Procrastination Scheduling for Fixed-Priority Tasks with Preemption Thresholds

XiaoChuan He and Yan Jia

Institute of Network Technology and Information Security
School of Computer Science
National University of Defense Technology
Changsha, China 410073
XiaoChuanHe@gmail.com

Abstract. Dynamic Voltage Scaling (DVS), which adjusts the clock speed and supply voltage dynamically, is an effective technique in reducing the energy consumption of embedded real-time systems. However, the longer a job executes, the more energy in the leakage current the device/processor consumes for the job. Procrastination scheduling, where task execution can be delayed to maximize the duration of idle intervals by keeping the processor in a sleep/shutdown state even if there are pending tasks within the timing constraints imposed by performance requirements, has been proposed to minimize leakage energy drain. This paper targets energy-efficient fixed-priority with preemption threshold scheduling for periodic real-time tasks on a uniprocessor DVS system with non-negligible leakage power consumption. We propose a two-phase algorithm. In the first phase, the execution speed, i.e., the supply voltage of each task are determined by applying offline algorithms, and in the second phase, the procrastination length of each task is derived by applying on-line simulated work-demand time analysis, and thus the time moment to turn on/off the system is determined on the fly. A series of simulation experiments was evaluated for the performance of our algorithms. The results show that our proposed algorithms can derive energy-efficient schedules.

1 Introduction

Low power utilization has been an important issue for hardware manufacturing for next-generation portable, scalable, and sophisticated embedded systems. To reduce the power consumption without the sacrifice of performance, architectural techniques have been proposed to dynamically trade the performance and power consumption. Dynamic Voltage Scaling (DVS), which adjusts the supply voltage and its corresponding clock frequency dynamically, is one of the most effective low-power design technique for embedded real-time systems. Since the energy consumption of CMOS circuits has a quadratic dependency on the supply voltage, lowering the supply voltage is one of the most effective ways of reducing the energy consumption.

In many real-time applications, average or worst-case task response time is an important non-functional design requirement of the system. For example, to maintain the system stability, many embedded real-time systems must complete the tasks before

J. Cao et al. (Eds.): NPC 2008, LNCS 5245, pp. 255–265, 2008.

their deadlines. For real-time systems targeting commercial variable voltage microprocessors, since lowering the supply voltage also decreases the maximum achievable clock speed [1], *energy-efficient* task scheduling is to reduce supply voltage dynamically to the lowest possible level while satisfying the tasks'timing constraints. In the past decade, energy-efficient task scheduling with various deadline constraints received extensive attention, especially for the minimization of the energy consumption of the dynamic voltage scaling part in a uniprocessor environment [2].

Recently, researchers have started exploring energy-efficient scheduling with the considerations of leakage current since the power consumption resulting from leakage current is comparable to the dynamic power dissipation [3]. To reduce the energy consumption resulting from leakage current, a system might be turned off (to enter a *dormant mode*). For periodic real-time tasks, Jejurikar et al. [4] and Lee et al. [5] proposed energy-efficient scheduling on a uniprocessor by procrastination scheduling to decide when to turn off the system. Jejurikar and Gupta [3] then further considered real-time tasks that might complete earlier than its worst-case estimation by extending the algorithms presented in [4].

Fixed-priority preemptive (FPP) scheduling algorithms and fixed-priority non-preemptive (FPNP) scheduling algorithms are two important classes of real-time scheduling algorithms. To obtain the benefits of both FPP and FPNP algorithms, there are several other algorithms trying to fill the gap between them. The fixed-priority with preemption threshold (FPPT) scheduling algorithm [6] is one of them. Under FPPT, each task has a pair of priorities: regular priority and preemption threshold, where the preemption threshold of a task is higher than or equal to its regular priority. The preemption threshold represents the tasks running-time preemption priority level. It prevents the preemption of the task from other tasks, unless the preempting tasks priority is higher than the preemption threshold of the current running task. Saksena and Wang have shown that task sets scheduled with FPPT can have significant schedulability improvements over task set using fixed priorities [6].

This paper considers energy-efficient FPPT scheduling of periodic real-time tasks on a uniprocessor whose dynamic voltage scaling portion might be turned off for further energy saving. We further combine procrastination scheduling with dynamic voltage scaling to minimize the total static and dynamic energy consumption of the system. An on-line algorithm was developed to calculate the respective procrastination interval for each task. A series of simulation experiments was also evaluated for the performance of our algorithms. The results show that our proposed algorithms can derive energy-efficient schedules.

The rest of this paper is organized as follows: Section 2 defines the leakage-aware energy-efficient FPPT scheduling problem in a uniprocessor system. Preliminary results are shown in Section 2. The proposed algorithms are in Section 3. Experimental results for the performance evaluation of the proposed algorithms are presented in Section 4. Section 5 is the conclusion.

2 System Model

2.1 Task Model

This study deals with the fixed priority preemptive scheduling of tasks in a real-time systems with hard constraints, i.e., systems in which the respect of time constraints is mandatory. The activities of the system are modeled by periodic tasks.

The model of the system is defined by a task set $\mathcal{T}$ of cardinality n, $\mathcal{T} = \{\tau_1, \tau_2, ..., \tau_n\}$. The j_{th} job of task τ_i is denoted as $J_{i,j}$. The index, j, for jobs of a task is started from zero. A periodic task τ_i is characterized by a 3-tuple (C_i, T_i, D_i) where each request of τ_i, called instance, has an execution CPU cycles (denoted as C_i), and a relative deadline (denoted as D_i). T_i time units separate two consecutive instances of τ_i (hence T_i is the period of the task). Given a set $\mathcal{T}$ of n tasks, the hyper-period of $\mathcal{T}$, denoted by $\mathcal{L}$, is defined so that $\mathcal{L}/T_i$ is an integer for any task τ_i in $\mathcal{T}$. The number of jobs in the hyper-period of task τ_i is $\mathcal{L}/T_i$. For example, $\mathcal{L}$ is the least common multiple (LCM) of the periods of tasks in $\mathcal{T}$ when the periods of tasks are all integer numbers. We focus on the case that all of the tasks arrive at time 0.

We also associate with each task τ_i a unique priority $\pi_i \in \{1, 2, ..., n\}$ such that contention for resources is resolved in favor of the job with the highest priority that is ready to run.

The analysis presented in section 3 uses the concept of *busy* and *idle periods* [7]. These are defined as follows: A *level-i busy period* is a continuous time interval during which the notional run-queue contains one or more tasks of active priority level π_i or higher. Similarly, a *level-i idle period* is a time interval during which the run-queue is free of level π_i or higher priority tasks. We note that the run queue may become momentarily free of level-i tasks, when one tasks completes and another is released. This appears in our formulation as an idle period of zero length.

2.2 Power Consumption and Execution Models

We explore energy-efficient scheduling on a dynamic voltage scaling (DVS) processor. The power consumption is contributed by the dynamic power consumption resulting from the charging and discharging of gates on the CMOS circuits and the static power consumption resulting from leakage current. The dynamic power consumption P_d of the dynamic voltage scaling part of the processor is a function of the adopted processor speed f:

$$P_d = C_{eff} V_{DD}^2 f \tag{1}$$

$$f = \alpha k' \frac{(V_{DD} - V_{TH})^\alpha}{V_{DD}} \tag{2}$$

where k' is a device related parameter, V_{TH} is the threshold voltage, C_{eff} is the effective switching capacitance per cycle and α ranges from 2 to 1.2 depending on the device technology. Since power varies linearly with the clock speed and the square of the voltage, adjusting both can produce cubic power reductions, at least in theory. The static power consumption P_s of the system comes from the leakage current of the processor, system I/O devices, and RAM. It might be modeled as a nonnegative constant, as in [8], or a linear function of the supply voltage (a sub-linear function of the execution speed) [3], [4],[9], [10].

The power consumption of processor is denoted by P, which is the sum of the dynamic and static power consumption. We consider systems in which $P(f)$ is a convex and increasing function, and $P(f)/f$ is a convex function, similarly to [4],[11].

Recent processors support multiple variable voltage and frequency levels for energy efficient operation of the system. Let the available frequencies be $\{FLK_1, FLK_2, ..., FLK_s\}$ in increasing order of frequency and the corresponding voltage levels be $\{v_1, v_2, ..., v_s\}$. We assume that the CPU speed f_i of task τ_i can be changed between a minimum speed FLK_1 (minimum supply voltage necessary to keep the system functional) and a maximum speed FLK_s. In our framework, the voltage/speed changes take place only at context switch time and while state saving instructions execute. If not negligible, the voltage change overhead can be incorporated into the worst-case workload of each task.

The system could enter the *dormant mode* (or be turned off) whenever needed. The power consumption of the system is treated as 0 when it is in the *dormant mode* [8] by scaling the static power consumption. We consider systems that could be turned on/off at instant. When needed, turning the system off might further reduce the energy consumption. The energy consumption to turn off the system is assumed to be negligible, but it might require additional energy to turn on the system [12]. We denote E_{sw} as the energy of the switching overhead from the dormant mode to the active mode. For the rest of this paper, we say the system is idle at time instant t, if the processor does not execute any task at time instant t. When the system is active and idle, the processor executes NOP instructions and must be at processor speed FLK_1 to minimize the energy consumption. Let P_I be the power consumption when the system is idle and active, where $P_I = P(FLK_1)$.

2.3 Critical Speed

The critical speed $\hat{f}$ is defined as the available speed of the processor to execute a cycle with the minimum energy consumption. Because of the convexity of $P(f)$, executing at a common speed for a CPU cycle minimizes the energy consumption. Hence, the energy consumption to execute a CPU cycle at speed f is $P(f)/f$. Since the power consumption function $P(f)$ is a convex and increasing function, where $P(f)/f$ is merely a convex function. $P(f)/f$ is minimized when f is equal to f^*, with $\frac{d(P(f^*)/f^*)}{df^*} = 0$. As a result, to minimize the execution energy consumption of $\mathcal{T}$, we do not have to consider schedules that execute jobs at any lower speed than f^* since we could execute jobs at speed f^* with lower energy consumption and less execution time. If f^* is between FLK_1 and FLK_s, we know that $\hat{f}$ is f^*. If f^* is less than FLK_1, $\hat{f}$ is set to FLK_1 to satisfy the hardware constraint. Similarly, if f^* is greater than FLK_s, $\hat{f}$ is set to FLK_s, and jobs are executed at FLK_s to minimize the energy consumption. As a result, $\hat{f}$ is $min\{max\{f^*, FLK_1\}, FLK_s\}$. Executing a job of task τ_i at any speed less than $\hat{f}$ would either consume more energy than that at $\hat{f}$ with more execution time or violate the speed constraint. We assume that f^* could be obtained efficiently or pre-determined as a specified parameter in the input.

2.4 Problem definition

The problem considered in this paper is as follows:

Definition 1. (Leakage-Aware Energy-Efficient Scheduling for FPPT, LAEES-FPPT) *Consider a set $\mathcal{T}$ of n independent tasks ready at time 0. Each periodic task $\tau_i \in \mathcal{T}$ is associated with a computation requirement equal to C_i CPU-cycles and its period T_i, where the relative deadline of τ_i is equal to D_i. And each task $\tau_i \in \mathcal{T}$ is assigned with a unique priority $\pi_i \in \{1, 2, ..., n\}$ and a preemption threshold $\gamma_i \in \{1, 2, ..., n\}$ ($\gamma_i \geq \pi_i$), where π_i is used to compete for processor and $gamma_i$ is used to protect τ_i from unnecessary task preemptions after τ_i starts. The power consumption function $P(f)$ is a convex and increasing function, while $P(f)/f$ is merely a convex function. The processor is with a discrete spectrum of the available speeds in $[FLK_1, FLK_s]$. The energy of the switching overhead from the dormant mode to the active mode of a system is E_{sw}, and the power consumption when the system is active and idle is P_I . The problem is to minimize the energy consumption in the hyper-period $\mathcal{L}$ of tasks in $\mathcal{T}$ in the scheduling of fixed-priority tasks with preemption thresholds in $\mathcal{T}$ without missing the timing constraints.* □

A schedule of a task set $\mathcal{T}$ is an assignment of the available processor speeds for each corresponding task execution, where the job arrivals of each task $\tau_i \in \mathcal{T}$ satisfy its timing constraint D_i. A schedule is feasible if no job misses its deadline. A schedule is optimal for the LAEES-FPPT problem, if it is feasible, and its energy consumption is the minimum among all feasible schedules. For the rest of this paper, let S^* be an optimal schedule for $\mathcal{T}$. For a schedule, an idle interval is a maximal interval when the system is idle, while an execution interval is a maximal interval when the processor executes some jobs. The system might be turned off or be at the active mode in an idle interval, while the system is active in an execution interval. For any set X, let $|X|$ be the cardinality of the set. For example, $|I_S|$ is the number of idle intervals in schedule S in $(0, \mathcal{L}]$. If the idle interval is greater than E_{sw}/P_I, turning off the system is worthwhile. Let t_θ be the *threshold idle interval* E_{sw}/P_I. If the idle interval is greater than t_θ, the longer the idle interval is, the more the energy saved by turning off the system.

The energy consumption of a schedule S, denoted as $\mathrm{E}(S)$, consists of two parts: the execution energy consumption $\phi(S)$ and the idle energy consumption $\varepsilon(S)$. The execution energy consumption is the sum of the energy consumption of the executions of jobs in S in the time interval $(0, \mathcal{L}]$. The idle energy consumption is the sum of the energy consumption in the intervals in $(0, \mathcal{L}]$ in which the system does not execute any job. Let $\upsilon(t, S)$ be the speed at time instant t in schedule S. The execution energy consumption $\phi(S)$ in $\mathrm{E}(S)$ is $\int_0^{\mathcal{L}} P(\upsilon(t, S))dt$. The idle energy consumption $\varepsilon(S)$ in $\mathrm{E}(S)$ is the summation of E_{sw} times the number of instances that the system is turned from the dormant mode to the active mode and P_I times the total interval length that the system is idle and active in $(0, \mathcal{L}]$.

3 Proposed Algorithms

This section presents a two-phase algorithm for periodic real-time tasks. The algorithms determine, in the first phase, the execution speed, i.e., the supply voltage, of each task, and in the second phase the moment to turn on/off the system on the fly.

3.1 An On-Line Procrastination Algorithm to Minimize the Energy Leakage: LA-FPPT

Let S_e be the resulting FPPT schedule by applying some off-line dvs algorithms [13]. For brevity, let $\boldsymbol{C}_i$ be the execution time of a job of task τ_i in S_e, $\boldsymbol{C}_i = C_i / f_i^{opt}$. The first phase of the proposed algorithm [13] decides the execution speed of tasks in $\mathcal{T}$ to meet the timing constraints and minimize the execution energy consumption.

The second phase is to reduce the idle energy consumption by turning the system off on the fly. The idea behind scheduling on the fly is to lengthen and aggregate the idle intervals so that the resulting idle time is long enough to turn off the system. The determination of idle intervals can be done by procrastinating the arrival time of the next job to the system, as in [4],[14] for EDF scheduling, and in [9],[11] for fixed-priority scheduling. In [4], [9]procrastination is done by computing the maximum procrastination intervals of all of the tasks in $\mathcal{T}$ based on the system utilization, while the idle intervals in [14],[11] are determined by procrastinating the remaining jobs as late as possible.

In this section, we proposes an on-line simulated work-demand analysis approach to the determination of idle intervals. If a job completes at time instant t, and the ready queue is empty, we have to decide whether the system should be turned off or idle. Let $r_i(t)$ be the arrival time of the next job of τ_i for any τ_i in $\mathcal{T}$ arrived after time instant t, i.e., $r_i(t) = \left\lceil \frac{t}{T_i} \right\rceil \cdot T_i$. Let $d_i(t)$ be the next deadline on an invocation of task τ_i after time instant t, i.e., $d_i(t) = r_i(t) + D_i$.

Our formulation stems from considering the schedulability of each fixed-priority task with preemption thresholds at time instant t. We focus on finding the maximum amount of idle interval, $S_{\pi_i}^{max}(t)$, which may be stolen at priority level π_i, during the interval $[t, t + d_i(t))$, whilst guaranteeing that task τ_i meets its deadline. (Note, $S_{\pi_i}^{max}(t)$ may not actually be available for idle due to the constraints on hard deadline tasks with priorities lower than π_i. We return to this point later). To guarantee that task τ_i will meet its deadline, we need to analyze the worst case scenario from time t onwards. We therefore assume that all tasks τ_j are re-invoked at their earliest possible next release $r_j(t)$ and subsequently with a period of T_j.

In attempting to determine the maximum guaranteed idle time, $S_{\pi_i}^{max}(t)$, it is instructive to view the interval $[t, t + d_i(t))$ as comprising a number of *level-i* busy and idle periods. Any *level-i idle time* between the completion of task τ_i and its deadline could be swapped for task τ_i's procrastination interval Z_i without causing the deadline to be missed. Hence the maximum procrastination interval Z_i which may be stolen is equal to the total *level-i idle time* in the interval. We use this result to calculate $S_{\pi_i}^{max}(t)$.

We first derive equation 3 using techniques given in [15]. Although the ready queue is empty at time instant t, two components still determine the extent of the busy period under the influence of procrastination scheduling:

1. For the task τ_k with priority $\pi_k < \pi_i < \gamma_k$, τ_k's released workload just before the start of busy period
2. For the task τ_j with priority $\pi_j > \gamma_i$, τ_j's released workload during the busy period

The second component implies a recursive definition. As the processing released increases monotonically with the length of the busy period, a recurrence relation can be used to find $w_i(t)$:

$$w_{\pi_i}^{m+1}(t) = S_{\pi_i}(t) + \max_{\forall k, \pi_k < \pi_i < \gamma_k} \boldsymbol{C_k} + \sum_{\forall j, \pi_j > \pi_i} \left(\left\lceil \frac{w_{\pi_i}^{m}(t) - x_j(t)}{T_j} \right\rceil \cdot \boldsymbol{C_j} \right) \quad (3)$$

The term $S_{\pi_i}(t)$ represents the beginning of *level-i idle time* from time t.

The recurrence relation begins with $w_{\pi_i}^{0}(t) = 0$ and ends when $w_{\pi_i}^{m+1}(t) = w_{\pi_i}^{m}(t)$ or $w_{\pi_i}^{m+1}(t) > d_i(t)$. Proof of convergence follows from analysis of similar recurrence relations by Audsley et al [15]. The final value of $w_{\pi_i}(t)$ defines the length of the busy period. Alternatively, we may view $t + w_{\pi_i}(t)$ as defining the start of a *level-i idle time*.

Given the start of a *level-i idle time*, within the interval $[t, t + d_i(t))$, the end of the idle time, which may be converted to procrastination interval of task τ_i, occurs either at the next release of a task τ_j with priority $\pi_j > \pi_i$ or at the end of the interval. Equation 4 gives the length, $l_i(t, w_{\pi_i}(t))$, of the *level-i idle time*.

$$l_i(t, w_{\pi_i}(t)) = \min \left[\begin{array}{l} d_i(t) - w_i(t), \\ \min\limits_{\forall j, \pi_j \geq \gamma_i} \left(\left\lceil \frac{w_{\pi_i}(t) - r_j(t)}{T_j} \right\rceil \cdot T_j + r_j(t) - w_{\pi_i}(t) \right) \end{array} \right] \quad (4)$$

where the term $d_i(t) - w_{\pi_i}(t)$ means that the end of *level-i idle time* come about at the end of $[t, t + d_i(t))$, the term $\left\lceil \frac{w_{\pi_i}(t) - r_j(t)}{T_j} \right\rceil \cdot T_j + r_j(t)$ describe the workload contributed by task τ_j in the *level-i busy period*, whose length is denoted by $w_{\pi_i}(t)$.

Combining equations 3 and 4, our method for determining the maximum idle time, $S_{\pi_i}^{max}(t)$, proceeds as follows:

1. The idle time which may be derived, $S_{\pi_i}(t)$, is initially set to zero
2. Equation 3 is used to compute the end of a busy period in the interval $[t, t + d_i(t))$
3. The end of the busy period is used as the start of an idle period by equation 4 which returns the length of contiguous idle time.
4. The idle time, $S_{\pi_i}(t)$ is incremented by the amount of idle time found in step 3.
5. If the deadline on task τ_i has been reached, then the maximum idle time which can be derived is given by $S_{\pi_i}(t)$. Otherwise, we repeat steps 2 to 5.

The pseudo-codes of dynamic procrastination algorithm at time instant t when the ready queue is empty, and a job completes are shown in Algorithm 1.

4 Case Studies and Simulations

Section 3 showed that our two-phase algorithm (EE-FPPT [13] + LA-PFFT) will always render the controlled leakage current in CMOS circuits and reduced energy consumptions that will maintain the schedulability of the workload. we use randomly-generated workloads to examine broad trends across a range of design points.

We investigate workload characteristics that affect the energy saving capability attainable through LA-FPPT. We now simulate and analyze randomly generated systems

Algorithm 1. On-line Algorithm to Minimize Energy Leakage

procedure DYNAMIC PROCRASTINATION(t)
▷ a job completes at t and the ready queue is empty
  sort $\mathcal{T}$ by ascending priority order
  for $(i = 1; i \leq n; i \neq n)$ **do**
    $r_i(t) \leftarrow \left\lceil \frac{t}{T_i} \right\rceil \cdot T_i$
    $d_i(t) \leftarrow r_i(t) + D_i$
    $S_{\pi_i}(t) \leftarrow 0$
    $w^{m+1}_{\pi_i}(t) \leftarrow 0$
    while $(w^{m+1}_{\pi_i}(t) \leq d_i(t))$ **do**
      $w^{m}_{\pi_i}(t) \leftarrow w^{m+1}_{\pi_i}(t)$

$$w^{m+1}_{\pi_i}(t) = S_i(t) + \max_{\forall k, \pi_k < \pi_i < \gamma_k} \boldsymbol{C_k} + \sum_{\forall j, \pi_j \geq \pi_i} \left(\left\lceil \frac{w^m_{\pi_i}(t) - r_j(t)}{T_j} \right\rceil \cdot \boldsymbol{C_j} \right)$$

      if $(w^{m}_{\pi_i}(t) = w^{m+1}_{\pi_i}(t))$ **then**
        $S_{\pi_i}(t) \leftarrow S_{\pi_i}(t) + l_i(t, w^m_{\pi_i}(t))$
        $w^{m+1}_{\pi_i}(t) \leftarrow w^{m+1}_{\pi_i}(t) + l_i(t, w^m_{\pi_i}(t))$
      end if
    end while
    $S^{max}_{\pi_i}(t) \leftarrow S_i(t)$
    revise the arrival time $r'_i(t)$ of job $J_{i,t}$ by setting $r'_i(t) \leftarrow r_i(t) + S^{max}_{\pi_i}(t)$
  end for
  if $(\min_{\forall \tau_i \in \mathcal{T}} r'_i(t) - t > t_\theta)$ **then**
    turn the system off at time t and turn on at $\min_{\forall \tau_i \in \mathrm{T}} r'_i(t)$
  else
    remain on the active mode
  end if
end procedure

of tasks to better understand our approaches. The power consumption function of the system speed f was set as $P(f) = f^3 + 3$.

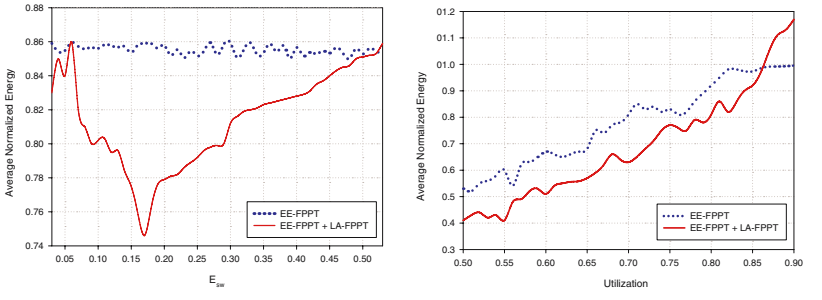
The *normalized total energy* was adopted as the performance metrics. The normalized total energy of an algorithm for an input instance is the energy consumption of the derived solution in $(0, \mathcal{L}]$ divided by the energy consumption by applying the original FPPT scheduling without processor slowdown, procrastination and by turning off the system when the idle interval is long enough.

We tried two different experimental settings. The first experiment investigate separately the effect of the switching overhead E_{sw}, the system utilization on the limited energy consumption achieved by our methods. To cover a wide range of design points, 20,000 real-time task sets with 10 tasks each were randomly generated. These were created so 1000 have a utilization of 50%, 1000 have 52% utilization, and so on up to 90%. For each group of task sets who hold the same utilization, those were created so 20 have a E_{sw} of 0.03, 20 have 0.04, and so on up to 0.53. The second one focused on the impact of the number of tasks and E_{sw} (0.17), another 20,000 real-time task sets with system

utilization 67% each were randomly generated too. Those were created so 1000 include 5 independent tasks, 1000 include 6 independent tasks, and so on up to 25 tasks.

Task periods is assigned randomly in the range [1, 100] with a uniform probability distribution function. Moreover, task deadlines were set equal to their respective periods (for simplicity, though not necessary). Tasks' WCETs were set to incur the required overall system utilization. All 40,000 real-time task sets generated were schedulable with a fully preemptive policy.

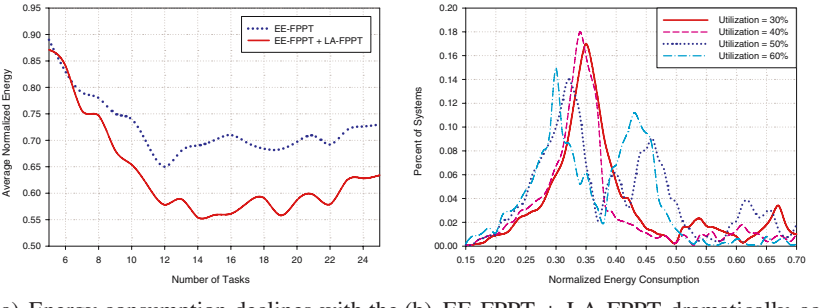
Using the MPTA, the total energy produced by each system was computed and normalized to the energy required by the original version of the system. The average normalized energy were then plotted as a function of E_{sw}, the system utilization and the number of tasks in turn. The results are shown in figures 1(a), 1(b) and 2(a) respectively.



(a) Energy consumption produced for LA-FPPT rises with large E_{sw}, but keep almost constant for EE-FPPT, with utilization = 0.67

(b) Energy consumption rises with system utilization, but soars up for high-utilization systems.

Fig. 1. Experiment I results for power saving of our approaches



(a) Energy consumption declines with the increment of the number of tasks, on the condition that system utilization = 0.67 and E_{sw} = 0.17

(b) EE-FPPT + LA-FPPT dramatically accomplishes the energy savings, even for high-utilization systems.

Fig. 2. Experiment II results for power saving of our approaches

In Figure 1(a), the more the switching overhead E_{sw} was, the more the normalized energy consumption was for schedules derived from Algorithms LA-FPPT. When E_{sw} is relatively small ($E_{sw} \leq 0.18$), the energy consumption from leakage current in CMOS circuits still have litter influence on the total energy consumption, thus the more E_{sw}, the less normalized energy consumption.

In Figure 1(b), our algorithms (EE-FPPT + LA-FPPT) outperformed original Algorithm FPPT when the system utilization was greater than 0.87. When the system utilization was large enough, procrastination might create two (or more) idle intervals to turn the system off, but the original FPPT schedule might make the system idle for a short interval and turn the system off for a longer interval. As a result, the energy consumption of procrastination schedules might consume more energy than the original FPPT schedule when the system utilization is large enough. Moreover, when the utilization for task execution is large, the improvement on idle energy consumption is marginal since task execution dominates the total energy consumption.

In Figure 2(a), for all the simulated algorithms, the normalized energy consumption decreased for small number of tasks with $n \leq 12$, and was steady for $n > 12$. This is because the resulting utilization of a task was large when n was small in the experimental setup, and, hence, there was only little room for procrastination to save energy. For task sets with $n > 12$, the maximum procrastination interval was dominated by tasks with small periods, and, hence, the improvement became marginal.

Another interesting property is the distribution of the 20,000 systems of Experiment I among the different normalized power consumption levels. Figure 2(b) show this distribution for the overall system utilization levels of 30%, 40%, 50%, and 60%, respectively. As can be seen, the workloads scheduled with the fixed-priority schemes depend on the system utilization level to some extent.

5 Conclusions

In this paper we discuss the energy-efficient scheduling problem of periodic realtime tasks by applying FPPT policy on a uniprocessor dynamic voltage scaling system that can go into the dormant mode for energy efficiency. We propose a two-phase scheduling algorithm. In the first phase, the execution speed, i.e., the supply voltage, of each task is determined by applying off-line algorithms. In the second phase, the time moment to turn on/off the system is determined on the fly. Theoretical analysis shows that our proposed algorithms could derive scheduling solutions with at most $max\{\frac{1}{(U_{bd})^2}, 2\}$ times of the energy consumption of optimal solutions, where the term U_{bd} represents the *breakdown utilization* [16] of a task set. A series of simulation experiments was evaluated to demonstrate the performance of the proposed algorithms. Our experimental results show that our approaches can accomplish dramatic energy savings as the same time keep the schedulability of task set.

References

1. Takayasu Sakurai, A.R.N.: Alpha-power law mosfet model and its applications to cmos inverterdelay and other formulas. IEEE Journal of Solid-State Circuits 25(2), 584–594 (1990)

2. Padmanabhan Pillai, K.G.S.: Real-time dynamic voltage scaling for low-power embedded operating systems. In: 18th ACM Symposium on Operating System Principles, Chateau Lake Louise, Banff, Alberta, Canada, vol. 35, pp. 89–102. ACM, New York (2001)
3. Ravindra Jejurikar, R.K.G.: Dynamic slack reclamation with procrastination scheduling in real-time embedded systems. In: Joyner Jr., W.H., Martin, G., Kahng, A.B. (eds.) 42nd Design Automation Conference, San Diego, CA, USA, pp. 111–116. ACM Press, New York (2005)
4. Jejurikar, R., Pereira, C., Gupta, R.K.: Leakage aware dynamic voltage scaling for real-time embedded systems. In: Kahng, S.M., Fix, L., Andrew, B. (eds.) 41th Design Automation Conference, San Diego, CA, USA, pp. 275–280. ACM, New York (2004)
5. Lee, Y.-H., Reddy, K.P., Mani Krishna, C.: Scheduling techniques for reducing leakage power in hard real-time systems. In: 15th Euromicro Conference on Real-Time Systems (ECRTS 2003), Porto, Portugal, pp. 105–112. IEEE Computer Society, Los Alamitos (2003)
6. Manas Saksena, Y.W.: Scalable real-time system design using preemption thresholds. In: 21st IEEE Real-Time Systems Symposium, pp. 25–34 (2000)
7. Lehoczky, J.P.: Fixed priority scheduling of periodic task sets with arbitrary deadlines. In: IEEE Real-Time Systems Symposium, Lake Buena Vista, Florida, USA, pp. 201–213. IEEE Computer Society Press, Los Alamitos (1990)
8. Xu, R., Zhu, D., Rusu, C., Chem, R.G.M., Moaaé, D.: Energy-efficient policies for embedded clusters. In: Paek, Y., Gupta, R. (eds.) 2005 ACM SIGPLAN/SIGBED Conference on Languages, Compilers, and Tools for Embedded Systems, Chicago, Illinois, USA. ACM, New York (2005)
9. Ravindra Jejurikar, R.K.G.: Procrastination scheduling in fixed priority real-time systems. In: 2004 ACM SIGPLAN/SIGBED Conference on Languages, Compilers, and Tools for Embedded Systems, Washington, DC, USA, pp. 57–66. ACM, New York (2004)
10. Ravindra Jejurikar, R.K.G.: Dynamic voltage scaling for systemwide energy minimization in real-time embedded systems. In: Roy, R.V.J., Choi, K., Tiwari, V. (eds.) 2004 International Symposium on Low Power Electronics and Design, Newport Beach, California, USA, pp. 78–81. ACM, New York (2004)
11. Quan, G., Niu, L., Hu., X.S., Mochocki, B.: Fixed priority scheduling for reducing overall energy on variable voltage processors. In: 25th IEEE Real-Time Systems Symposium, Lisbon, Portugal, pp. 309–318. IEEE Computer Society, Los Alamitos (2004)
12. Irani, S., Shukla, S.K., Gupta, R.K.: Algorithms for power savings. In: Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 37–46. ACM, New York (2003)
13. XiaoChuan He, Y.J.: Energy-efficient scheduling fixed-priority tasks with preemption thresholds on variable voltage processors. In: Li, K., Jesshope, C., Jin, H., Gaudiot, J.-L. (eds.) NPC 2007. LNCS, vol. 4672, pp. 133–142. Springer, Heidelberg (2007)
14. Linwei Niu, G.Q.: Reducing both dynamic and leakage energy consumption for hard real-time systems. In: Irwin, M.J., Zhao, W., Lavagno, L., Mahlke, S.A. (eds.) 2004 International Conference on Compilers, Architecture, and Synthesis for Embedded Systems, Washington DC, USA, pp. 140–148. ACM, New York (2004)
15. Audsley, N.C., Burns, A., Richardson., M.F., Wellings, A.J.: Applying new scheduling theory to static priority pre-emptive scheduling. Software Engineering Journal 8(5), 284–292 (1993)
16. Lehoczky, J.P., Lui Sha, Y.D.: The rate monotonic scheduling algorithm: Exact characterization and average case behavior. In: IEEE Real-Time Systems Symposium 1989, pp. 166–171 (1989)