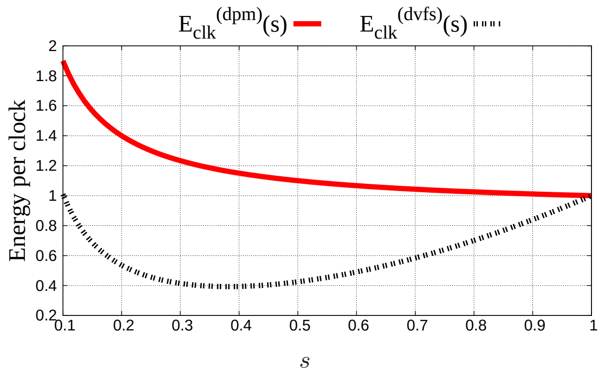

Fig. 2. $E_{clk}^{(dvfs)}(s)$ and $E_{clk}^{(dpm)}(s)$ functions.

An additional feature provided by almost all the current processors is the possibility to switch to low-power states while the code execution is suspended. Different device sub-systems could be switched off defining different low-power states. Each state σ_x is characterized by the power consumed in it, denoted as P_{σ_x}, and by the times required to enter and exit it, denoted as $\delta_{s\rightarrow\sigma_x}$ and $\delta_{\sigma_x\rightarrow s}$, respectively. The sum of such switching times, referred to as *break-even time*, δ_x, determines the shortest idle interval that must be available in the schedule to exploit the sleep state. Such an overhead is assumed to be independent of the actual running speed. The energy consumed during a transition from active to σ_x and viceversa is denoted by E_{δ_x}. Different power states are characterized by different parameters, as illustrated in Figure 3, which shows three different state transitions. The first case refers to a sleep state with a short break-even time, but a power consumption only slightly smaller than that of the active state. On the other hand, the third case refers to a state with the lowest power consumption, but with the longest transition times from active to sleep states and viceversa. Finding the most suitable low-power state depends on the available idle time and on the real-time constraints of the application.

Measurements carried out on Flex Boards equipped with a Microchip dsPIC33FJ256MC710 microcontroller [35] show that the power consumption at the slowest speed is 86.12mA, while the deepest sleep state consumes 26.40mA, requiring 19.30ms for handling a complete transition. On the other hand, the second low-power state is characterized by a consumption of 56.38mA and a much shorter break-even time, equal to 8 instruction cycles.

B. Overhead

When evaluating the performance of a energy-aware scheduling algorithm, different overhead sources need to be taken into account, including the preemption cost, the time taken to switch to a low-power state, and the complexity of the online algorithm. As mentioned in Section II, most of the existing works underestimate the importance of these factors, neglecting them in the schedulability analysis. This results in overly optimistic performance characterizations that can significantly differ from the real performance of the system.

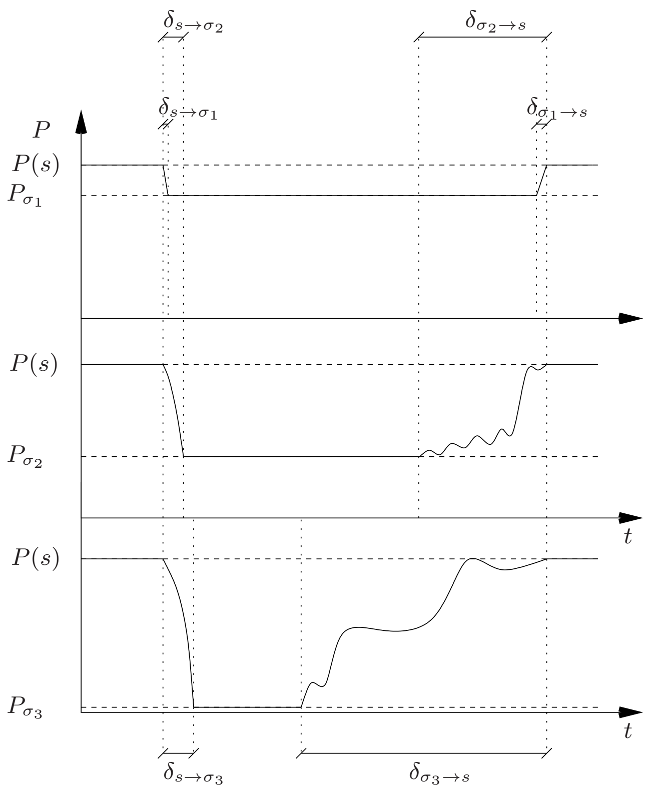

Fig. 3. Overhead due to switching from active to sleep state and viceversa.

In this paper, we explicitly consider the preemption cost, denoted as ξ, which includes several penalties, such as the context switch overhead, the pipeline invalidation delay, and the cache-related preemption delay. Since only the context switch time depends on the actual speed, the preemption cost ξ is assumed to be constant and speed independent.

The switching overhead between two frequencies affects both time and energy. Performing the operations needed to modify voltage and frequency requires time to change and stabilize these parameters. More precisely, the switching overhead increases with the difference between the two frequencies, as more operations need to be performed. For example, the speed tuning is delegated to a PLL module which is a close feedback loop, such that the wider the frequency difference, the longer the required time to stabilize the loop. On the dsPIC33FJ256MC710 microcontroller [35], it takes almost 10μs to scale from 40 to 8 MIPS, and barely 3μs to scale from 40 to 35 MIPS. Moreover, when passing from 2 MIPS to 40, the PLL must be turned on, since when operating at 2 MIPS the clock is taken directly from the external crystal. In this case, the tuning time is more than 1ms.

Finally, we also take into account the overhead related to the online execution of the energy-aware algorithm. Depending on the complexity and frequency of invocation of the algorithm, additional operations need to be performed, decreasing the available idle time and increasing the overall energy consumed by the system. As a consequence, aggressive DPM algorithms that require too many on-line operations can result in a decreased system performance, compensating the longer time spent in sleep modes with the additional energy required to run the algorithm.

IV. Experimental Results

This section presents a set of simulation experiments to show the impact of the runtime overhead on the overall energy consumption of the system.

Synthetic task sets consist of 10 periodic tasks randomly generated using the UUniFast algorithm [36], where the total utilization U is varied in a given range, and each computation time C_i is uniformly distributed in the interval $[100, 500]$. Periods are derived once computation times and task utilizations have been generated and relative deadlines are set equal to periods. Tasks are scheduled under fixed priorities assigned with the Rate Monotonic algorithm and each simulation run is performed until the hyperperiod. In each graph, output results for each parameter configuration are averaged over 100 different runs.

Simulations have been carried out assuming a processor with 19 discrete speeds varying in the range of $[0.1, 1]$ with step 0.05. Results are reported for both the power models introduced in Section III-A. For the sake of simplicity, a single sleep state is considered, with $P_\sigma = 0.0025$, and $E_\delta = 0.1 \cdot \delta$, where the break-even time δ is varied in a given range.

All simulations have been performed on the VOSS algorithm, by Chen and Kuo [19], because it has been shown to outperform other algorithms under fixed priority systems without requiring an additional hardware controller. However, it is worth noticing that VOSS is an on-line algorithm with a complexity of $O(n \cdot \log(n))$, which has to be paid at each idle interval. The slowest feasible speed used by VOSS has been computed using the more precise Response Time Analysis [37] (including preemption costs) instead of Liu and Layland's bound [3].

The first experiment considers the DVFS-sensitive power model, $P^{(dvfs)}(s)$, comparing the improvement obtained using aggressive DPM techniques with the penalty due to preemption overhead. Figure 4 reports the improvement of VOSS for different preemption costs ($\xi \in \{0, 5, 10\}$) and negligible break-even time $\delta = 0$, with respect to a plain DVFS solution with no preemption overhead. In other words, the figure shows the net improvement of VOSS (including the preemption cost) over a non-DPM scheduler that runs always at the speed obtained by the Response Time Analysis with null preemption cost.

For utilizations smaller than 0.35, the energy improvement is high. At these utilizations, the task sets are schedulable at speeds smaller than the critical one s^*. Since these speeds are non convenient form an energy point of view, the task sets are scheduled at speed s^*, leading to a significant slack time exploitable by the DPM algorithm.

For larger utilizations, task sets are instead executed at the minimum speed that guarantees all deadlines to be met, leading to smaller idle times. Nevertheless, when the preemption overhead is neglected ($\xi = 0$), VOSS is able to decrease the power consumed by around 10-13%. The relative improvement decreases for larger utilizations. This is because larger utilizations imply higher speeds and, therefore, higher energy requirements, so that the relative improvement of the DPM algorithm represents a smaller percentage of the overall power consumed.

When considering a more realistic scenario with a non negligible preemption cost, the relative performance of VOSS significantly decreases. For example, setting $\xi = 10$, the energy improvement at $U = 0.7$ drops from almost 10% down to less than 4%. This is due the large number of preemptions in the considered systems. As noted by Kim et al. [12], executing at the lowest possible speed leads to an increase in the number of preemptions. The reason is that there is less idle time, so that higher priority arrivals are more likely to happen when another task is executing, leading to a preemption. This causes an additional workload that must be executed by the system, increasing the energy consumption.

Increasing the utilization, the performance loss due to preemptions is larger. Comparing the curves with $\xi = 0$ and $\xi = 10$, the energy loss due to the preemption overhead goes from $1-2\%$ at utilization 0.4, to $6-7\%$ at utilization 0.8. Note that the number of preemptions is almost constant at different utilizations, due to speed scaling. The performance loss at higher utilizations is instead due to the larger impact of the preemption overhead when executing at higher speeds. The additional overhead is executed at energy-expensive speeds, increasing the overall consumption.

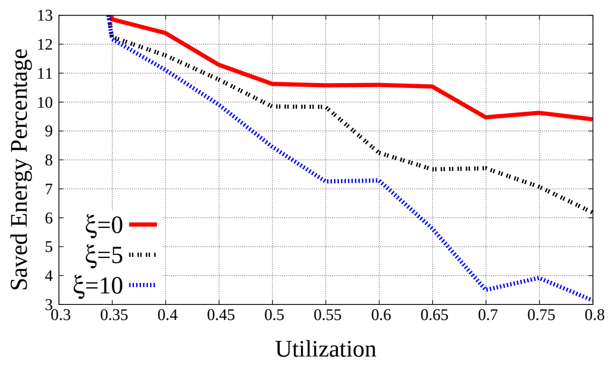

Fig. 4. Percentage of saved energy with $\delta = 0$ and $\xi \in \{0, 5, 10\}$.

When the break-even time is considered non-negligible, the situation is even worse, as reported in Figure 5 ($\delta = 500$) and Figure 6 ($\delta = 1000$). In such examples, the break-even time is considered once and twice the maximum task execution time, respectively. In many cases, there is a negative improvement, meaning that the impact of the preemption overhead is so high that the DPM algorithm is not able to compensate such overhead.

When the DPM-sentitive model $P^{(dpm)}(s)$ is considered, the impact of the preemption overhead is significantly smaller, due to several reasons. First of all, tasks are always executed at the highest speed, so that there is significant idle time in the system, improving the performance of DPM algorithms and

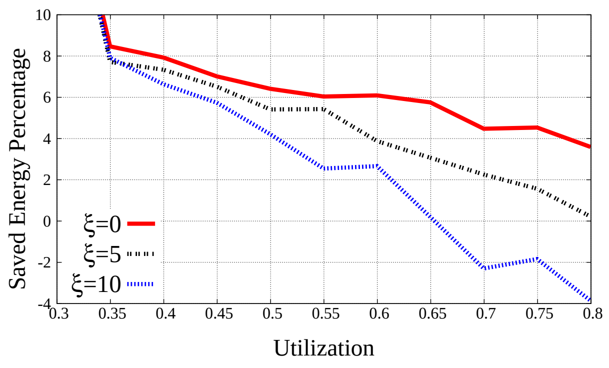

Fig. 5. Percentage of saved energy with $\delta = 500$ and $\xi \in \{0, 5, 10\}$.

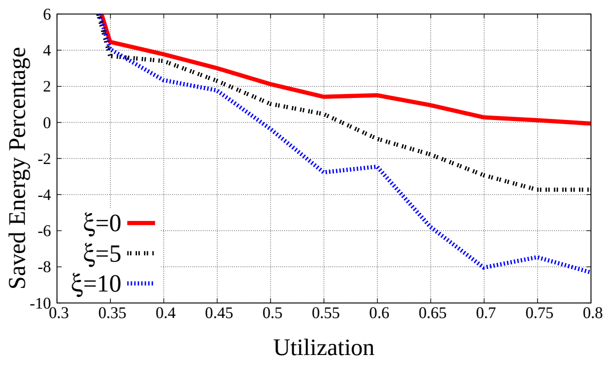

Fig. 6. Percentage of saved energy with $\delta = 1000$ and $\xi \in \{0, 5, 10\}$.

decreasing the number of preemptions. Moreover, as noticed by Awan and Petters [22], the number of preemptions is further reduced when applying DPM algorithms, because jobs arriving during the sleep phase are executed in priority order when the processor wakes up, without any preemption.

As already introduced in Section III-B, the overhead due to the DPM algorithm itself is another key parameter. Since VOSS has a complexity $O(n \cdot \log(n))$, its overhead has been considered equal to $2 \cdot n \log(n)$ units of time, at each idle interval. For $n = 8$, this corresponds to less than a half of the smallest task execution time that can be generated. The impact of the algorithm overhead on the overall energy consumption is reported in Figure 7 and Figure 8 for $P^{(dvfs)}(s)$ and $P^{(dpm)}(s)$, respectively. The experiments assume a negligible preemption cost ξ, a global utilization of 0.7, and a break-even time δ varying in $\{0, 500, 1000\}$.

With null break-even time, the impact of the algorithm overhead is around $1-2\%$, increasing with the number of tasks. Considering non-negligible break-even times ($\delta \in \{500, 1000\}$), the impact is much higher, up to 6% and 17% for the $P^{(dvfs)}(s)$ and $P^{(dpm)}(s)$ power model, respectively. The overhead has a greater impact on DPM-oriented architectures, since the algorithm is invoked more often due to the larger number of idle intervals in the system.

When increasing the number of tasks, the relative impact of the overhead decreases. This is because the ideal configuration with null algorithm overhead, which is used as a comparing term, has a drastically lower performance. In fact, a larger number of tasks implies a higher fragmentation of the idle intervals, which are often smaller than the break-even time, so that the DPM algorithm cannot be efficiently invoked.

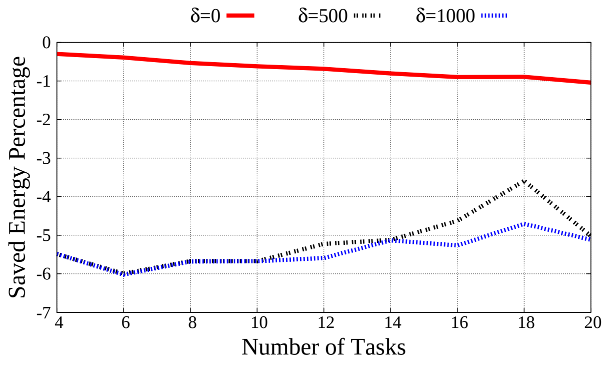

Fig. 7. Impact of algorithm overhead for $P^{(dvfs)}(s)$.

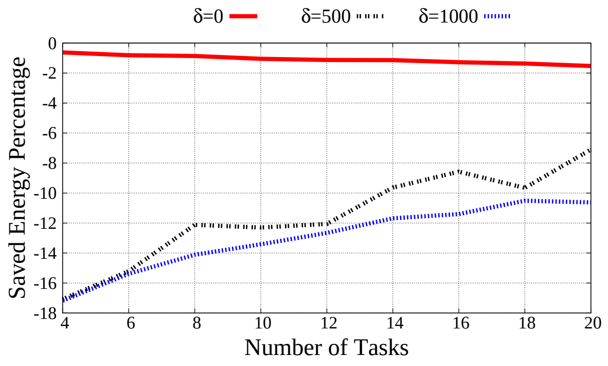

Fig. 8. Impact of algorithm overhead with $P^{(dpm)}(s)$.

V. Possible solutions

The results reported in the experimental section indicate that preemption cost degrades the performance of energy-aware algorithms. As a consequence, we believe that limited preemptive scheduling can be effectively exploited to further reduce energy consumption while guaranteeing real-time constraints.

The two main benefits coming from the limited preemptive approach are a reduced number of preemptions (which reduces the relative overhead) and an increased number of schedulable task sets with respect to both fully preemptive and non preemptive fixed priority schedulers.

In particular, the possible improvements concerning DVFS and DPM approaches are discussed in Section V-A and Section V-B, respectively.

A. DVFS approach

Using the limited preemptive algorithm to guarantee the feasibility of a higher number of task sets can be equivalent to using a lower speed to schedule the task sets than under fully preemptive and non-preemptive algorithms, hence saving more energy.

To better understand this aspect, let us consider a processor with two speeds, $s_1 = 0.5$ and $s_2 = 1$, executing two tasks, τ_1 and τ_2, with the following parameters: $C_1 = 30$, $T_1 = D_1 = 80$, $C_2 = 25$ and $T_2 = D_2 = 200$ (computation times are referred to speed s_2). Tasks are scheduled using Rate Monotonic and, for the sake of simplicity, preemption costs are considered negligible. The processor utilization factor at speed s_2 is $U = 0.5$ and the task set results feasible under fully-preemptive, non-preemptive, and limited preemptive scheduling. Switching to s_1, however, computation times become $C_1 = 60$ and $C_2 = 50$, causing a global utilization $U = 1$, which makes the task set unfeasible under both fully-preemptive and non-preemptive modes. Nevertheless, a feasible schedule can be found under the limited preemptive model by splitting task τ_2 in three chunks of length (under speed s_1) equal to 10, 20, and 20 units of time, respectively. The schedules produced by the Rate Monotonic under the three different preemption modes at s_2 are shown in Figure 9.

A more extensive analysis has been carried out to investigate such feature. More precisely, data reported in Figure 10 represents the average lowest speed that guarantees the task set feasibility considering non-preemptive, fully preemptive and limited preemptive models, obtained by considering $P^{(dvfs)}(s)$ and 700 task sets for each utilization step of 0.025. Although the non-preemptive model is not affected by the preemption cost, it requires always the highest average speed without guaranteeing the task set feasibility for at least half of the generated task sets at utilizations higher than 0.9. Even without considering preemption costs, the limited preemptive task model allows reducing the execution speed of 2% and 1% with respect to non-preemptive and fully preemptive modes, respectively. If preemption cost is accounted in the analysis, the fully preemptive performance drops significantly, even for $\xi = 10$, whereas the limited preemption behavior is only slightly degraded.

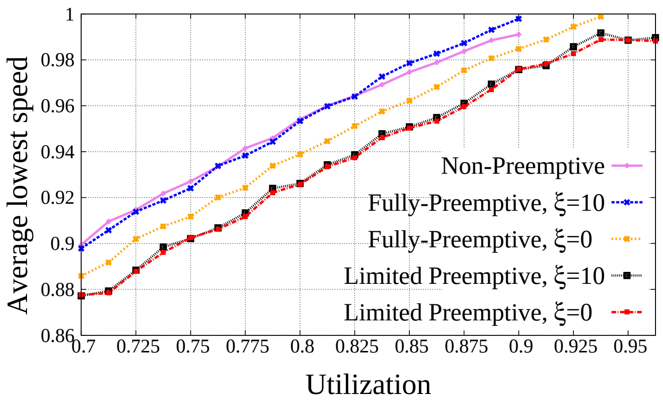

Fig. 10. Average lowest speed for different schedulers and preemption costs.

Hence, an interesting research challenge is to find an off-line algorithm that can define a set of preemption points such that the feasible speed at which a real-time task set can be executed is minimized. A promising possibility is to extend the algorithm proposed by Bertogna et al. [38] to return not

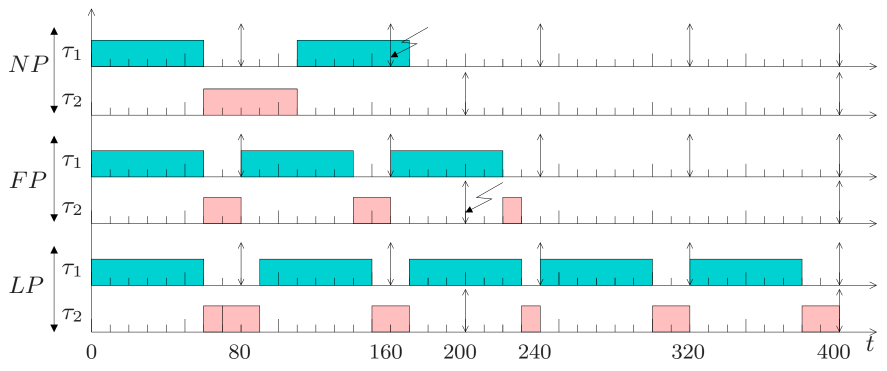

Fig. 9. Schedules produced by Rate Monotonic at speed $s = 0.5$ under Non-Preemptive (NP), Fully-Preemptive (FP), and Limited Preemptive (LP) scheduling.

only the set of preemption points, but also the optimal speed. Since the chunks' lengths depend on the actual speed and on the system feasibility, which in turns relies on the chunks' lengths, such an analysis is not trivial, as there is a cyclic dependency among the variables.

Moreover, another interesting aspect concerns the possibility of scaling the running speed at run-time while guaranteeing a low algorithm complexity. The main problem is related to the fact that scaling down the running speed would make the execution of the actual non-preemptive chunk longer, thus increasing the blocking time introduced in higher priority tasks.

B. DPM approach

From a DPM point of view, there is significant room for improvement, thanks to the larger blocking tolerances that can be obtained using the limited preemptive model. This property is mainly due to the smaller response times of tasks having a non-preemptive region at the end of their execution. In fact, all higher priority requests arriving while such a non-preemptive region is executed are postponed after the end of the considered task, decreasing the overall interference and reducing the number of preemptions.

Unfortunately, applying DPM techniques, like those adopted in VOSS, to limited preemptive models is not trivial. While the idle times of a fully preemptive schedule can be collected by VOSS just by postponing the task arrivals, the same is not necessarily true for a limited preemptive schedule, where the interference on a task changes depending on the particular time at which a higher priority job arrives. Robust methods are needed to safely delay task executions in order to exploit the allowed blocking tolerance to extend the sleep times. This has to be done with a reduced system overhead, without requiring overly complicated on-line computations that would compromise potential improvements.

VI. Conclusion

This paper investigated the impact of different overhead sources on the performance of power-aware scheduling techniques commonly adopted in the literature. In particular, the preemption overhead has been showed to significantly contribute to the overall power consumed, especially for DVFS-oriented architectures. These systems tend to execute at the smallest speed that guarantees all deadlines to be met, leading to a typically large number of preemptions. DPM techniques applied to these systems can be efficiently adopted to exploit the available idle time switching the processor into sleep state. However, the power consumed due to preemptions might be as large, if not bigger, than the performance gain obtained with these techniques, depending on system parameters like preemption cost, break-even time and task set utilization.

DPM-oriented architectures are less influenced by the preemption overhead, because they execute at larger speeds than the smallest feasible one. This results in larger idle times, with fewer preemptions and bigger energy gains for DPM techniques. However, these algorithms are subject to a significant on-line overhead when aggressive techniques are adopted to collect the available slack. Using complex on-line routines has a big impact on the overall power consumed. This impact is bigger for DPM-oriented architectures, which tend to execute at a larger speed. The reason is twofold. Firstly, the abundance of idle time causes a more frequent invocation of the DPM routines, increasing the overhead. Secondly, such an overhead is paid at larger speeds and, therefore, with a higher cost in terms of energy.

The adoption of limited preemptive techniques for power-aware scheduling algorithms has been proposed to decrease the system overhead, with particular relation to the context switch overhead. The improved schedulability properties of limited preemptive systems can be exploited to decrease the processor speed of DVFS-oriented systems. The larger blocking tolerances can be used to extend the sleep times in DPM-oriented systems. However, particular care should be taken on the complexity of the on-line algorithm, limiting as much as possible the amount of operations to be performed at run-time.

References

[1] G. C. Buttazzo, M. Bertogna, and G. Yao, “Limited preemptive scheduling for real-time systems: a survey,” *IEEE Transactions on Industrial Informatics*, 2012, to appear.

[2] F. Yao, A. Demers, and S. Shenker, “A scheduling model for reduced cpu energy,” in *Proc. of the 36th Annual Symp. on Foundations of Computer Science*, ser. FOCS ’95. Washington, DC, USA: IEEE Computer Society, 1995, pp. 374–.

[3] C. Liu and J. Layland, “Scheduling algorithms for multiprogramming in a hard-real-time environment,” *Journal of the Association for Computing Machinery*, vol. 20, no. 1, pp. 46–61, January 1973.

[4] C.-H. Lee and K. G. Shin, “On-line dynamic voltage scaling for hard real-time systems using the edf algorithm,” in *Proc. of the 25th IEEE International Real-Time Systems Symp.* Washington, DC, USA: IEEE Computer Society, 2004, pp. 319–327.

[5] M.-S. Gong, Y. R. Seong, and C.-H. Lee, “On-line dynamic voltage scaling on processor with discrete frequency and voltage levels,” in *Proc. of the 2007 International Conference on Convergence Information Technology*, ser. ICCIT ’07. Washington, DC, USA: IEEE Computer Society, 2007, pp. 1824–1831.

[6] M. Bambagini, F. Prosperi, M. Marinoni, and G. Buttazzo, “Energy management for tiny real-time kernels,” in *Energy Aware Computing (ICEAC), 2011 International Conference on*. IEEE, 2011, pp. 1–6.

[7] H. Aydi, P. Mejía-Alvarez, D. Mossé, and R. Melhem, “Dynamic and aggressive scheduling techniques for power-aware real-time systems,” in *Proc. of the 22nd IEEE Real-Time Systems Symp.*, ser. RTSS ’01. Washington, DC, USA: IEEE Computer Society, 2001, pp. 95–.

[8] E. Bini, G. Buttazzo, and G. Lipari, “Speed modulation in energy-aware real-time systems,” in *Proc. of the 17th Euromicro Conference on Real-Time Systems*. Washington, DC, USA: IEEE Computer Society, 2005, pp. 3–10.

[9] Y. Zhang and K. Chakrabarty, “Energy-aware adaptive checkpointing in embedded real-time systems,” in *Proceedings of the conference on Design, Automation and Test in Europe - Volume 1*, ser. DATE ’03. Washington, DC, USA: IEEE Computer Society, 2003, pp. 10 918–. [Online]. Available: http://dl.acm.org/citation.cfm?id=789083.1022841

[10] D. Zhu, R. Melhem, and D. Mosse, “The effects of energy management on reliability in real-time embedded systems,” in *Proceedings of the 2004 IEEE/ACM International conference on Computer-aided design*, ser. ICCAD ’04. Washington, DC, USA: IEEE Computer Society, 2004, pp. 35–40. [Online]. Available: http://dx.doi.org/10.1109/ICCAD.2004.1382539

[11] B. Zhao, H. Aydin, and D. Zhu, “Energy management under general task-level reliability constraints,” in *Proceedings of the 2012 IEEE 18th Real Time and Embedded Technology and Applications Symposium*, ser. RTAS ’12. Washington, DC, USA: IEEE Computer Society, 2012, pp. 285–294. [Online]. Available: http://dx.doi.org/10.1109/RTAS.2012.30

[12] W. Kim, J. Kim, and S. L. Min, “Preemption-aware dynamic voltage scaling in hard real-time systems,” in *Proceedings of the 2004 international symposium on Low power electronics and design*, ser. ISLPED ’04. New York, NY, USA: ACM, 2004, pp. 393–398. [Online]. Available: http://doi.acm.org/10.1145/1013235.1013328

[13] Kim, T. Austin, J. S. Hu, and M. Jane, “Leakage current: Moore’s law meets static power,” 2003.

[14] Y.-H. Lee, K. P. Reddy, and C. M. Krishna, “Scheduling techniques for reducing leakage power in hard real-time systems,” in *ECRTS’03*, 2003, pp. 105–112.

[15] R. Davis and A. Welling, “Dual priority scheduling,” in *Proceedings Real Time Systems Symposium*, ser. RTSS ’05, 1995.

[16] R. Jejurikar, C. Pereira, and R. K. Gupta, “Leakage aware dynamic voltage scaling for real time embedded systems,” in *In Proc. of the Design Automation Conference*, 2004, pp. 275–280.

[17] R. Jejurikar and R. Gupta, “Dynamic slack reclamation with procrastination scheduling in real-time embedded systems,” in *Proceedings of the 42nd annual Design Automation Conference*, ser. DAC ’05. New York, NY, USA: ACM, 2005, pp. 111–116. [Online]. Available: http://doi.acm.org/10.1145/1065579.1065612

[18] ——, “Procrastination scheduling in fixed priority real-time systems,” in *In Proceedings of the Language Compilers and Tools for Embedded Systems*, 2004.

[19] J.-J. Chen and T.-W. Kuo, “Procrastination for leakage-aware rate-monotonic scheduling on a dynamic voltage scaling processor,” *SIGPLAN Notices*, vol. 41, no. 7, pp. 153–162, Jun. 2006.

[20] M. Marinoni, M. Bambagini, F. Prosperi, F. Esposito, G. Franchino, L. Santinelli, and G. Buttazzo, “Platform-aware bandwidth-oriented energy management algorithm for real-time embedded systems,” in *Emerging Technologies & Factory Automation (ETFA), 2011 IEEE 16th Conference on*. IEEE, 2011, pp. 1–8.

[21] 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 2011 23rd Euromicro Conference on Real-Time Systems*, ser. ECRTS ’11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 92–101.

[22] ——, “The roman conquered by delay: Reducing the number of preemptions using sleep states,” in *The 17th IEEE Real-Time and Embedded Technology and Applications Symposium Work-In-Progress Session*, 2011.

[23] S. Irani, S. Shukla, and R. Gupta, “Algorithms for power savings,” *ACM Trans. Algorithms*, vol. 3, no. 4, Nov. 2007. [Online]. Available: http://doi.acm.org/10.1145/1290672.1290678

[24] L. Niu and G. Quan, “Reducing both dynamic and leakage energy consumption for hard real-time systems,” in *Proceedings of the 2004 international conference on Compilers, architecture, and synthesis for embedded systems*, ser. CASES ’04. New York, NY, USA: ACM, 2004, pp. 140–148. [Online]. Available: http://doi.acm.org/10.1145/1023833.1023854

[25] G. Quan, L. Niu, X. S. Hu, and B. Mochocki, “Fixed priority scheduling for reducing overall energy on variable voltage processors,” in *Proceedings of the 25th IEEE International Real-Time Systems Symposium*, ser. RTSS ’04. Washington, DC, USA: IEEE Computer Society, 2004, pp. 309–318. [Online]. Available: http://dx.doi.org/10.1109/REAL.2004.23

[26] K. Huang, L. Santinelli, J.-J. Chen, L. Thiele, and G. C. Buttazzo, “Periodic power management schemes for real-time event streams.” in *the 48th IEEE Conf. on Decision and Control (CDC)*. Shanghai, China: IEEE, 2009, pp. 6224–6231.

[27] ——, “Adaptive dynamic power management for hard real-time systems,” in *the 30th IEEE Real-Time Systems Symp. (RTSS)*. Washington D.C. U.S.: IEEE, 2009, pp. 23–32.

[28] A. Rowe, K. Lakshmanan, H. Zhu, and R. Rajkumar, “Rate-harmonized scheduling and its applicability to energy management,” *IEEE Trans. Industrial Informatics*, vol. 6, no. 3, pp. 265–275, 2010.

[29] W. Wang and P. Mishra, “System-wide leakage-aware energy minimization using dynamic voltage scaling and cache reconfiguration in multitasking systems,” *IEEE Trans. VLSI Syst.*, vol. 20, no. 5, pp. 902–910, 2012.

[30] C. Maxim, L. Cucu-Grosjean, and O. Zendra, “Towards reducing preemptions to save energy,” in *the 5th Junior Researcher Workshop on Real-Time Computing (JRWRTC 2011)*, Nantes, France, Sep. 2011.

[31] T. Martin and D. Siewiorek, “Non-ideal battery and main memory effects on cpu speed-setting for low power,” *IEEE Transactions on VLSI Systems*, vol. 9, no. 1, pp. 29–34, 2001.

[32] A. Kandhalu, J. Kim, K. Lakshmanan, and R. Rajkumar, “Energy-aware partitioned fixed-priority scheduling for chip multi-processors,” in *RTCSA (1)*, 2011, pp. 93–102.

[33] “Motorola xoom android tablet,” http://www.motorola.com.

[34] “Nvidia tegra 2,” http://www.nvidia.com/.

[35] “Microchip web site,” http://www.microchip.com/.

[36] E. Bini and G. C. Buttazzo, “Measuring the performance of schedulability tests,” *Real-Time Syst.*, vol. 30, no. 1-2, pp. 129–154, May 2005.

[37] N. Audsley, A. Burns, M. Richardson, K. Tindell, and A. Wellings, “Applying new scheduling theory to static priority pre-emptive scheduling,” *Software Engineering Journal*, vol. 8, no. 5, pp. 284–292, September 1993.

[38] M. Bertogna, G. Buttazzo, and G. Yao, “Improving feasibility of fixed priority tasks using non-preemptive regions,” in *Proceedings of the 2011 IEEE 32nd Real-Time Systems Symposium*, ser. RTSS ’11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 251–260. [Online]. Available: http://dx.doi.org/10.1109/RTSS.2011.30