Peripheral-conscious energy-efficient scheduling for weakly hard real-time systems

Linwei Niu*

Department of Math and Computer Science,
West Virginia State University,
Institute, WV 93311, USA
Email: lniu@wvstateu.edu
*Corresponding author

Gang Quan

Electrical and Computer Engineering Department,
Florida International University,
Engineering Center 3911, 10555 West Flagler Street,
Miami, FL 33174, USA
Email: gang.quan@fiu.edu

Abstract: In this paper, we study the problem of reducing the energy consumption for a weakly hard real-time system. The weakly hard real-time system is modelled by the (m, k)-constraints, which require that at least m out of any k consecutive jobs of a task meet their deadlines. Since the system energy is consumed not only by the processor alone but also in a large part by other peripheral devices, we first propose a static approach, with the specifications of peripheral devices taken into consideration, to partition the jobs into mandatory/optional jobs to achieve the dual goals of (m, k)-guarantee and overall energy minimisation. Based on that, we present a dynamic scheduling algorithm that adopts preemption control technique and dynamic mandatory/optional partitioning strategy to reduce the energy consumption of the whole system dynamically. Our approach can effectively minimise the system-wide energy consumption and guarantee the (m, k)-deadlines at the same time. The novelty and effectiveness of our techniques are demonstrated through extensive simulation studies.

Keywords: peripheral; energy-efficient scheduling; weakly hard real-time systems.

Reference to this paper should be made as follows: Niu, L. and Quan, G. (2015) 'Peripheral-conscious energy-efficient scheduling for weakly hard real-time systems', *Int. J. Embedded Systems*, Vol. 7, No. 1, pp.11–25.

Biographical notes: Linwei Niu received his BS in Computer Science and Technology from Peking University, Beijing, China in 1998, his MS in Computer Science from the State University of New York at Stony Brook in 2001, and his PhD in Computer Science and Engineering from the University of South Carolina in 2006. Currently, he is an Associate Professor in the Department of Math and Computer Science, West Virginia State University, USA. His research interests include power-aware design for embedded systems, design automation, real-time scheduling and software/hardware co-design.

Gang Quan is currently an Associate Professor in the Electrical and Computer Engineering Department, Florida International University. He received his PhD degree from the Department of Computer Science and Engineering, University of Notre Dame, USA, his MS degree from the Chinese Academy of Sciences, Beijing, China, and his BS degree from the Department of Electronic Engineering, Tsinghua University, Beijing, China. His research interests and expertise include real-time systems, embedded system design, power-/thermal-aware computing, advanced computer architecture and reconfigurable computing. He is the recipient of a National Science Foundation Faculty Career Award.

1 Introduction

Energy reduction is critical to increase the mobility for today's pervasive computing systems and thus becomes a widespread research area (Kathuria, 2013; Yang et al., 2013; Ji et al., 2013). Power aware scheduling has been proven to be an effective way to reduce the energy consumption. Rooted in the traditional real-time scheduling technology, the power aware scheduling techniques change the system computing performance accordingly based on the dynamically varying computation demand. Two main types of techniques are reported in the literature. The first one is commonly known as the *dynamic power down* (DPD), i.e., to shut down a processing unit and save power when it is idle. The second one is called *dynamic voltage scaling* (DVS) which updates the processor's supply voltages and working frequencies dynamically.

Extensive power aware scheduling techniques have been published for energy reduction, but most of them (e.g., Aydin et al., 2001; Kim et al., 2002; Quan et al., 2009) have been focused solely on reducing the processor energy consumption. While the processor is one of the major power hungry units in the system, other peripherals such as network interface card, memory banks, disks also consume significant amount of power. The empirical study by Viredaz and Wallach (2003) on a Itsy pocket computer, which is a typical portable electronic systems based on the StrongARM SA-1100 processor, reveals that the processor core consumes around 28.8% of total power when playing a video file on a hardware testbed (Viredaz and Wallach, 2003) for handheld devices, while the DRAM consumes about 28.4% of the total power. Note that this testbed (Viredaz and Wallach, 2003) lacks disk storage and wireless networking capability, which may contribute as much power consumption as the processor core if not more (Zedlewski et al., 2003; Doherty et al., 2001). This implies that the techniques that attack the processor energy alone may not be overall energy efficient. On the other hand, while DVS techniques have proven to be able to dramatically reduce the dynamic power consumption of the processor, most of the peripheral devices do not have the DVS capabilities. For peripheral devices, the most efficient way to save power is simply shut them down when they are not in use. As a result, the research on employing a combination of DVS and DPM has also gained its momentum to reduce the system-wide energy consumption.

Recently, several techniques (e.g., Kim and Ha, 2001) have been proposed to reduce the energy consumption for *hard* real-time systems consisting of both core processors and peripheral devices. However, few real-time applications are truly *hard* real-time, i.e., many practical real-time applications can allow some deadline misses provided that user's perceived quality of service (QoS) constraints (Wang et al., 2012) can be satisfied. While the statistic information such as the average deadline miss rate is commonly used to quantify the QoS requirements for the system, this metric can be problematic for some real-time applications. For example, many real-time applications can tolerate occasional deadline misses of the real-time tasks, and the information carried by these tasks can be estimated to a reasonable accuracy using techniques such as interpolation. However, even with very low overall miss rate tolerance, it is still possible that a large number of deadline misses could occur consecutively in a short period of time such that critical information could be lost (in those time periods).

The *weakly hard real-time model* is more suitable for this type of applications. In the weakly-hard real-time model, tasks have both firm deadlines (i.e., task instances that missed their deadlines are not counted as valid ones) and a throughput requirement (i.e., *sufficient* task instances must meet deadlines to provide required quality levels). Ramamritham and Stankovic (1994) proposed a so-called (m, k)-model, with a periodic task being associated with a pair of integers, i.e., (m, k), such that among any k consecutive instances of the task, at least m of the instances must finish by their deadlines for the system behaviour to be acceptable. A *dynamic failure* occurs, which implies that the temporal QoS constraint is violated and the scheduler is thus considered failed, if within any consecutive k jobs more than ($k - m$) job instances miss their deadlines. Based on this (m, k)-model, Ramanathan (1999) proposed to partition the jobs into *mandatory* and *optional* jobs. So long as all the mandatory jobs can meet their deadlines, the (m, k)-constraints can be ensured.

In this paper, we study the problem of reducing the system wide energy consumption for the weakly hard real-time system modelled with the (m, k)-model. The problem becomes more challenging since we need to deal with not only the tradeoffs between DVS and DPD, but also the mandatory/optional partitioning problems, i.e., to determine which jobs are mandatory (whose deadlines have to be met to guarantee no dynamic failure occur) and which jobs can be optional. This problem is known as NP-hard (Quan and Hu, 2000). In this paper, we propose a novel mandatory/optional job partitioning strategy and a sufficient condition for checking the feasibility. Based on which, we present a dynamic scheduling scheme that adopts preemption control (Kim and Roy, 2004) technique and mandatory job pattern adjustment (Niu and Quan, 2006a) simultaneously to achieve higher efficiency in energy savings.

The rest of the paper is organised as follows. Section 2 talks about the related work. Section 3 presents the system model, and motivations. Section 4 presents our new approach in determining the mandatory/optional job partitioning and a feasibility condition to guarantee the (m, k)-firm deadlines. Section 5 presents our dynamic algorithm to reduce the system energy. In Section 6, we presents our experimental results. Section 7 draws the conclusions.

2 Related work

Several weakly hard models have been proposed for soft real time systems, e.g., Bernat and Burns (1997), Quan and Hu (2000), Hamdaoui and Ramanathan (1995) and

Ramanathan (1999). Koren and Shasha (1995) proposed a static partitioning strategy, called *the deeply-red pattern* or *R-pattern*. According to this scheme, for job τ_{ij}, i.e., the j^{th} job of task τ_i, we have

$$\pi_{ij} = \begin{cases} 1 & 0 \le j \bmod k_i < m_i \\ 0 & \text{otherwise} \end{cases} \qquad j = 0,1, \cdots, k_i - 1 \tag{1}$$

Therefore, τ_{ij} is mandatory if its corresponding bit is '1' and optional otherwise. Ramanathan (1999) proposed another partitioning strategy as follows.

$$\pi_{ij} = \begin{cases} 1 & \text{if } j = \left\lfloor \left\lceil \frac{j \times m_i}{k_i} \right\rceil \times \frac{k_i}{m_i} \right\rfloor \\ 0 & \text{otherwise} \end{cases} \qquad j = 0,1, \cdots, k_i - 1 \tag{2}$$

The mandatory jobs defined with formula (2) are evenly distributed along the pattern, and thus referred as the *evenly distributed pattern* or *E-pattern*. Bernat and Cayssials (2001) proposed a bi-modal scheduler in which the tasks are first scheduled according to the generic scheduling policy in the *normal* mode and then switched to the *panic* mode if the dynamic failure is likely to occur.

Recently, there have been increasing research efforts (Lee et al., 2003; Jejurikar et al., 2004; Jejurikar and Gupta, 2004b; Kong et al., 2011) that use real-time scheduling techniques to reduce energy consumption for real-time embedded systems. To balance the dynamic power and static power during the execution of a job, Jejurikar et al. (2004) propose to use the critical speed as the lower bound for speed scaling. Since the critical speed may require the processor to run at *higher-than-necessary* speeds to execute a given set of real-time tasks, it can potentially fragment the execution of tasks and cause a large number of scattered idle intervals. To effectively reduce the energy consumption during these idle intervals, procrastination scheduling was proposed in Jejurikar et al. (2004) with the purpose of extending the idle intervals to facilitate shut down. Their approach assumes all tasks running with worst case execution times. Considering early completion of jobs, which is pervasive in most real-time systems, new dynamic reclaiming technique was proposed in Jejurikar et al. (2005) to incorporate job slack time into procrastination scheduling. However, as shown in Niu (2011), their approach might not be able to merge the idle intervals efficiently and is therefore less efficient in reducing the overhead of shutting-down. Lee et al. (2003) proposed a leakage reduction scheduling technique called LC-DP, by extending the dual-priority (DP) scheduling model presented in Davis and Wellings (1995) for real-time systems based on fixed-priority (FP) scheme. However, as shown in Jejurikar and Gupta (2004b), the LC-DP algorithm cannot guarantee the deadlines of tasks because of its discrepancy with the original dual priority scheduling algorithm (Davis and Wellings, 1995). To guarantee deadlines, Jejurikar and Gupta (2004b) further proposed to delay the execution of tasks by the minimal promotion time over all lower and equal priority tasks based on the dual priority scheduling. However, as shown in Chen and Kuo (2006), this approach cannot guarantee the deadlines for real-time tasks based on FP scheme either. Also Chen and Kuo (2006) proposed an FP-based online simulated-scheduling (VOSS) algorithm which scales the job speed when the idle time length is less than the break even time or the 'effective power' based on virtual schedules can be less. However, their approach must assume worst case execution times for all real-time jobs (to construct the virtual schedule) and might not be applicable when real-time jobs present actual execution times less than their worst case. In order to facilitate processor shut-down, Awan and Petters (2011) proposed a 'race to halt' approach which attempts to put the processor into sleeping mode whenever the reclaimable slack time reaches certain length limit for the idle interval. However, since this idle interval length limit is statically computed by assuming all real-time jobs will present their worst case execution times, it is rather pessimistic in a more common situation where most real-time jobs present actual execution times much less than their worst case. As a result, it may not be able to merge the sleeping intervals efficiently to further reduce the energy overhead of shutting-down. Kim and Roy (2004) proposed a preemption control technique to reduce preemptions among tasks and hence the energy cost incurred by preemptions among tasks. However, their approach needs to increase the processing speeds of the jobs under consideration, which could, to some extent, counteract the energy reduction from reducing preemptions. Bambagini et al. (2013) proposed an energy reduction framework by exploring the limited preemptive scheduling framework (Buttazzo et al., 2013; Bertogna and Baruah, 2010), which attempts to reduce the preemption among tasks by using lower priority tasks to 'block' higher priority tasks. Their approaches need to handle the 'blocking' among tasks (based on the blocking protocols) at each job arrival, which will incur significant overall time/energy overhead. Moreover, their approach is based on FP scheme and needs to divide each real-time task into a set of non-preemptive regions, which might not be effective for general fully preemptive EDF scheduling.

The above approaches are focused on saving energy consumed by the processor only. More recently, a number of researches (e.g., Jejurika and Gupta, 2004a; Kim and Ha, 2001; Kim and Roy, 2004; Zhuo and Chakrabarti, 2005) are reported to reduce the energy consumption for systems consisting of DVS processors and peripheral devices. Kim and Ha (2001) proposed a technique for *hard* real-time system, while scheduling decisions are made on a timeslot-by-timeslot basis. To facilitate a run-time mechanism, the processor speed for each task is determined by analysing the energy savings based on a pre-determined set of execution times. Jejurikar and Gupta (2004a) introduced a heuristic search method to find the critical speed to balance the energy consumption between the processor and peripheral devices. Zhuo and Chakrabarti (2005) proposed a theoretical formulation of the optimal scaling factor and computed it numerically. Based on

which, they also incorporated the idea of preemption control in Kim and Roy (2004) to improve the energy savings. In Zhao and Aydin (2009) and Devadas and Aydin (2012), the interplay between DVS and DPD were studied to minimise energy consumption for real time systems containing a core processor and multiple devices. The system model of their approach contains only a single task with its deadline equal to the period. Kong et al. (2010) proposed an offline approach based on mathematical programming to integrate DVS and DPD to deal with system energy minimisation for multiple tasks. Their approaches target the so called 'frame-based' system which is a very special case of real-time applications that all tasks have the same deadlines and periods. In Devadas and Aydin (2010), more advanced techniques were provided to minimise the system level energy consumption by inserting so-called 'device forbidden region' (DFR) into real time tasks. The DFRs are some precomputed time intervals during which the devices are forced into sleeping mode and will be treated as separate tasks with higher priority than all the original tasks in the system. However, as shown in Devadas and Aydin (2010), the problem of generating feasible schedule with DFRs is NP-hard in the strong sense. Although the authors explored some heuristics to determine the DFRs offline for hard real-time systems with deadlines equal to periods, it is not clear if that approach could be applied to systems with weakly hard constraints. There are also a number of researches investigating the scheduling problem for system with non-DVS processor and I/O devices (Cheng and Goddard, 2006).

All the approaches above target hard real-time systems.

We are more interested in developing scheduling techniques for weakly hard real-time systems. Hua and Qu (2004) introduced a greedy approach to minimise the energy consumption for systems with (m, k)-constraints running on a processor with two different speeds. However, their approach cannot guarantee the (m, k)-constraints, even though the system is *underloaded.* AlEnawy and Aydin (2005) introduced a scheduling technique to maximise (instead of guarantee) the quality service level under energy constraints for real-time systems with (m, k)-constraints. Niu and Quan (2006a) presented a combined static/dynamic DVS scheduling method to reduce processor energy with (m, k)-guarantee. All of these techniques only take the processor energy consumption into consideration. To incorporate the energy of the peripheral devices into consideration, Niu and Quan, (2006b) investigated the scheduling problem for a system that consists of a non-DVS processor and a single peripheral device. In this paper, we target reducing the system-wide energy for a weakly hard real time system model that contains a DVS processor and multiple peripheral devices. And each device can have different power characteristics.

3 Preliminary

In this section, we first introduce the system and architecture model, followed by a motivation example.

3.1 System models

The real-time system considered in this paper contains n independent periodic tasks, $\mathcal{T} = \{\tau_0, \tau_1, \cdots, \tau_{n-1}\}$, scheduled according to the earliest deadline first (EDF) policy. Each task contains an infinite sequence of periodically arriving instances called *jobs*. Task τ_i is characterised using five parameters, i.e., (T_i, D_i, C_i, m_i, k_i). T_i, Di ($D_i \leq T_i$), and C_i represent the period, the relative deadline and the worst case execution time for τ_i, respectively. A pair of integers, i.e., (m_i, k_i) ($0 < m_i \leq k_i$), represent the QoS requirement for τ_i, requiring that, among any k_i consecutive jobs of τ_i, at least m_i jobs meet their deadlines. Each periodic task consists of a sequence of instances, called jobs. When it does not cause confusion, we use $J_i = (r_i, c_i, d_i)$ to represent the current or upcoming job of task τ_i, where r_i, c_i, and d_i are the arrival time, actual execution cycles, and absolute deadline, respectively.

The system architecture consists of a core DVS processor and a number of devices, M_0, M_1, ..., M_p. The DVS processor used in our system can operate at a finite set of discrete supply voltage levels $\mathcal{V} = \{V_1, ..., V_{\max}\}$, each with an associated speed. To simplify the discussion, we normalise the processor speeds to $S_{\max}$, the speed corresponding to $V_{\max}$, which results in $\mathcal{S} = \{S_1, ..., 1\}$. We assume that C_i is the worst case execution time for task τ_i in the highest voltage mode. Therefore, if τ_i is executed under speed S_j, the worst case execution time for τ_i becomes $\frac{C_i}{S_j}$. The devices do not have DVS capability and can only support the DPD mechanism.

We denote the processor power with P_{cact} when running a task, and P_{cidle} when the processor is idle (yet still *on*). When the processor is shut down, its power consumption is denoted as P_{csleep}. Besides running on the DVS processor, each task τ_i requires to access a subset of peripheral devices $\Phi^i \subseteq \{M_0, M_1, ..., M_p\}$. At any particular time, at most one task can access the same device.

Each peripheral device can be in one of the two states: *active* or *sleep*. When the task is active, its corresponding associated peripheral device(s) must also be in active mode to provide timely service. The power consumption for the device M_i is denoted as P^i_{dact} and P^i_{dsleep} for its active mode and sleep mode, respectively. It will not be feasible or beneficial to shut down the processor or the devices if the time interval is not long enough. We use $L^i_{\min}$ to represent the minimal time interval during which the device M_i can be feasibly shut down with positive energy gain. For convenience, we also call $L^i_{\min}$ the *minimal shut-down interval* for device M_i. The general idea is: if we assume the energy and time overhead of powering-down/waking-up device M_i to be E^i_o and t^i_o respectively, then the device can be shut down with positive energy gain only if the length of the idle interval is larger than $L^i_{\min}$, which can be computed as

$$L^i_{\min} = \max\left\{\frac{E^i_o - P^i_{dsleep} \times t^i_o}{P^i_{dact} - P^i_{dsleep}}, t^i_o\right\}. \quad (3)$$

The minimal shut-down interval for the core processor can be computed in a similar way and we use $T_{\min}$ to represent it.

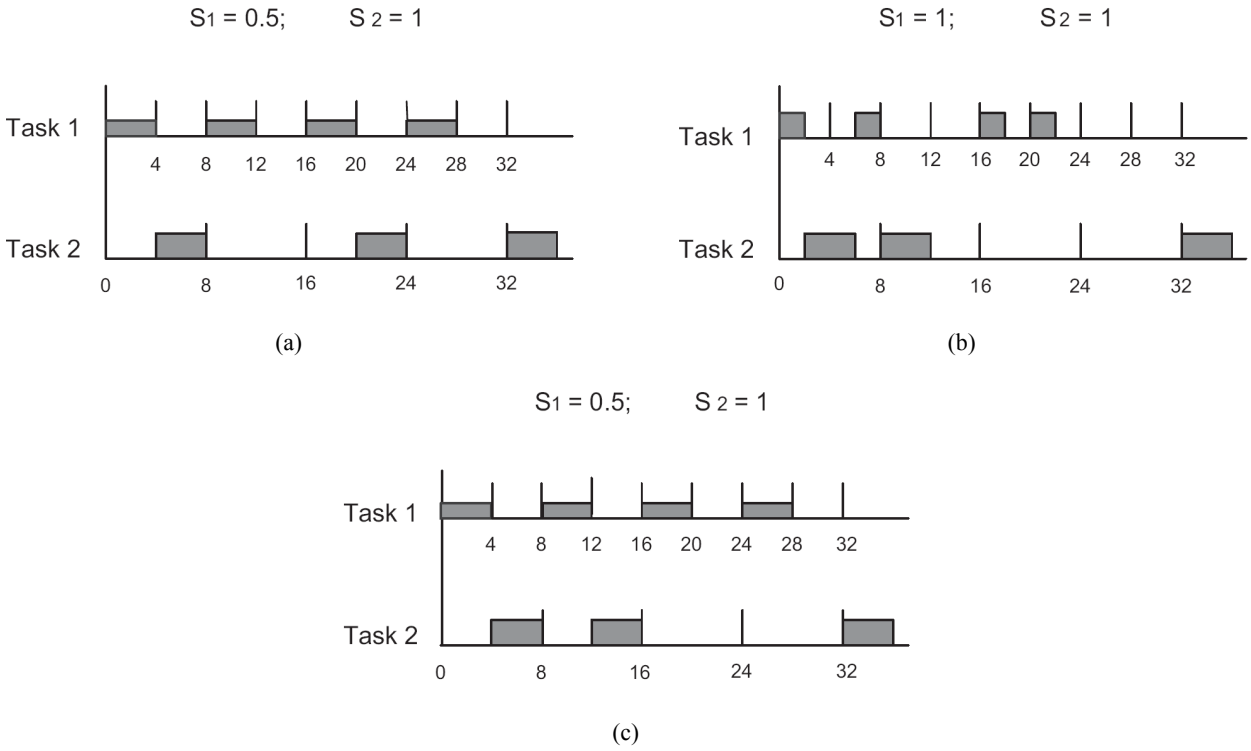
3.2 The motivations

Our goal is to employ DVS and DPD judiciously to save energy and guarantee the (m, k)-constraints in the mean time. Note that for weakly hard real-time systems with (m, k) requirements, the mandatory/optional partition has great impact on the processor/device power consumption. For example, the E-pattern helps to distribute the mandatory jobs evenly for a task. The work by Niu and Quan (2006a) shows that significant energy can be saved for tasks with E-patterns than that with R-patterns, since the processor speed can be reduced further without violating the deadlines. On the other hand, however, E-patterns distribute the mandatory jobs evenly, which leads to short scattered idle intervals that is not in favour of shutting down the devices. Consider a task set of two tasks, i.e., τ_1 = (4, 4, 2, 2, 4) and τ_2 = (8, 8, 4, 2, 4). Suppose the device shut down intervals $L^1_{\min} = 6$ and $L^2_{\min} = 16$ and the power consumption for the devices $P^1_{dact} = 0.2$ and $P^2_{dact} = 0.5$. Figure 1(a) shows the EDF schedule based on E-pattern. Note that in Figure 1(a) the speed of task τ_1 can be scaled efficiently but the devices for both tasks cannot be shut down.

R-pattern, on the other hand, seems to be a better choice in this scenario because it congregates the mandatory jobs and makes longer and fewer idle intervals possible. However, owing to the poor schedulability of R-pattern, the processor speed cannot be effectively scaled down. As shown in Figure 1(b), τ_1 has to be executed at a much higher processor speed (represented by the height of the rectangles) than that in Figure 1(a).

Figure 1(c) presents a schedule that can achieve the dual goal of scaling down the processor speed for task τ_1 and shut down the peripheral device for task τ_2 simultaneously. A careful study of Figure 1(c) would reveal that such solution is obtained by employing a combination of the E-pattern and R-pattern, i.e., partitioning τ_1 with E-pattern and τ_2 with R-pattern. In this way, we can effectively scale down the processor speed and also maintain long idle interval to shut down devices with high power consumption (i.e., device 2). As a result, the total power consumption within LCM can be greatly reduced, i.e., by 31% when compared with that by E-pattern in Figure 1(a) and 18% when compared with that by R-pattern in Figure 1(b).

Figure 1 (a) Executing the mandatory jobs of task set (τ_1 = (4, 4, 2, 2, 4); τ_2 = (8, 8, 4, 2, 4);) according to their *E*-patterns (b) Executing the mandatory jobs of the same task set according to their *R*-patterns (c) Executing the mandatory jobs of the same task set according to their hyb-patterns



4 The hybrid partitioning strategy

The motivation example implies that different partitioning strategies may have profound impacts on the energy savings. Unlike previous work that adopts either E-pattern or R-pattern alone, we intend to adopt a hybrid partitioning strategy, using both E-pattern and R-pattern simultaneously for the same task set. Two immediate problems follow:

1 how to ensure the schedulability of a task set with mixed E-pattern and R-pattern

2 how to assign the appropriate E-pattern or R-pattern to each task. In what follows, we address these two problems separately.

4.1 The feasibility condition

One of the key problems in our approach is the capability to predicate the schedulabilty of a task set with designated mandatory/optional pattern assignment. The following theorem provides us a practical way to predict the schedulability for the resulting mandatory job set.

Before we introduce this theorem, the following notation, i.e., $\lceil x \rceil^+$, helps us to formulate the problem and present the proof.

$$\lceil x \rceil^+ = 1 + \lfloor x \rfloor \tag{4}$$

With the definition of $\lceil x \rceil^+$, the following theorem allows one to predict the schedulability for a mandatory job set by checking only a limited number of time points (the proof is provided in Niu and Quan, 2013).

Theorem 1: Given system $\mathcal{T}$, let $\mathcal{R}$ and $\mathcal{E}$ be the subsets of $\mathcal{T}$ that are partitioned according to the R-pattern and E-pattern, respectively. Also, let

$$W_i^R(0,t) = \left(m_i \left\lfloor \frac{\left\lceil \frac{t-D_i}{T_i} \right\rceil^+}{k_i} \right\rfloor + \min\left\{ \left(\left\lceil \frac{t-D_i}{T_i} \right\rceil^+ - m_i \left\lfloor \frac{\left\lceil \frac{t-D_i}{T_i} \right\rceil^+}{k_i} \right\rfloor \right), m_i \right\} \right) C_i; \tag{5}$$

if task $\tau_i \in \mathcal{R}$ and

$$W_i^E(0,t) = \left(\left\lceil \frac{m_i}{k_i} \left\lceil \frac{t-D_i}{T_i} \right\rceil^+ \right\rceil \right) C_i, \tag{6}$$

if task $\tau_i \in \mathcal{E}$.

Then $\mathcal{T}$ is schedulable if all the mandatory jobs arriving within [0, L] can meet their deadlines, i.e.,

$$\sum_i^{\mathcal{R}} W_i^R(0,t) + \sum_i^{\mathcal{E}} W_i^E(0,t) \le t \tag{7}$$

for all $t \le L$ where L is either the ending point of the first busy period (Niu and Quan, 2006a) or the least common multiple of T_i, $i = 0, ..., (n-1)$, whichever is smaller, and

$$t = \begin{cases} pT_i + D_i, & p \in Z, p \bmod k_i \le m_i, \forall \tau_i \in \mathcal{R} \\ \left\lfloor q \frac{k_i}{m_i} \right\rfloor T_i + D_i, & q \in Z, \forall \tau_i \in \mathcal{E}. \end{cases} \tag{8}$$

Theorem 1 indicates that the schedulability of the mandatory jobs can be guaranteed if the mandatory jobs within the first busy interval or the LCM of the periods can meet their deadlines. The detailed proof of it is provided in Niu and Quan (2013).

Algorithm 1 The hybrid pattern assignment (algorithm PA_{HYB})

```
1:  Input: T, s_crit and L_min^i for τ_i;
2:  E = T, R = Ø;
3:  Update = TRUE;
4:  while Update do
5:      Update = FALSE;
6:      E' = {τ_i | s_crit(τ_i) ≥ 1, τ_i ∈ E};
7:      if E' ≠ Ø then
8:          Let τ' ∈ E' such that s_crit(τ') is the largest;
9:          if E − τ' schedulable then
10:             E = E − τ';
11:             R = R − τ';
12:             Update = TRUE;
13:         end if
14:     else
15:         for τ_i ∈ E do
16:             Let E_r(τ_i) (E_e(τ_i)) represent the energy
                consumption on τ_i within one k_i window
                according to R-pattern (E-pattern) assignment;
17:             if E_r > E_e AND E − τ_i is schedulable then
18:                 E = E − τ';
19:                 R = R − τ';
20:                 Update = TRUE;
21:             end if
22:         end for
23:     end if
24: end while
```

4.2 The pattern assignment

The problem then becomes how to assign R-patterns and E-patterns for different tasks appropriately to balance the processor and device power in order to save the overall energy. Based on Theorem 1, we present a heuristic for E/R-pattern assignment that incorporates such factors as the relative power consumption of the device and processor, the length of the minimal idle intervals, and task attributes among others.

Several observations help to develop our heuristic. Considering a job with workload w, the total energy

($E_{total}(s)$) consumed to finish this job with speed s can be represented as

$$E_{total}(s) = \left(P_{cact}(s) + P_{dact}\right) \times \frac{w}{s}. \quad (9)$$

Hence, the speed (*scrit)*that can minimise *Etotal*(s) in equation (9), so called *the critical speed* (Jejurika and Gupta, 2004a; Zhuo and Chakrabarti, 2005), can balance the processor and device power and minimise the overall energy consumption. Note that E-patterns tend to fragment the idle intervals for the devices. Therefore, when devices have long minimal idle intervals, it is desirable to choose R-pattern than E-pattern. Based upon the above observations and Theorem 1, we propose the pattern assignment algorithm in Algorithm 1.

Algorithm 1 works as follows: all tasks are initially assigned as E-patterns since E-pattern has better schedulability that R-pattern and has more potential to scale down the task speed. However, when the critical speed of task is larger than the maximal available speed on the processor, both E-pattern and R-pattern should not scale down the tasks speed below the maximal processor speed. At the same time, R-pattern tends to provide more opportunities to shut-down the device as it can provide longer idle intervals. So the assignment for a task will be updated when:

1 its critical speed is higher than 1

2 it cannot be shut down even in the longest possible idle interval with E-pattern but could be done so with R-pattern.

The algorithm terminates if no pattern assignment is updated.

After mandatory/optional patterns are assigned to each task, we can then scale down the processor speed for each task to their critical speed as much as possible to save energy. This can be done by exploiting Theorem 1 together with branch and bound scheme similar to that in Niu and Quan (2006a).

5 The dynamic scheduling algorithm

After performing the mandatory/optional job partitioning according to Algorithm 1 and computing the appropriate scaling factor based on Theorem 1, we are able to schedule the given task set and guarantee the (m, k)-constraints. Our dynamic scheduling algorithm consists of two parts: dynamic preemption control and dynamic pattern adjustment.

5.1 Dynamic preemption control

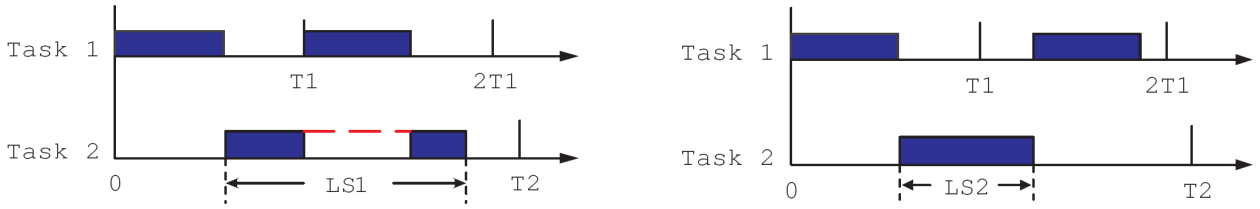
Since the peripheral devices do not have DVS capabilities, the most efficient way to save power is simply shut down the devices during its idle intervals. Moreover, when there are multiple tasks being executed in the system, the preemption effects among tasks will be pervasive. According to the experiment report in Kim and Min (2003), the number of task preemptions can grow up to 500% under DVS over non-DVS executions, which can have negative impact on the system-wide energy consumption in several ways (Kim and Roy, 2004): first, the preemption overhead may increase the energy consumption in memory subsystems; second, the lengthened *lifespan* of the preempted task may increase the energy consumption in the peripherals associated to it; third, as the number of active tasks increases, the number of active peripheral devices is likely to increase, consuming more energy.

In order to address the negative impact of preemptions on system-wide energy consumption, one effective way is to adopt the so called 'preemption control' technique. The main idea is: if the lower priority job can be finished earlier before it is preempted by the upcoming higher priority jobs, the lifespan of the lower priority job can be shortened. This is shown by the example in Figure 2. As shown in Figure 2(b), by temporarily withholding the executions of the upcoming higher priority job from task 1, the chances for the lower priority job from task 2 to be preempted by task 1 could be greatly reduced. Therefore, the lifespan of task 2 during which the devices associated to it must be in active mode could be shortened significantly. On the other hand, the shut-down interval of the devices associated with task 1 before the dispatch time of the second job of task 1 could also be extended effectively. As a result, the energy consumption of both the devices associated with higher priority jobs and those associated with lower priority jobs could be reduced.

In Kim and Roy (2004) and Zhuo and Chakrabarti (2005), preemption control techniques are proposed to reduce the chance for the current task to be preempted by higher priority jobs. However, their approach needs to increase the processing speeds of the jobs, which would increase the processor energy consumption and therefore might not necessarily be energy efficient. In Buttazzo et al. (2013), Bertogna and Baruah, (2010) and Bambagini et al. (2013), the limited preemptive scheduling framework based on 'blocking tolerance' was adopted to reduce the preemption among tasks. Their approaches need to handle the 'blocking' among tasks (based on blocking protocols) at each task arrival, which could incur significant overall time/energy cost. Moreover, it is not clear whether their approach could merge the shut-down intervals of peripheral devices.

In what follows, we adopt a novel strategy to reduce the preemptions by procrastinating the execution of higher priority jobs. Different from the approach in Kim and Roy (2004) and Zhuo and Chakrabarti (2005), we do not need to increase the processor speed and therefore have a better energy saving potential.

Figure 2 Reducing the lifespan of task 2 with preemption control, (a) task set schedule without procrastinating task 1 generated long lifespan (LS_1) for task 2 (b) procrastinating task 1 shortened the lifespan (LS_2) for task 2 greatly (see online version for colours)



Before introducing our strategy, we first introduce the following definition.

Definition 1 (Niu and Quan, 2006b): Assume that $\mathcal{M}$ is the mandatory job sets from $\mathcal{T}$ according to the hybrid pattern and can be schedulable, and let R_i be the worst case response time of τ_i. The delay factor for τ_i (denoted as Y_i) is defined as $Y_i = (D_i - R_i)$.

The worst case response time R_i can be computed in a similar way to that in Spuri (1996), which can be computed offline. With the definition of delay factor *Yi*, we have the following theorem (the proof is provided in the Appendix).

Theorem 2: Let $\mathcal{T}_m$ be the mandatory job set such that no more than mi mandatory jobs are assigned for any k_i consecutive jobs from $\tau_i \in \mathcal{T}$. Also let $hp(J_i)$ be the set of jobs in which each job J_p has arrival time $r_p > r_i$ and priority higher than J_i. All jobs in $\mathcal{T}_m$ can meet their deadlines if the starting execution time of $hp(J_i)$ is delayed to t_{np}, where

$$t_{np} = \min_{J_p \in hp(J_i)} \left(r_p + Y_p\right). \tag{10}$$

Theorem 2 allows us to delay the higher priority jobs safely. Although it is similar to Theorem 2 in Niu and Quan (2006b) in a sense that they all try to find the maximal delay of the future jobs, the major difference here is that in Theorem 2 in Niu and Quan (2006b), the higher priority jobs are delayed only when the processor is idle to get longer idle intervals. Here we delay the higher priority jobs once the current job gets chance to be executed or the processor begins to idle because we want to achieve the dual goals of reducing the number of preemptions and extending the idle intervals, as well as shortening the lifespan of the task and shutting down the processor/devices whenever possible to do so. Note that the delay factor Y_p can be computed offline and applied online. More importantly, the processing speeds for the higher priority jobs do not have to be increased even if they are delayed.

5.2 Dynamic (m, k)-pattern adjustment

Although the static analysis based on a predetermined hybrid pattern helps to ensure the feasibility of the mandatory job sets and thus guarantees the QoS levels, it is usually performed based on the worst case scenario and rather pessimistic. Considering the large run-time variations in embedded systems, it would be extremely profitable to employ a scheduling technique that can exploit the irregularities and variations online. We are therefore interested in developing a dynamic scheduling technique to achieve better energy-saving performance.

Niu and Quan (2006a) proposed a strategy to change the mandatory job dynamically. The rationale is that, if some optional job can be finished at a lower processor speed, other mandatory jobs can be demoted to optional and save energy. While this technique is proposed for all tasks to be assigned with E-pattern, we can prove that after modification it is still applicable to our case when different tasks may be assigned different patterns. In addition, with the incorporation of peripheral devices, more issues need to be considered in choosing the optional jobs to dispatch and in determining their running speed.

When no mandatory job is available to be executed, some optional jobs can also meet their deadlines with low speed. That give us more opportunities to reduce the energy further by adjusting the pattern dynamically. In the dynamic pattern adjustment, two job ready queues, i.e., the mandatory job queue (MJQ) and the optional job queue (OJQ) will be maintained. Upon arrival, a job, i.e., $J_a \in \tau_i$ is designated as mandatory job or optional job based on its predetermined E-pattern or R-pattern and inserted to the MJQ or OJQ correspondingly. The jobs in MJQ always have higher priority than those in OJQ. If the MJQ is empty, then the jobs in OJQ will have chance to be executed. It is not difficult to see that there may be more than one optional jobs in OJQ, and selecting which one to execute may have profound impacts on the future job executions. While the optional jobs can be selected arbitrarily without causing any dynamic failure, we use a more delicate heuristic to achieve better energy saving performance. Specifically, when the processor is idle, we first check whether there are some optional jobs in OJQ that can be finished with their critical speed before $\min_i(r_i + Y_i)$ of the upcoming mandatory jobs. Note that the inspected optional job(s) should be chosen only when the associated device(s) cannot be shut down during this idle interval. At the same time, the qualified optional job should not finish earlier than the earliest arrival time of the upcoming mandatory jobs, either, because if it cannot consume the idle interval completely it will generate new scattered idle intervals which cannot be shut down. After that, the qualified optional job in OJQ that has the lowest energy cost will be chosen to be executed.

If one optional job in OJQ is chosen to be executed and finished by its deadline, the (m, k)-pattern for the task it

belongs to will be adjusted correspondingly by shifting the pattern to the right by one position. The details are presented in Algorithm 2.

As shown, Algorithm 2 combines both the dynamic mandatory job pattern adjustment and dynamic preemption control and therefore can achieve much better energy performance, which will be demonstrated using experimental results in Section 6. Moreover, to ensure the effectiveness and efficiency of this algorithm, we have the following theorem (the proof is provided in Niu and Quan, 2013).

Theorem 3: Algorithm 2, with complexity of $O(n)$, can ensure the (m, k)-requirements for $\mathcal{T}$ if $\mathcal{T}$ is schedulable under the hybrid patterns assigned according to Algorithm 1.

Algorithm 2 The peripheral conscious dynamic scheduling algorithm (algorithm MK_{PC})

```
1:  Input: The MJQ, OJQ, and the current time t_cur;
2:  if MJQ is not empty then
3:      choose the job in MJQ that has the highest priority
        according to EDF as the current job J_i;
4:  else if OJQ is not empty then
5:      choose the qualified job in OJQ that has the lowest
        energy cost as the current job J_i;
6:  end if
7:
8:  Compute t_np for the current job J_i based on equation (10);
9:  Execute J_i non-preemptively within [t_cur, t_np];
10: Update t_cur;
11: if J_cur is completed then
12:     Let t_a be the arrival time of the next coming
        mandatory job from the same task;
13:     if (t_a − t_cur) > L^i_min then
14:         Shut down the device L_i and set up the wake up
            timer to be (t_a − t_cur);
15:     end if
16:     if J_i is optional job then
17:         Shift the pattern correspondingly;
18:     end if
19: end if
```

6 Experimental results

In this section, we evaluate the performance of our approach using simulations. We used two groups of real-time task sets as test cases in our experiments, one was randomly generated and the other one was drawn from practical applications. Specifically, we implemented and compared the energy saving performance of the following approaches

- PC_R*:* The task sets are statically partitioned with R-patterns, and the mandatory jobs are executed with the statically determined speed. We use its results as the reference results.
- PC_E*:* The mandatory/optional jobs are partitioned based on E-patterns.
- PC_{HYB}*:* The static hybrid patterns proposed in Section 4.2 are adopted in partitioning the mandatory/optional jobs.
- $PC_{HYB\text{-}p}$*:* Based on the static hybrid patterns proposed in Section 4.2, we try to delay the mandatory jobs to facilitate dynamic preemption control and shut-down.
- $PC_{HYB\text{-}dyn}$*:* Based on the static hybrid patterns proposed in Section 4.2, dynamic pattern adjustment are adopted during the online phase.
- $PC_{HYB\text{-}dyn\text{-}p}$ Based on the static hybrid patterns proposed in Section 4.2, we combine the dynamic preemption control and pattern adjustment as shown in Section 5.

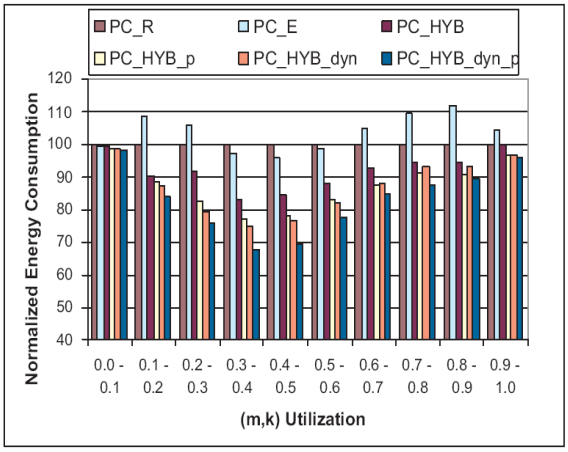
Besides the above approaches, we also studied the following approach that can exploit reducing the preemptions between the tasks to saving the system energy:

- $lppc_{DP}$*:* This approach was proposed by Kim and Roy (2004) which takes the preemptions between tasks into considerations. It tried to use some preemption control techniques to shorten the life time of the tasks and thus to reduce the period during which the peripheral devices must stay in active mode. Here we incorporate this approach into our static hybrid partitioning strategy.

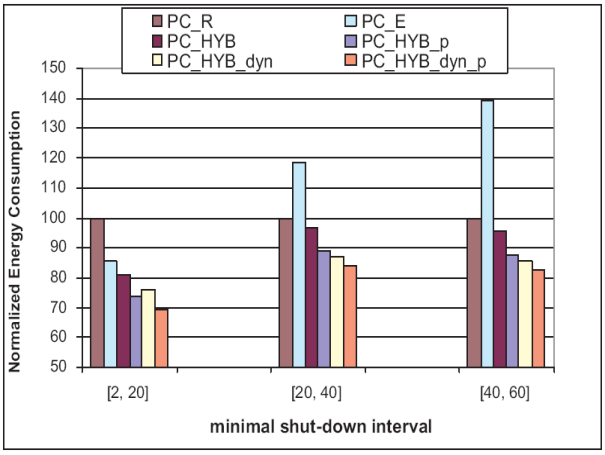
6.1 Experimental results for the synthesised task sets

Four sets of experiments were conducted in this case. In the first set of experiments, we study the energy-saving performance by different approaches corresponding to different workloads. We randomly generated periodic task set with five tasks. The periods were randomly chosen in the range of [5, 50] ms. The worst case execution time (WCET) of a task was set to be uniformly distributed from 1 ms to its deadline, and the actual execution time of a job was randomly picked from [0.4WCET, WCET]. The m_i and k_i for the (m, k)-constraints were also randomly generated such that k_i is uniformly distributed between 4 to 10, and $2 \le m_i < k_i$. We varied the (m, k)-utilisation, i.e., $\sum_i \frac{m_i C_i}{k_i T_i}$, of the task set by steps of 0.1, and generated at least 20 schedulable task sets within each interval or until at least 5,000 task sets have been generated. The devices associated with each task were randomly chosen from three types of devices: M_1 = (0.5, 5), M_2 = (1, 15), and M_3 = (5, 30), where device type M_i is characterised by a pair of parameters $(P^i_{dact}, L^i_{\min})$, representing its relative power (compared with the processor) and minimal shut-down interval length (in ms). We assume that the processor minimal shut-down interval length $T_{\min}$ = 2 ms. The results are shown in Figure 3(a).

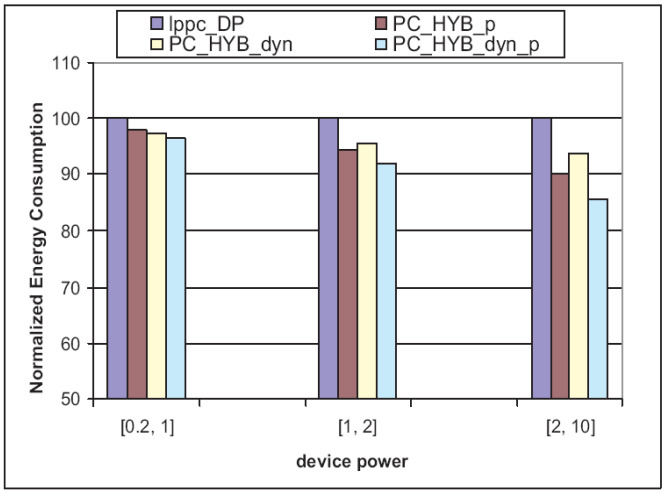
Figure 3 (a) The average total energy consumption by the different approaches (b) The energy comparison for different shut-down interval lengths (c) The energy comparison for different preemption control techniques (see online version for colours)



(a)



(b)



(c)

While it is shown (Niu and Quan, 2006a) that E-pattern assignment always dominates R-pattern assignment in reducing the processor energy, this is not necessarily true any more when peripheral devices are taken into consideration, as shown in Figure 3(a). This is mainly because R-pattern can provide longer maximal idle intervals than E-pattern and thus have more chances for the devices to be shut down. Also, one can immediately see from the results that, by adopting hybrid patterns, PC_{HYB} can achieve much better energy efficiency than those adopting E-pattern or R-pattern alone, i.e., up to around 18%. Moreover, by delaying the mandatory job, $PC_{HYB\text{-}p}$ can reduce the energy consumption further. On the other hand, the dynamic pattern adjustment by $PC_{HYB\text{-}dyn}$ can also improve the energy efficiency of PC_{HYB} obviously. It is also interesting to see that when the (m, k)-utilisation is relatively low, $PC_{HYB\text{-}dyn}$ dominated $PC_{HYB\text{-}p}$, while when the (m, k)-utilisation is relatively high, $PC_{HYB\text{-}p}$ dominated $PC_{HYB\text{-}dyn}$. This is because when the (m, k)-utilisation is relatively low, there is more space to execute the optional jobs and to adjust the patterns dynamically, while with the increase of the (m, k)-utilisation the space for dynamic pattern adjustment is becoming much less. The best energy efficiency is achieved when the two approaches are combined together. As can be seen from Figure 3(a), the combined dynamic algorithm, i.e., $PC_{HYB\text{-}dyn\text{-}p}$, can reduce the energy of PC_{HYB} further by up to 15%.

In the second set of experiments, we investigate the energy saving performance for devices with different minimal shut-down intervals. The powers of the devices remain the same. Three sub-sets of experiments were conducted with the minimal shut-down interval sets of the devices randomly selected from one of three ranges [2, 20] ms, [20, 40] ms, and [40, 60] ms, respectively. The results for task sets (generated in the same way as those for the first set) with (m, k)-utilisation falling into representative interval [0.3, 0.4] are shown in Figure 3(b).

As shown in Figure 3(b), when the minimal shut-down intervals are chosen from shorter interval range, i.e., [2, 20] ms, in most cases both E-pattern and R-pattern can help shut the peripheral devices. In this case E-pattern has better energy performance since E-pattern helps to better slow down the processor speed and thus reduce the overall energy. However, as the minimal shut-down interval

length grows, R-pattern becomes much better as it provides more chances for the devices to be shut down for the same reason as stated above, especially when the minimal shut-down interval becomes significantly large, i.e., [40, 60] ms in Figure 3(b). Note that in all three cases, using hybrid pattern (PC_{HYB}) can achieve the best energy performance among them. And similar to the results above, either dynamic preemption control ($PC_{HYB\text{-}p}$) or dynamic pattern adjustment ($PC_{HYB\text{-}dyn}$) alone can reduce the energy consumption further. And the lowest energy consumption is achieved by combining them together. Compared to PC_{HYB}, the combined approach $PC_{HYB\text{-}dyn\text{-}p}$ can reduce the energy further by around 15%.

The third set of experiments mainly evaluate the effectiveness of our technique on dynamic preemption control. Since the preemption control scheme by Kim and Roy (2004) can also be incorporated into our approach PC_{HYB}, we are interested in how much our approach can help improve the approach proposed in Kim and Roy (2004) (represented by $lppc_{DP}$). The task sets were generated in the same way as that for the second set. For the devices, we fixed their minimal shut-down intervals but vary their relative power consumption. Three sub-sets of tests were also conducted, within each we randomly selected the relative power consumption for devices from one of three power ranges, [0.2, 1], [1, 2], and [2, 10]. The results, normalised to that by $lppc_{DP}$, are shown in Figure 3(c).

As shown in Figure 3(c), when the device power is very small, the improvements of our approaches, i.e., $PC_{HYB\text{-}p}$ and $PC_{HYB\text{-}dyn}$, over $lppc_{DP}$ are limited as the critical speed of the task is much smaller than the maximal speed, which provides more space for $lppc_{DP}$ to adjust the job speed and delay the higher priority mandatory jobs. However, as the device power increases, the improvement of $PC_{HYB\text{-}p}$ and $PC_{HYB\text{-}dyn}$ becomes more significant.

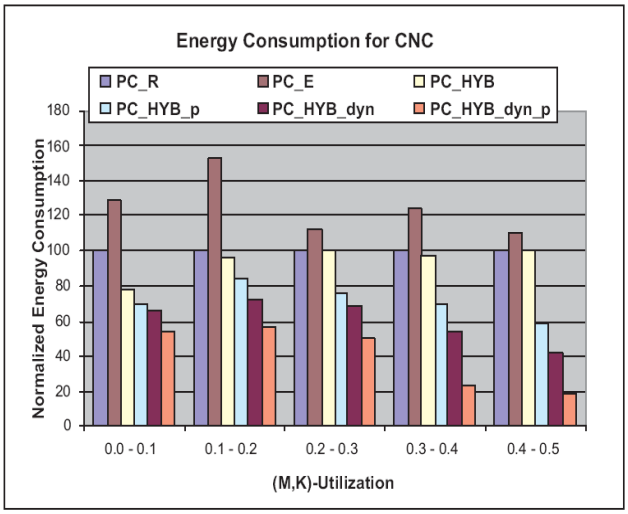
This is because when the device power becomes larger, the critical speed for each task becomes closer to or higher than the maximal processor speed, which makes little slack for $lppc_{DP}$ to adjust speeds for the higher priority mandatory jobs. And as expected, when $PC_{HYB\text{-}p}$ and $PC_{HYB\text{-}dyn}$ are combined together, the energy saving is even much better. For example, when the device power is larger than twice the processor power, the improvement of $PC_{HYB\text{-}dyn\text{-}p}$ over $lppc_{DP}$ can be around 15% as shown in the figure.

6.2 Experimental results from real applications

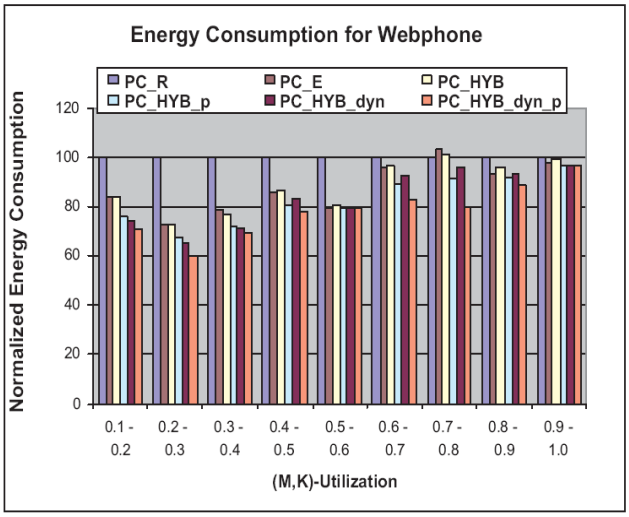
Next, we tested our approach in a more practical environment. Instead of random examples, we generated the test cases with specifications drawn from three real world applications: computerised numerical control (CNC) machine controller (Kim et al., 1996), inertial navigation system (INS) (Burns et al., 1995), and Webphone (Shin et al., 2001). The timing parameters such as the deadlines, periods, and execution times were adopted from these practical applications directly. The actual execution time of a job was randomly picked from [0.4WCET, WCET] and the (m, k)-constraints were generated as we did for the synthesised task sets. Similarly, we divided the total (m, k)-utilisation into intervals of length 0.1. Within each interval, we generated at least 20 task sets that were schedulable with the R-pattern, or until at least 5,000 task sets had been generated. The total energy consumption of the system for each approach in each test were collected. All the results were normalised to those by *PCR* and shown in Figure 4.

The experimental results based on the practical applications further demonstrate the effectiveness of our proposed approaches based on hybrid pattern (i.e., PC_{HYB}) and dynamic preemption control and dynamic pattern adjustment (i.e., $PC_{HYB\text{-}p}$ and $PC_{HYB\text{-}dyn}$). From the experiment results in Figure 4, it is interesting to see that different static pattern assignments can have different impacts on the energy consumption of different applications. For example, for the CNC [Figure 4(a)] and INS application [Figure 4(c)], it seems that most of the time R-pattern has better energy performance than E-Pattern because the shut down interval lengths of the peripheral devices are relatively large when compared to the periods of the tasks. However, for the Webphone application [Figure 4(b)], E-pattern has better energy performance over R-pattern for most of the intervals because E-pattern can scale down the processor speeds more efficiently than R-pattern. Also, it is interesting to see that for all three applications the static approach based on hybrid pattern (i.e., PC_{HYB}) is always better or close to the lowest one among them. The maximal improvement by the hybrid pattern can be nearly 40% over E-pattern and up to 25% over R-pattern. More importantly, adopting dynamic preemption control and pattern adjustment techniques can further improve the energy reduction greatly. For CNC, as we can see from Figure 4(a), the energy reduction of $PC_{HYB\text{-}p}$ over PC_{HYB} can be up to 40% because it can reduce the preemptions between tasks and thus shorten the lifespan of the tasks and reduce the energy of the peripheral devices significantly. On the other hand, the energy reduction of $PC_{HYB\text{-}dyn}$ over PC_{HYB} can be up to 58% because by adjusting the pattern of the tasks dynamically, the jobs are partitioned into mandatory/optional jobs more adaptively during the run-time, which could help reduce the total energy consumption efficiently. The maximal energy reduction is achieved when these two techniques are integrated together. As can be observed in Figure 4(a), compared to PC_{HYB}, the energy reduction by $PC_{HYB\text{-}dyn\text{-}p}$ can be up to 80% on CNC. Similarly, for Webphone and INS, the energy reduction of $PC_{HYB\text{-}dyn\text{-}p}$ over PC_{HYB} can be up to 15% and 60%, respectively, as shown in Figures 4(b) and 4(c). Also as we can see from these results, the energy savings differ from different applications. This is because the applications with relatively lower processor utilisation requirement, such as the CNC and INS, can have better energy saving potential in applying preemption-control and dynamic pattern adjustment, than the ones with much higher processor utilisation requirement such as Webphone.

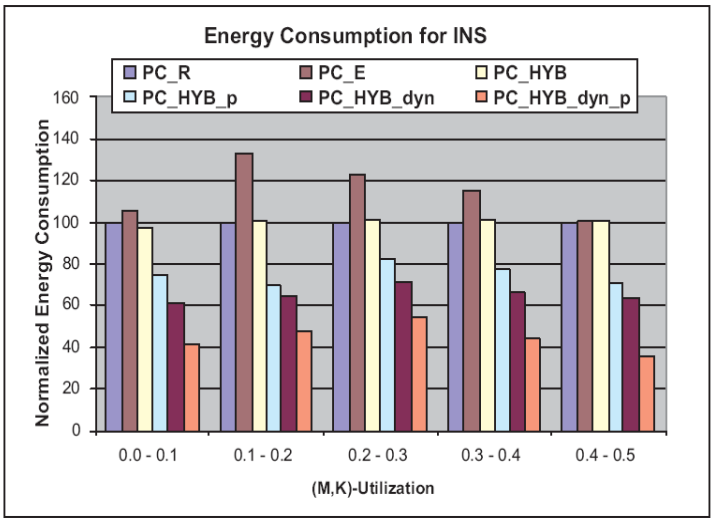
Figure 4 The energy comparison for the different approaches based on real word applications, (a) CNC (Kim et al., 1996) (b) Webphone (Shin et al., 2001) (c) INS (Burns et al., 1995) (see online version for colours)



(a)



(b)



(c)

In summary, the experimental results based on both the synthesised systems as well as the practical applications have shown that our proposed approaches can achieve significantly better energy savings with guaranteed QoS levels when compared with the conventional ones.

7 Summary

In this paper, we present a dynamic scheduling algorithm to minimise the system wide energy consumption with (m, k)-guarantee. The system consists of a core processor and a number of peripheral devices, which can have different power characteristics. Different from previous work that adopted single known mandatory/optional partitioning strategy, we proposed to incorporate different partitioning strategies based on the power characteristics of the devices as well as the application specifications. We introduced theorems that can predict the feasibility of such a strategy, and based on which, we proposed an algorithm to perform the mandatory/optional job partitions dynamically. We also proposed novel preemption control schemes, which can be well incorporated into our dynamic scheduling algorithm. Extensive experiments based on both synthesised task sets and real world applications have been performed to demonstrate the effectiveness of our approach.

Acknowledgements

This work is supported in part by NSF under projects CNS-0969013, CNS-0917021, and CNS-1018108.

References

AlEnawy, T.A. and Aydin, H. (2005) 'Energy-constrained scheduling for weakly-hard real-time systems', *RTSS*.

Awan, M.A. and Petters, S.M. (2011) '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*, *ECRTS '11*, pp.92–101, IEEE Computer Society, Washington, DC, USA.

Aydin, H., Melhem, R., Mosse, D. and Alvarez, P. (2001) 'Determining optimal processor speeds for periodic real-time tasks with different power characteristics', in *ECRTS01*, June.

Bambagini, M., Bertogna, M., Marinoni, M. and Buttazzo, G.C. (2013) 'An energy-aware algorithm exploiting limited preemptive scheduling under fixed priorities', in *SIES*.

Bernat, G. and Burns, A. (1997) 'Combining (*n*, *m*)-hard deadlines and dual priority scheduling', in *RTSS*, December.

Bernat, G. and Cayssials, R. (2001) 'Guaranteed on-line weakly-hard real-time systems', in *RTSS*.

Bertogna, M. and Baruah, S. (2010) 'Limited preemption EDF scheduling of sporadic task systems', *Industrial Informatics*, *IEEE Transactions on*, Vol. 6, No. 4, pp.579–591.

Burns, A., Tindell, K. and Wellings, A. (1995) 'Effective analysis for engineering real-time fixed priority schedulers', *IEEE Transactions on Software Engineering*, May, Vol. 21, No. 5, pp.920–934.

Buttazzo, G., Bertogna, M. and Yao, G. (2013) 'Limited preemptive scheduling for real-time systems. A survey', *Industrial Informatics*, *IEEE Transactions on*, Vol. 9, No. 1, pp.3–15.

Chen, J-J. and Kuo, T-W. (2006) 'Procrastination for leakage-aware rate-monotonic scheduling on a dynamic voltage scaling processor', *SIG-PLAN Notices*, Vol. 7, pp.153–162.

Cheng, H. and Goddard, S. (2006) 'Online energy-aware I/O device scheduling for hard real-time systems', *DATE*.

Davis, R. and Wellings, A. (1995) 'Dual priority scheduling', in *RTSS*.

Devadas, V. and Aydin, H. (2010) 'DFR-EDF: a unified energy management framework for real-time systems', *Real-Time and Embedded Technology and Applications Symposium*, *IEEE*, pp.121–130.

Devadas, V. and Aydin, H. (2012) 'On the interplay of voltage/frequency scaling and device power management for frame-based real-time embedded applications', *IEEE Transactions on Computers*, Vol. 61, No. 1, pp.31–44.

Doherty, L., Warneke, B., Boser, B. and Pister, K. (2001) 'Energy and performance considerations for smart dust', *International Journal of Parallel Distributed Systems and Networks*, Vol. 4, No. 3, pp.121–133.

Hamdaoui, M. and Ramanathan, P. (1995) 'A dynamic priority assignment technique for streams with (*m*, *k*)-firm deadlines', *IEEE Transactions on Computes*, December, Vol. 44, No. 12, pp.1443–1451.

Hua, S. and Qu, G. (2004) 'Energy-efficient dual-voltage soft real-time system with (*m*, *k*)-firm deadline guarantee', in *CASE '04*.

Jejurikar, R. and Gupta, R. (2004a) 'Dynamic voltage scaling for system-wide energy minimization in real-time embedded systems', *ISLPED*.

Jejurikar, R. and Gupta, R. (2004b) 'Procrastination scheduling in fixed priority real-time systems', *LCTES*.

Jejurikar, R., Pereira, C. and Gupta, R. (2004) 'Leakage aware dynamic voltage scaling for real-time embedded systems', *DAC*.

Jejurikar, R., Pereira, C. and Gupta, R. (2005) 'Dynamic slack reclamation with procrastination scheduling in realtime embedded systems', *DAC*.

Ji, Y.C., Zhang, G-A., Zhang, X-G. and Huang, X. (2013) 'Outage-optimal power allocation for space-time cooperative network coding with amplify-and-forward protocol', *International Journal of Embedded Systems*, Vol. 5, No. 3, pp.127–133.

Kathuria, J. (2013) 'Low power clock gating techniques for synchronous buffer-based queue for 3D MPSOC', *International Journal of Embedded Systems*, Vol. 5, Nos. 1/2, pp.36–43.

Kim, C. and Roy, K. (2004) 'Preemption-aware dynamic voltage scaling in hard real-time systems', *ISLPED*.

Kim, J.K.W. and Min, S.L. (2003) 'Quantitative analysis of dynamic voltage scaling algorithms for hard real-time systems', *Proceedings of the SoC Design Conference*, November.

Kim, M. and Ha, S. (2001) 'Hybrid run-time power management technique for real-time embedded system with voltage scalable processor', *OM '01*, pp.11–19.

Kim, N., Ryu, M., Hong, S., Saksena, M., Choi, C. and Shin, H. (1996) 'Visual assessment of a real-time system design: a case study on a CNC controller', in *RTSS*, December.

Kim, W., Kim, J. and Min, S.L. (2002) 'A dynamic voltage scaling algorithm for dynamic-priority hard real-time systems using slack analysis', *DATE*.

Kong, F., Deng, Q. and Yi, W. (2011) 'Energy-efficient scheduling of real-time tasks on cluster-based multicores', in *DATE*.

Kong, F., Wang, Y., Deng, Q. and Yi, W. (2010) 'Minimizing multi-resource energy for real-time systems with discrete operation modes', in *Proceedings of the 22nd Euromicro Conference on Real-Time Systems*, *ECRTS '10*, pp.113–122, IEEE Computer Society, Washington, DC, USA.

Koren, G. and Shasha, D. (1995) 'Skip-over: algorithms and complexity for overloaded systems that allow skips', in *RTSS*.

Lee, Y., Reddy, K. and Krishna, C. (2003) 'Scheduling techniques for reducing leakage power in hard real-time systems', *ECRTS*.

Niu, L. (2011) 'Energy efficient scheduling for real-time systems with QoS guarantee', *Journal of Real-Time System*, Vol. 47, No. 2, pp.75–108.

Niu, L. and Quan, G. (2006a) 'Energy minimization for real-time systems with (*m*, *k*)-guarantee', *IEEE Trans. on VLSI*, Special section on *Hardware/Software Codesign and System Synthesis*, pp.717–729, July.

Niu, L. and Quan, G. (2006b) 'System wide dynamic power management for weakly hard real-time systems', *ISM*.

Niu, L. and Quan, G. (2013) *Peripheral-conscious Energy-efficient Scheduling for Weakly Hard Real-time Systems*, in Technical Report TR-2013-1002.

Quan, G. and Hu, X. (2000) 'Enhanced fixed-priority scheduling with (*m*, *k*)-firm guarantee', in *RTSS*, pp.79–88.

Quan, G., Niu, L., Hu, X.S. and Mochocki, B. (2009) 'Real time scheduling for reducing overall energy on variable voltage processors', *International Journal of Embedded System: Special Issue on Low Power Embedded Computing*, Vol. 4, No. 2, pp.127–140.

Ramamritham, K. and Stankovic, J.A. (1994) 'Scheduling algorithms and operating system support for real-time systems', *Proceedings of the IEEE*, January, Vol. 82, No. 1, pp.55–67.

Ramanathan, P. (1999) 'Overload management in real-time control applications using (*m*, *k*)-firm guarantee', *IEEE Trans. on Paral. and Dist. Sys.*, June, Vol. 10, No. 6, pp.549–559.

Shin, D., Kim, J. and Lee, S. (2001) 'Intra-task voltage scheduling for low-energy hard real-time applications', *IEEE Design and Test of Computers*, March–April, Vol. 18, No. 2.

Spuri, M. (1996) *Analysis of Deadline Scheduled Real-Time Systems*, in Rapport de Recherche RR-2772, INRIA, France.

Viredaz, M.A. and Wallach, D.A. (2003) 'Power evaluation of a handheld computer', *IEEE Micro*, Vol. 23, No. 1, pp.66–74.

Wang, X., Du, Z., Xue, Y., Fan, L. and Wang, R. (2012) 'Self-adaptive QoS-aware resource allocation and reservation management in virtualised environments', *International Journal of Computational Science and Engineering*, October, Vol. 7, No. 4, pp.308–318.

Yang, C., Sheng, M., Li, J., Li, H. and Li, J. (2013) 'Energy-aware joint power and rate control in overlay cognitive radio networks: a Nash bargaining perspective', *International Journal of Embedded Systems*, Vol. 5, No. 3, pp.118–126.

Zedlewski, J., Sobti, S., Garg, N., Zheng, F., Krishnamurthy, A. and Wang, R. (2003) 'Modeling hard-disk power consumption', *FAST '03*, pp.217–230.

Zhao, B. and Aydin, H. (2009) 'Minimizing expected energy consumption through optimal integration of DVS and DPM', in *ICCAD*.

Zhuo, J. and Chakrabarti, C. (2005) 'System level energy efficient dynamic task scheduling', *DAC*.

Appendix

Proof for Theorem 2

To prove Theorem 2, we first need to prove the following lemma.

Lemma 1: Let $\mathcal{M}$ be the mandatory job set from T according to the hybrid pattern. Then if the processor starts *its execution at* $t_s = \min(Y_i)$, $i = 0, 1, ..., n-1$, *no mandatory job in* $\mathcal{M}$ *will miss its deadline.*

Proof: Use contradiction. Assume that when the processor starts its execution at t_s, some mandatory job $J_p = \{r_p, c_p, d_p\}$ misses its deadline, where r_p, c_p, and d_p represent the arrival time, execution time and absolute deadline of J_p. J_p must be in the first busy interval since the delay of the processor execution would not cause J_p to miss its deadline otherwise. Therefore

$$\sum_i W_i\left(0, d_p\right) > \left(d_p - t_s\right). \tag{11}$$

where $W_i(t_1, t_2)$ represents the work demand between interval $[t_1, t_2]$.

Assume J_p finishes at $f_p (r_p < f_p \le d_p)$ when the processor starts at $t = 0$. Let

- $\mathcal{J}(0, d_p)$ represent the mandatory jobs with deadlines no later than d_p
- $\mathcal{J}(0, d_f)$ represent the mandatory jobs arriving earlier than f_p with deadlines no later than d_p
- $\mathcal{J}(d_f, d_p)$ represent the mandatory jobs arriving *NO* earlier than f_p with deadlines no later than d_p.

It is easy to see that $\mathcal{J}(0, d_p) = \mathcal{J}(0, d_f) \cup \mathcal{J}(d_f, d_p)$. Let $W'(\mathcal{J})$ represent the workload, i.e., total execution time, of $\mathcal{J}$. Then we have

$$W'(\mathcal{J}) \tag{12}$$

and

$$W'\left(\mathcal{J}\left(0, f_p\right)\right) \le f_p. \tag{13}$$

Now consider job $J_q = \{r_q, c_q, d_q\} \in \mathcal{J}(d_f, dp)$ such that J_q finishes at f_q (the latest before dp) when the processor starts at $t = 0$. Then we have

$$W'\left(\mathcal{J}\left(f_p, d_p\right)\right) \le \left(d_p - f_p\right) - \left(d_q - f_q\right). \tag{14}$$

As $(d_q - f_q) \ge (D_q - R_q) = Y_q$ (Definition 1) and $Y_q \ge ts$, from equation (14), we have

$$W'\left(\mathcal{J}\left(f_p, d_p\right)\right) \le \left(d_p - f_p\right) - t_s. \tag{15}$$

Then from equations (12), (13) and (15), we have

$$\sum_i W_i\left(0, d_p\right) \le d_p - t_s, \tag{16}$$

which contradicts equation (11). □

Then we can proceed to prove Theorem 2 based on Lemma 1. Again we use contradiction. Given at time t_0, we have three possibilities:

1 the processor is idle

2 the processor is executing an optional job J_i

3 the processor is executing a mandatory job J_i.

Here we need to deal with three cases separately

- The processor is idle at t_0.

  Assume a mandatory job J_p misses its deadline when the processor resumes its execution at $t = t_0 + t_s$. Therefore we have

  $$\sum_i W_i\left(t_0, d_p\right) > d_p - t. \tag{17}$$

  Now consider $\mathcal{M}'$, the mandatory job set from T according to the hybrid pattern. Since $\mathcal{M}$ is schedulable when processor delays its execution to t_s, we have

  $$\sum_i W_i\left(0, d_p - t_0\right) < d_p - t_0 - t_s = d_p - t. \tag{18}$$

  In addition, for $\mathcal{M}$, there are no more than m_i jobs among any consecutive k_i jobs from τ_i are mandatory. Therefore, we have

  $$\sum_i W_i\left(0, d_p - t_0\right) \ge \sum_i W_i\left(t_0, d_p\right) > d_p - t, \tag{19}$$

which contradicts equation (18).

- The processor is executing an optional job J_i at t_0.

  In this case $hp(J_i)$ is the mandatory job set that arrive later than J_i. Since the optional job J_i cannot interfere the execution of all the mandatory jobs, the situation in this case is the same as the previous case.

- The processor is executing a mandatory job J_i at t_0.

  Assume a mandatory job J_p misses its deadline after the execution of $hp(J_i)$ is delay to $t = t_0 + t_s$, then J_p cannot be from $hp(J_i)$ for the same reason as stated above. J_p cannot be J_i either as delaying the execution of $hp(J_i)$ is in favour of the earlier completion of J_i when compared with the non-delay case. In other words, J_p can only be from some mandatory job with priority lower than J_i. In this case, the delay of the execution of $hp(J_i)$ to $t = t_0 + ts$ will not change $\sum_i W_i(t_0, d_p)$, i.e., the total work demand between $[t_0, t_p]$. Since J_p misses its deadline at d_p, we must have

$$\sum_i W_i\left(t_0, d_p\right) > d_p - t_0. \tag{20}$$

Consider $\mathcal{M}'$, the mandatory job set from $\mathcal{T}$ according to the hybrid pattern. Since $\mathcal{M}'$ is schedulable, we have

$$\sum_i W_i\left(0, d_p - t_0\right) < d_p - t_0. \tag{21}$$

Then similar as that in the above cases, we have

$$\sum_i W_i\left(t_0, d_p\right) \le \sum_i W_i\left(0, d_p - t_0\right) \le d_p - t_0. \tag{22}$$

which contradicts equation (20).