Peripheral-Conscious Scheduling on Energy Minimization for Weakly Hard Real-time Systems

Abstract

In this paper, we present a dynamic scheduling algorithm to minimize the energy consumption by both the DVS processor and peripheral devices in a weakly hard real-time system. In our approach, we first use a static approach to partition real-time jobs into mandatory and optional part to meet the weakly hard real-time constraints. We then adopt an on-line approach that can effectively exploit the run-time variations and reduce the preemption impacts to leverage the energy saving performance. Extensive simulation studies demonstrate that our approach can effectively reduce the system-wide energy consumption while guaranteeing the weakly hard constraints at the same time.

1 Introduction

Power aware scheduling has proven to be an effective way to reduce the energy consumption which is critical to increase the mobility for today's pervasive computing systems. 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. [12, 18]) 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 reveals that the processor core consumes around 28.8% of total power when playing a video file on a hardware testbed [16] for handheld devices, while the DRAM consumes about 28.4% of the total power. Note that this testbed [16] lacks disk storage and wireless networking capability, which may contribute as much power consumption as the processor core if not more [19, 3]. This implies that the techniques that attack the processor energy alone may not be overall energy efficient.

Recently, several techniques (e.g. [6]) 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 tolerate some deadline misses provided that user's perceived quality of service (QoS) constraints can be satisfied. The *weakly hard real-time model* is more accurate for practical applications. In the weakly-hard real-time model, tasks have both firm deadlines (i.e., deadline missing is useless) and a throughput requirement (i.e., *sufficient* task instances must meet deadlines to provide required quality levels).

Many weakly hard real-time models have been proposed (e.g. [14, 17]). Specifically, Ramanathan *et. al.* [14] 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 behavior 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.

In this paper, we study the problem of reducing the system-wide energy consumption for the weakly hard real-time system modeled with the $(m,k)-$model. The problem becomes more challenging since we need to deal with not only the tradeoffs between DVS and DPD (as most peripheral devices support only DPD mechanism), 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, which is known to be NP-hard [13]. In this paper, we propose a novel mandatory/optional partitioning strategy and a sufficient condition for checking the feasibility. Based on which, we present a dynamic scheduling scheme that extends previous approaches on preemption control [5] and mandatory job pattern adjustment [10] to achieve higher efficiency in energy savings. The novelty and effectiveness of this approach are demonstrated with extensive simulation studies.

The rest of the paper is organized as follows. Section 2

presents the system model, related work, and motivations. Section 3.1 describes a feasibility condition to guarantee the (m,k)-firm deadlines. Section 3 presents our new approach in determining the mandaory/optional job partitioning. Section 4 presents our overall algorithm to reduce the system energy. In section 5, we presents our experimental results. Section 6 draws the conclusions.

2 Preliminary

In this section, we first introduce the system and architecture model. We then survey the related work, followed by a motivation example.

2.1 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 characterized using five parameters, *i.e.*, $\tau_i = (T_i, D_i, C_i, m_i, k_i)$. T_i, D_i ($D_i \leq T_i$), and C_i represent the period, the 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.

The system architecture consists of a DVS processor and n devices, $M_0, M_1, ..., M_{n-1}$, each of which is dedicated to one different task. 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 S_i, which is normalized to the speed corresponding to V_{max}. We denote the processor power as P_{cact} when running a task at its maximal speed and P_{csleep} when it is shut down. We use three parameters to characterize a peripheral device, i.e., $M_i = (P^i_{dact}, P^i_{dsleep}, L^i_{min})$, where P^i_{dact} represents the active mode power consumption, P^i_{dsleep} represents the sleep mode power consumption, and L^i_{min} represents the minimal time interval that the device can be feasibly shut down with positive energy-saving gain. Similarly, we use T_{min} to represent the minimal time interval for the processor when it works at the highest speed.

2.2 Related work

Most DVS real-time scheduling approaches are focused on saving energy consumed by the processor only. Recently, a number of researches (e.g. [4, 6, 5, 20]) are reported to reduce the energy consumption for systems consisting of DVS processors and peripheral devices. Kim and Ha [6] proposed a time slot-based scheduling technique for *hard* real-time system. Jejurikar and Gupta [4] introduced a heuristic search method to find the so called *critical speed* to balance the energy consumption between the processor and peripheral devices. Kim *et al.* [5] and Zhuo *et al.* [20] considered controlling the preemption between tasks in order to reduce the active period of the devices and therefore their energy consumption. There are also a number of researches investigating the scheduling problem for systems with non-DVS processor and I/O devices [2]. All these approaches target hard real-time systems.

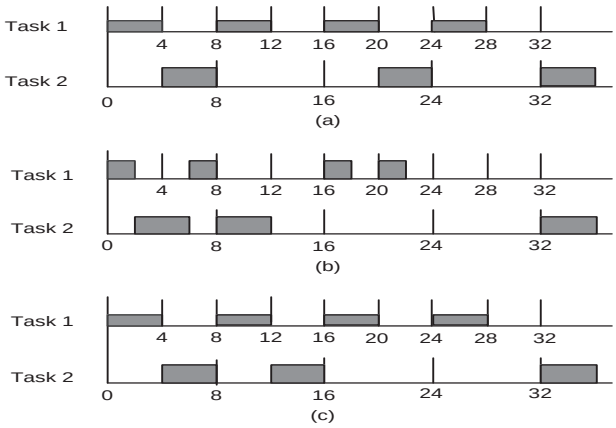


Figure 1. (a) Executing the mandatory jobs of task set ($\tau_1 = (4,4,2,2,4)$; $\tau_2 = (8,8,4,2,4)$;) according to their E-patterns; (b) Executing the mandatory jobs of the same task set according to their R-patterns; (c) Executing the mandatory jobs of the same task set according to their hyb-patterns.

We are more interested in developing scheduling techniques for real-time systems with (m,k)-constraints. The related mandatory/optional partitioning and scheduling problem, due to its NP-hard nature [13], add another degree of complexity in conserving the system wide energy. For minimizing energy consumption for weakly hard real-time systems modeled by (m,k)-model, Alenawy and Aydin [1] introduced a scheduling technique to maximize (instead of guarantee) the quality level under energy constraints for real-time systems with (m,k)-constraints. Niu and Quan [10] presented a combined static/dynamic DVS scheduling method to reduce processor energy with (m,k)-guarantee. Both techniques focus only on the minimization of the processor energy consumption. Recently, Niu and Quan [11] proposed a scheduling method to reduce the system-wide energy consumption for real-time systems with (m,k)-constraints. The systems in their approach consist of only a non-DVS processor and peripheral devices.

2.3 The motivations

Our goal is to employ DVS and DPD judiciously to save energy and guarantee the (m,k)-constraints in the mean time. The mandatory/optional partitioning plays a critical role in our problem since different mandatory/optional partitions can lead to dramatically different feasibility conditions and therefor have tremendous impacts on the proces-

sor/device power consumption.

There are two known mandatory/optional partitioning techniques proposed in the literature, i.e., R-pattern and E-pattern [10]. The R-pattern, first proposed by Koren *et al.* [7], always the first m_i jobs in a k_i job window as mandatory. It congregates the optional jobs and can lead to large idle intervals. The E-pattern, proposed by Ramanathan *et al.* [15] distributed m_i mandatory jobs evenly among every k_i jobs. The task set is easier to be schedulable with E-patterns since the interference between the mandatory jobs are reduced. As long as the mandatory jobs can meet their deadlines, the (m,k)-constraints are satisfied.

Niu et al. [10] showed that E-patterns can lead to significant dynamic energy reduction for the processor. However, it is not necessary always energy efficient when considering the energy consumed by other peripheral devices. Consider a task set of two tasks, *i.e.*, $\tau_1 = (4,4,2,2,4)$ *and* $\tau_2 = (8,8,4,2,4)$.. 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. Since E-patterns distribute the mandatory jobs evenly, we can see that from Figure 1(a) that the speed of task τ_1 can be reduced quite effectively. However, since the mandatory jobs are allocated evenly, the idle intervals becomes very short and thus devices cannot be shut down. R-pattern, on the other hand, seems to be a better choice in increasing the length of idle intervals. However, due 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).

It is desirable to devise a new mandatory/optional partitioning strategy based on different characteristics of tasks and peripheral devices. However, to ensure the schedulability and its effectiveness of overall energy savings can be extremely difficult since the partitioning problem as well as the feasibility problem has shown to be NP-hard. We could, however, incorporate the advantages of both R-pattern and E-pattern to achieve better energy saving performance. For example, Figure 1(c) presents a schedule that can serve the purposes of scaling down the processor speed and shutting down the peripheral device simultaneously. By partitioning τ_1 with E-pattern and τ_2 with R-pattern, we can effectively scale down the processor speed while maintaining long idle interval to shut down devices with high power consumption (i.e. device 2). A number of immediate problems follows: (i) how to ensure the mandatory jobs according to the mixed E-pattern and R-pattern can meet their deadlines and thus the (m,k)-constraints, (ii) how to assign the appropriate E-pattern or R-pattern to each task to maximize the energy savings, and (iii) how to employ dynamic scheduling techniques based on the assigned hybrid patterns to deal with the run-time variations.

3 The hybrid partitioning strategy

In this section, we first derive a sufficient and necessary condition to check the schedulability for a task set with mixed E-patterns and R-patterns. We then develop a heuristic to assign E-pattern and R-pattern for different tasks based on characteristics of tasks and peripheral devices.

3.1 The feasibility condition

A key problem 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 resultant mandatory job set.

Theorem 1 *Let $\mathcal{T} = \mathcal{R} \bigcup \mathcal{E}$, where $\mathcal{R}$ and $\mathcal{E}$ represent the mandatory job sets according to the R-pattern and E-pattern, respectively. Also, let $W^R(0,t)$ and $W^E(0,t)$ represent the total workload from $\mathcal{R}$ and $\mathcal{E}$ that arrive at or after time 0 have to be finished before t. Then all mandatory jobs can meet their deadlines* iff

$$W^R(0,t) + W^E(0,t) \leq t \tag{1}$$

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

$$t = \begin{cases} pT_i + D_i & p \in Z, p \bmod k_i \leq m_i, \forall \tau_i \in \mathcal{R} \\ \lfloor q\frac{k_i}{m_i} \rfloor T_i + D_i, & p \in Z, \forall \tau_i \in \mathcal{E}. \end{cases} \tag{2}$$

Given the regularity of E-patterns and R-patterns, $W^R(0,t)$ and $W^E(0,t)$ can be well formulated [9]. 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 proof of Theorem 1 can be done by exploiting the general sufficient and necessary condition for tasks scheduled according to EDF as well as the fact that for both R-pattern and E-pattern $W^R(0,t)$ and $W^E(0,t)$ are the largest, compared with any mandatory workload within the same length. Due to the page limit, we omit the details for the proofs.

3.2 The pattern assignment

With the schedulability condition estabilished, the problem then becomes how to assign R-patterns and E-patterns appropriately to balance the processor and device power in order to save the overall energy. The following observations help us develop our heuristic (see Algorithm 1) for assigning different patters for different tasks.

Considering a job with workload w and power function for core processor as $P_{cact}(s)$ and the power function for the peripheral device as P_{dact}, the total energy $(E_{total}(s))$

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

Input: $\mathcal{T}$, s_{crit} and L^i_{min} for τ_i;
$\mathcal{E} = \mathcal{T}, \mathcal{R} = \emptyset$, Update = TRUE;
while Update **do**
  Update = FALSE;
  $\mathcal{E}' = \{\tau_i | s_{crit}(\tau_i) \geq 1, \tau_i \in \mathcal{E}\}$;
  if $\mathcal{E}' \neq \emptyset$ **then**
    Let $\tau' \in \mathcal{E}'$ such that $s_{crit}(\tau')$ is the largest;
    if $\mathcal{E} - \tau'$ schedulable **then**
      $\mathcal{E} = \mathcal{E} - \tau', \mathcal{R} = \mathcal{R} + \tau'$;
      Update = TRUE;
    end if
  else
    for $\tau_i \in \mathcal{E}$ **do**
      Let $E_r(\tau_i)$ ($E_e(\tau_i)$) represent the energy consumption on τ_i within one k_i window according to R-pattern (E-pattern) assignment;
      if $E_r > E_e$ AND $\mathcal{E} - \tau_i$ is schedulable **then**
        $\mathcal{E} = \mathcal{E} - \tau_i, \mathcal{R} = \mathcal{R} + \tau_i$;
        Update = TRUE;
      end if
    end for
  end if
end while

consumed to finish this job with speed s can be represented as

$$E_{total}(s) = (P_{cact}(s) + P_{dact}) \times \frac{w}{s}. \quad (3)$$

Then, the speed (s_{crit}) that can minimize $E_{total}(s)$ in equation 3 (so called *the critical speed* [4, 20]) can be computed by solving the following equation:

$$\frac{dE_{total}(s)}{ds}|_{s=s_{crit}} = 0 \quad (4)$$

Since different tasks need different devices, the critical speeds for different tasks can be different. Note that a critical speed higher than 1 implies that the processor speed should never be scaled down for the purpose of saving the overall energy. Assigning R-pattern to such a task helps to extend the idle interval to shut down the corresponding device. On the other hand, if the processor speed is scaled down to lower than the critical speed itself, it will consume more energy to complete a job. Therefore, the processor speed should not be scaled down below its critical speed even it can be done so.

When the processor speed can be scaled down to a level higher than the critical speed but lower than the maximal speed, it becomes more difficult to determine which pattern should be adopted. This is due to the following reasons: (1) setting the processor speed too low will shorten the idle intervals which is not in favor of peripheral device shut down; (2) setting processor speed too high will increase the dynamic energy consumption of the processor; (3) setting processor speed at different levels also affect the pattern assignments for other tasks. In our approach, we solve this problem by comparing the energy consumption for executing the task (e.g. τ_i) within one k_i window. Specifically, we scale down processor speeds for τ_i under R-pattern and E-pattern separately based on feasibility condition (Theorem 1). We then compute the total energy consumption to finish the mandatory jobs of τ_i within one k_i window. Finally, we assign a task with R-pattern if the result energy consumption with R-pattern is lower than that with E-pattern. The algorithm terminates if no pattern assignment is updated.

4 The dynamic scheduling algorithm

Algorithm 1 helps to statically determine the mandatory/optional job partitions and also set up the appropriate scaling factor for each task to guarantee the (m,k)-constraints. 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 on-line. We are therefore interested in developing a dynamic scheduling technique to achieve better energy-saving performance.

Niu et al. [10] proposed a strategy to change the mandatory/optional jobs dynamically. We can prove that this strategy is still valid in our case when different tasks may be assigned different patterns. With the consideration of peripheral devices, the only difference is to run the optional jobs when the associated device cannot be shut down and run it with the critical speed rather than the lowest possible speed. Kim et al. [5] proposed another method, i.e., to control the preemptions dynamically, to save the energy. Their approach needs to increase the processing speeds of the jobs, which would increase the processor energy consumption and therefore might not necessarily energy efficient. In what follows, we adopt another strategy to delay the executions of higher priority jobs. Different from the delay approach from that in [5], we do not need to increase the processing speed and therefore have a better energy efficiency.

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

Definition 1 *[11] Let $\mathcal{T}_m$ be the mandatory job set, determined based on either R-patterns or E-patterns, scheduled according to EDF. Let R_i be the worst case response time of $\tau_i \in \mathcal{T}_m$. The* delay factor *for τ_i (denoted as Y_i) is defined as $Y_i = (D_i - R_i)$.*

The worst case response time for a task set scheduled with EDF can be computed off-line in a similar way to that in [8]. With the definition of delay factor Y_i, we have the following theorem.

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

1: **Input:** The current job J_{cur} and the current time t_{cur} ;
2: Let $\mathcal{J}_{cur}$ be coming mandatory jobs with priorities higher than J_{cur};
3: Compute t_{np} based on equation (5);
4: Execute J_{cur} non-preemptively within $[t_{cur}, t_{np}]$;
5: Update t_{cur};
6: **if** J_{cur} is completed **then**
7: **if** J_i is optional job **then**
8: Shift the pattern based on the approach in [10];
9: **end if**
10: Let t_n be the arrival time of the next coming mandatory from the same task;
11: **if** $t_n - t_{cur} > L^i_{min}$ **then**
12: Shut down the device L_{cur} and set up the wake up timer to be $t_n - t_{cur}$;
13: **end if**
14: **end if**

Theorem 2 *Let $\mathcal{T}_m$ be the mandatory job set, determined based on either R-patterns or E-patterns, scheduled according to EDF. Also let $hp(J_i)$ be the set of each job J_p with 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*

$$t_{np} = \min_{J_p \in hp(\tau_i)} (r_p + Y_p). \quad (5)$$

Theorem 2 allows us to delay the higher priority jobs safely without increasing the processor speed. Delaying the execution of higher priority jobs helps to reduce their preemptions on lower priority ones. As a result, the devices associated for the lower priority jobs can be shut down earlier instead of being kept active during the preemption period.

With Theorem 2, we are now ready to formulate our dynamic scheduling algorithm, which is shown in Algorithm 2. Algorithm 2 combines both the dynamic mandatory job pattern adjustment and dynamic preemption control and therefore can achieve much better performance as demonstrated in the next section. To ensure the effectiveness and efficiency of this algorithm, we have the following theorem.

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.*

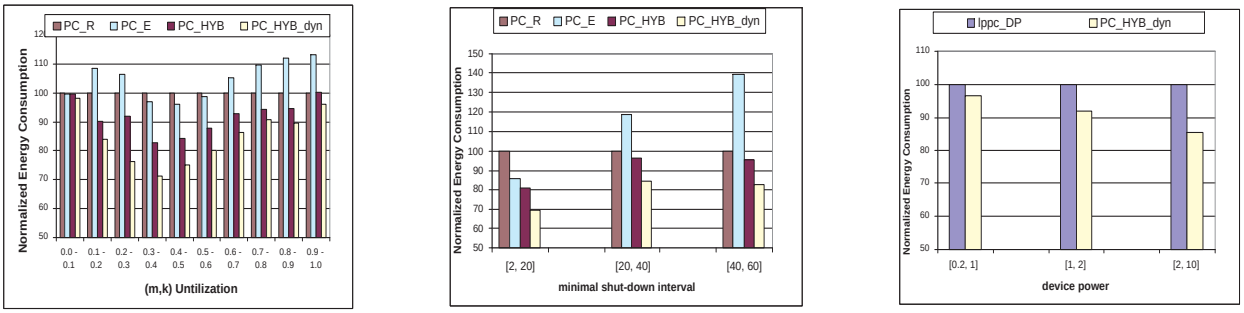
5 Experimental Results

In this section, we evaluate the performance of our approach through simulations. We implemented four different approaches. In the first approach, the task sets are statically partitioned with *R*-patterns, and the mandatory jobs are executed with the statically determined speed. We refer this approach as (PC_R) and use its results as the reference results. The second approach (PC_E) partitions the mandatory/optional jobs based on *E*-patterns. PC_E is essentially the approach in [10] with the extra considerations of critical speed. The third approach (PC_{HYB}) adopts the static hybrid pattern proposed in Section 3.2. The fourth approach ($PC_{HYB-dyn}$) is our new approach adopting the dynamic preemption control and pattern adjustment proposed in Section 4.

Three groups of experiments were conducted. In the first group 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 $1ms$ 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 $2 \leq m_i < k_i$. We varied the (m,k)-utilization, i.e., $\sum_i \frac{m_i C_i}{k_i T_i}$, of the task by step of 0.1, and generated at least 20 schedulable task sets within each interval or until at least 5000 task sets have been generated. The device associated with each task was randomly chosen from three types of devices: $M_1(0.5,0,5)$, $M_2 = (1,0,15)$, and $M_3 = (5,0,30)$. The power consumption is related to the maximal consumption of the processor and the minimal interval length is in mini-second unit. We assume that the processor minimal shut-down interval length $T_{th} = 2ms$. The results are shown in Figure 2(a).

While it is shown [10] that E-pattern assignment always dominates the R-pattern assignment in reducing the processor energy, this is not the case any more when peripheral devices are taken into consideration as shown in Figure 2(a). We can 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, the dynamic algorithm $PC_{HYB-dyn}$ that adopts dynamic preemption control and dynamic pattern adjustment can further reduce the energy by up to 15%.

In the second group 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-groups 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 group) with (m,k)-utilization within [0.3, 0.4] are shown in Figure 2(b).



(a) (b) (c)

Figure 2. (a) The average total energy consumption by the different approaches; (b) The energy comparison for different shut-down interval length; (c) The energy comparison for different preemption control techniques.

As shown in Figure 2(b), when the minimal shut-down intervals are chosen from shorter interval range, *i.e.*, [2, 20]ms, E-patter has better energy performance since E-patterns helps to better slow down the processor. However, as the minimal shut-down interval length grows, R-pattern becomes much better as it provides more chances for the device to be shut down, especially when the shut-down overhead becomes significantly large, i.e., [40,60]ms in Figure 2(b). Note that in all three cases, using hybrid pattern (PC_{HYB}) can achieve the best energy performance among the three. And the dynamic preemption control and pattern adjustment help to further reduce the energy, i.e., around 15%.

The third group of experiments evaluate the effectiveness of our technique on dynamic preemption control. We incorporated the preemption control scheme by Kim [5] into our approach (represented by $lppc_{DP}$) and compared with $PC_{HYB-dyn}$. The task sets were generated in the same way as that for the second group. 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 power consumption for devices from one of three power ranges, [0.5,1], [1,5], and [5,10]. The results, normalized to that by $lppc_{DP}$, are shown in Figure 2(c).

As shown in Figure 2(c), when the device power is very small, the improvement our approach ($PC_{HYB-delay}$) over $lppc_{DP}$ is very limited as the critical speed of the task is much smaller than the maximal speed, which provides more space for $lppc_{DP}$ to change the speed and delay the higher priority mandatory jobs. However, as the device power increases, the improvement of $PC_{HYB-delay}$ becomes more significant. This is because that as 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 delaying higher priority jobs according to $lppc_{DP}$. When the device power is larger than two times of the processor power, the improvement can be around 15% as shown in the figure.

6 Summary

In this paper, we present a dynamic scheduling algorithm to minimize the system wide energy consumption with (m,k)-guarantee. The system consists of a core processor a number of peripheral devices, which have different power characteristics. Different from previous work that adopted single known mandatory/optional partitioning strategy, we propose to incorporate different partitioning strategies based on the power characteristics of the devices as well as the application specifications. We introduce a feasibility condition, and based on which, we propose an algorithm to performance the mandatory/optional job partitions. We also propose a novel preemption control scheme, which can be well incorporated into our dynamic scheduling algorithm. Extensive experiments have been performed and demonstrate the effectiveness of our approach.

References

[1] T. A. AlEnawy and H. Aydin. Energy-constrained scheduling for weakly-hard real-time systems. *RTSS*, 2005.

[2] H. Cheng and S. Goddard. Online energy-aware i/o device scheduling for hard real-time systems. *DATE*, 2006.

[3] L. Doherty, B. Warneke, B. Boser, and K. Pister. Energy and performance considerations for smart dust. *International Journal of Parallel Distributed Systems and Networks*, 4(3):121–133, 2001.

[4] R. Jejurikar and R. Gupta. Dynamic voltage scaling for system-wide energy minimization in real-time embedded systems. *ISLPED*, 2004.

[5] C. Kim and K. Roy. Preemption-aware dynamic voltage scaling in hard real-time systems. *ISLPED*, 2004.

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

[7] G. Koren and D. Shasha. Skip-over: Algorithms and complexity for overloaded systems that allow skips. In *RTSS*, 1995.

[8] M.Spuri. Analysis of deadline scheduled real-time systems. In *Rapport de Recherche RR-2772, INRIA, France*, 1996.

[9] L. Niu and G. Quan. Energy-aware scheduling for real-time systems with (m,k)-guarantee. In *Technical Report TR-2005-005, Department of Computer Science and Engineering, University of South Carolina*, 2005.

[10] L. Niu and G. Quan. Energy minimization for real-time systems with (m,k)-guarantee. *IEEE Trans. on VLSI, Special*

Section on Hardware/Software Codesign and System Synthesis, pages 717–729, July 2006.

[11] L. Niu and G. Quan. System-wide dynamic power management for multimedia portable devices. *accepted by IEEE International Symposium on Multimedia (ISM'06)*, 2006.

[12] P. Pillai and K. G. Shin. Real-time dynamic voltage scaling for low-power embedded operating systems. In *SOSP*, 2001.

[13] G. Quan and X. Hu. Enhanced fixed-priority scheduling with (m,k)-firm guarantee. In *RTSS*, pages 79–88, 2000.

[14] K. Ramamritham and J. A. Stankovic. Scheduling algorithms and operating system support for real-time systems. *Proceedings of the IEEE*, 82(1):55–67, January 1994.

[15] P. Ramanathan. Overload management in real-time control applications using (m,k)-firm guarantee. *IEEE Trans. on Paral. and Dist. Sys.*, 10(6):549–559, Jun 1999.

[16] M. A. Viredaz and D. A. Wallach. Power evaluation of a handheld computer. *IEEE Micro*, 23(1):66–74, 2003.

[17] R. West and K. Schwan. Dynamic window-constrained scheduling for multimedia applications. In *ICMCS*, 1999.

[18] F. Yao, A. Demers, and S. Shenker. A scheduling model for reduced cpu energy. In *AFCS*, pages 374–382, 1995.

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

[20] J. Zhuo and C. Chakrabarti. Systemlevel energyefficient dynamic task scheduling. *DAC*, 2005.