Reducing Both Dynamic and Leakage Energy Consumption for Hard Real-Time Systems

Linwei Niu Gang Quan
Department of Computer Science and Engineering
University of South Carolina
Columbia, SC 29208
{niul,gquan}@cse.sc.edu

ABSTRACT

While the dynamic voltage scaling (DVS) techniques are efficient in reducing the dynamic energy consumption for the processor, varying voltage alone becomes less effective for the overall power reduction as the leakage power is growing rapidly, *i.e.,* five times per technical generation as predicted. In this paper, we study the problem of reducing both the static and dynamic power consumption at the same time for the hard real-time system scheduled by the earliest deadline first (EDF) strategy. To balance the dynamic and leakage energy consumption, *higher-than-necessary* processor speeds may be required when executing real-time tasks, which can result in a large number of idle intervals. To effectively reduce the energy consumption during these idle intervals, we propose a technique that can effectively merge these scattered intervals into larger ones without causing any deadline miss. Simulation studies demonstrate the effectiveness of our approach. Specifically, our experiments show that the proposed technique can lead up to more than 80% idle energy savings than that by the previous ones.

Categories and Subject Descriptors

D.4.1 [**Operating Systems**]: Process Management–Scheduling

General Terms

Algorithms

Keywords

DVS, real-time scheduling, leakage power reduction, low power design, embedded system

1. INTRODUCTION

Power consumption has become a major hurdle in design of next generation portable, scalable, and sophisticated embedded systems. Current power saving techniques have been mainly focused on reducing the dynamic power, *i.e.*, the power consumption due to the switching activities in the CMOS circuit, because it has been the predominant component in the overall power consumption for most embedded systems today. However, as VLSI technology marching towards deep submicron and nanoscale circuits operating at multi-GHz frequencies, the rapidly elevated leakage power dissipation will soon become comparable to, if not exceeding, the dynamic power consumption [10]. More advanced power saving techniques are demanded for enabling the development of future generations of embedded systems.

In the face of the increasing challenge presented by the leakage power consumption, design efforts in all fronts must be pursued to form an integration solution for this problem. Recently, many circuit and architecture techniques, such as [17, 5, 3, 12, 6, 1], have been proposed to control the leakage power consumption. It is our belief that real-time scheduling plays a unique role in this effort because not only most of the future embedded systems are real-time systems, but also real-time scheduling has been one of the most effective techniques in energy reduction for the embedded systems.

Many dynamic voltage scaling (DVS) based real-time scheduling techniques, *e.g.* [20, 9, 2, 18], have been proposed to conserve energy. DVS can effectively reduce the dynamic power consumption by dynamically varying the processor speed, based on the workload and the deadline requirements of the real-time applications. However, the energy saving performance achievable via voltage reducing is becoming severely limited with the dramatic increase of the leakage power consumption. To obtain the maximal gain for the overall energy reduction, the DVS-based scheduling techniques must take the leakage power consumption into consideration.

Recently, several scheduling techniques have been reported to reduce the leakage power consumption. In [19], a scheduling algorithm that combines DVS and adaptive body biasing (ABB) is proposed to reduce both dynamic power and leakage power. An analytic model is derived that can be applied to compute the optimal supply voltage and body bias voltage in terms of overall energy reduction for a given clock frequency. For processors with no complex control mechanism for body biasing voltage, processor shut-down is one of the most effective strategies to reduce the leakage power consumption. In regard to this, Lee *et. al.* [13] proposed a leakage reduction scheduling technique called **LC-EDF**, which carefully delays the execution of the arriving task instances when the processor is idle to extend the idle intervals

CASES'04, September 22–25, 2004, Washington, DC, USA.

and reduce the number of power transitions. Specifically, according to **LC-EDF**, the extended idle time is treated as one part of the tasks' execution time. As long as the resultant total utilization is less than or equal to 1, the schedulability of the task set is guaranteed. However, they assume a non-DVS processor model, which cannot optimize the dynamic power consumption. Irani *et. al.* [8] theoretically proved that, with a DVS processor, the voltage schedule that optimizes both the dynamic and leakage power consumption can be constructed from the corresponding DVS voltage schedule that can optimize the dynamic power consumption. In this case, higher-than-necessary processor speeds may be required to balance the dynamic and leakage power consumption, which will produce a large number of scattered idle periods. To better save the energy consumption during these idle periods, Jejurikar *et. al.* [11] proposed an approach, called **CS-DVSP**, to extend the idle interval by delaying the execution of task instances. They theoretically proved that their algorithm can guarantee the minimal idle interval larger than the one by LC-EDF. However, the problem with this approach is that it computes the delays based on the utilization factor, *i.e.*, $U = \sum_i \frac{C_i}{T_i} \leq 1$ [14]. When tasks' deadlines are less than their periods, one has to use the deadlines to replace the periods in this condition, which can be very conservative.

In this paper we are interested in studying the problem to reduce both the dynamic and leakage power consumption simultaneously for hard real-time systems running on a variable voltage processor. In particular, we propose a scheduling technique that combines DVS and shut-down to minimize the overall energy consumption. When the processor is active, the processor speed is chosen such that the dynamic and leakage power consumption are balanced [8]; when the processor is idle, the coming task instances are delayed as late as possible to extend the inter-task idle intervals. Different from **CS-DVSP** and **LC-EDF**, we propose a job-based strategy such that the delay for the coming task instances can be more precisely estimated. With a practical processor model and technology parameters [15], our experiment results show that our approach can significantly outperform the energy saving performance achieved by CS-DVSP and LC-EDF.

This paper is organized as follows. Section 2 introduces the preliminaries related to our problem. Section 3 discusses our leakage conscious DVS scheduling technique. Section 4 demonstrates the effectiveness of our approach based on simulations. Section 5 concludes the paper.

2. PRELIMINARIES

In this section, we describe the real-time system and power model we use in this paper.

2.1 System model

The real-time system that we are interested in consists of M independent periodic tasks, $\mathcal{T} = \{\tau_1, \tau_2, \cdots, \tau_M\}$, scheduled according to the earliest deadline first (EDF) scheme [14]. Each task, $\tau_i = (R_i, C_i, D_i, P_i)$, is characterized by its initial arrival time R_i, workload C_i (CPU cycles, for example), deadline D_i, and period P_i. We assume $D_i \leq P_i$. Each periodic task consists of a sequence of instances, called jobs. In this paper, we perform analysis based on a particular job set instead of task set. Therefore, we denote a job set as $\mathcal{J} = \{J_1, J_2, \cdots J_N\}$, and $J_i = (r_i, c_i, d_i)$, where r_i, c_i, and d_i are the arrival time, worst case execution cycles, and absolute deadline, respectively.

2.2 Power model

In a CMOS circuit, the power consumption includes dynamic and static components during its active operation. The dynamic power consumption consists of the switching power for charging and discharging the load capacitance, and the short circuit power due to the non-zero rising and falling time of the input and output signals. The dynamic power (P_{dyn}) can be represented [4] as

$$P_{dyn} = \alpha C_L V^2 f, \tag{1}$$

where α is the switching activity, C_L is the load capacitance, V is the supply voltage, and f is the system clock frequency. The static power (P_{leak}) can be expressed as

$$P_{leak} = I_{leak} V, \tag{2}$$

where I_{leak} is the leakage current which consists of both the subthreshold leakage current and the reverse bias junction current in the CMOS circuit (For a formal mathematical formulation of I_{leak} and detailed explanations of the related technical parameters, we refer the reader to [11]). The power consumption when the processor is in its active status, i.e, P_{act}, is thus

$$P_{act} = P_{dyn} + P_{leak}, \tag{3}$$

Leakage current increases rapidly with the scaling of the devices. It becomes particularly significant with the reduction of the threshold voltage. Therefore, the leakage power consumption is becoming a major component of P_{act} in future CMOS circuits.

The processor consumes energy not only in its active mode but also when it is idle. When the processor is idle, the major portion of the power consumption comes from the leakage which increases rapidly with the dramatic increasing of the leakage power consumption. It is imperative that this part of energy be effectively reduced for the purpose of overall energy reduction.

Power-down strategy can greatly reduce the energy consumption when the processor is idle. For example, it has been reported in [7] that the power dissipation when the processor is idle can be in the order of 10^3 of that when the processor is shut down. While the processor consumes less power in the power down state, it has to pay extra energy and timing overhead to shut down and later wake up the processor in order to save/restore the context as well as initiate the architectural components such as the cache, translation lookaside buffers, and branch target buffers. One has to be careful if energy can be saved when shutting down the processor since the energy overhead may outweigh the benefit of the energy saving if the idle interval is too short. Assume that the power consumptions of a processor in its idle state and sleeping state are P_{idle} and P_{sleep}, respectively, the energy overhead of shutdown/wakeup is E_o, and the timing overhead is t_o. Let

$$P_{idle} \times t = E_o + P_{sleep} \times t. \tag{4}$$

Then, the processor can be shut down with positive energy savings only when the length of the idle interval is larger than $T_{min} = \max(\frac{E_o}{P_{idle} - P_{sleep}}, t_o)$. We call T_{min} as the *minimal length of the idle interval.*

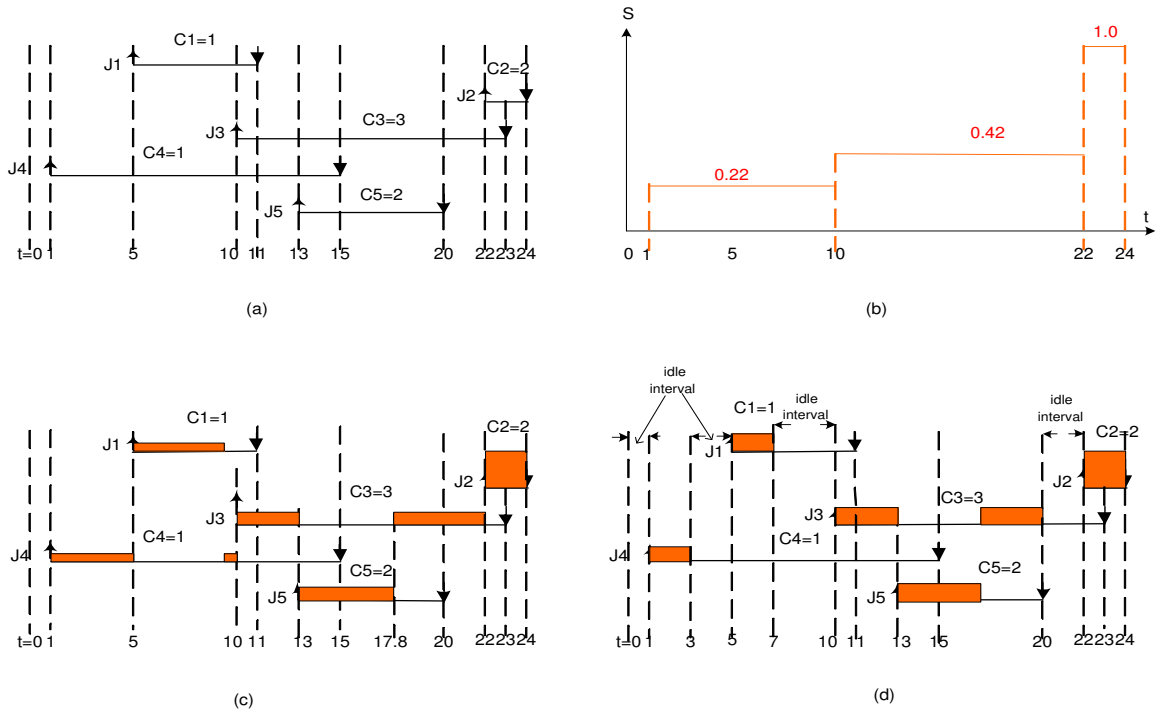

Figure 1: (a) A job set with five jobs. (b) The optimal voltage schedule [20]. (c) The actual job schedule according to the voltage schedule shown in *(b)*. (d) The job schedule with $s_{th} = 0.5$.

2.3 A motivation example

To develop a real-time schedule that can minimize the *overall* energy consumption, both the active energy and idle energy need to be minimized with the deadlines of the tasks guaranteed. While there have been polynomial time solution to reduce the dynamic power consumption alone [20], no practical optimal algorithm has been proposed when the leakage power consumption is taken into consideration.

When processor is active, reducing the processor supply voltage can greatly reduce the dynamic energy consumption since the energy consumption is quadratically related to the supply voltage. Without the consideration of the leakage, it would be most energy efficient to run the tasks with voltage as low as possible. However, reducing the supply voltage usually requires the reduction of the threshold voltage to maintain the circuit performance, which will dramatically increase the leakage current and, hence, the leakage power consumption. Considering a job with workload w and total power function as $P_{act}(s)$, the active energy ($E_{act}(s)$) consumed to finish this job with speed s can be represented as

$$E_{act}(s) = P_{act}(s) \times \frac{w}{s}. \tag{5}$$

Hence, to minimize the energy consumption, *i.e*, $E_{act}(s)$ in equation (5), we have

$$\frac{\partial E_{act}(s)}{\partial s} = 0 \tag{6}$$

and therefore

$$P_{act}(s) = P'_{act}(s)s. \tag{7}$$

Equation (7) computes the most energy efficient speed to execute one job. To increase or decrease the processor speed from this one will increase the dynamic or leakage power consumption. We therefore call this speed as the *threshold speed*, and denoted as s_{th}. Unfortunately, using such a speed is not always feasible in considering the deadlines and the preemptions of the jobs. In addition, the optimal voltage schedule that can optimize *both* the active energy and idle energy is yet to be developed.

Irani *et. al.* [8] present a lemma on the optimal processor speed selection in regard to the overall energy consumption. Let $\mathcal{S}_{dyn}(\mathcal{J})$ denote the voltage schedule for job set $\mathcal{J}$ that can minimize the dynamic energy consumption. The lemma can be rephrased as follows.

LEMMA 1. *[8] Considering both the dynamic and static power consumption of a processor, there exists an optimal voltage schedule $\mathcal{S}_{opt}(\mathcal{J})$ such that all jobs that run with the speeds at or higher than s_{th} in $\mathcal{S}_{dyn}(\mathcal{J})$ run at the same speed and same time in $\mathcal{S}_{opt}(\mathcal{J})$.*

Lemma 1 shows that an overall optimal voltage schedule can be built upon part of the optimal voltage schedule for the dynamic power consumption reduction, *i.e.*, the part in which a processor speed higher than s_{th} is required. For the rest of the jobs, according to equation (7), using s_{th} would be a good choice as it helps to reduce the total active energy consumption.

As a motivation example, Figure 1(a) shows an example of job set with five jobs. The optimal voltage schedule according to [20] and the corresponding schedule are shown in Figure 1(b) and Figure 1(c), respectively. And Figure 1(d) shows the scheduling results with $s_{th} = 0.5$.

As illustrated in Figure 1(b) and Figure 1(c), traditional DVS techniques [20] can effectively reduce the processor

speed. Moreover, when considering the leakage power consumption and applying the threshold speed ($s_{th} = 0.5$) to the voltage schedule, the tasks' deadlines are guaranteed (Figure 1(d)). The problem, however, is that applying s_{th} for the jobs with required speeds lower than s_{th} can result in a large number of small and scattered idle intervals, as shown in Figure 1(d). Shutting down and waking up the processor during these idle intervals will consume a significant amount of energy. In addition, assuming $T_{min} = 3$, it is only energy beneficial to shut down the processor during one in four of the idle intervals shown in Figure 1(d). This implies a great waste of energy in consideration of the fact that the idle power consumption can be 10^3 times as large as that when processor is shut down. In what follows, we introduce our approach that can effectively deal with the idle intervals to save the overall energy consumptions.

3. THE LEAKAGE CONSCIOUS DVS ALGORITHM

In this section, we first describe the general algorithm that can reduce both the dynamic and static power consumption simultaneously while guaranteeing the deadlines for hard real-time systems. Then we present a more delicate approach to deal with the idle intervals to reduce the leakage power during these intervals.

3.1 The general approach

Incorporating the threshold speed in a dynamic voltage schedule can effectively reduce the active energy consumption as explained earlier. Since the idle power consumption can be significantly larger than that when the processor is shut down, one may seek to extend the idle interval by increasing the processor speed. However, increasing processor speed cannot reduce the number of power transitions and thus the associated overheads. Moreover, as shown in equation 7, increasing speed over s_{th} will increase the total active power consumption. A better way to extend the idle intervals is to delay the executions of the coming jobs when the processor is idle. Temporarily withholding the executions of the jobs helps to merge the scattered smaller idle times into larger ones. The energy is saved since the processor may therefore be put into the shut-down status when intervals become longer. Moreover, it saves the energy overhead for shutting down and waking up the processor.

Before we explain our approach in more details, we first introduce several definitions. Since execution of jobs is delayed only when processor is idle, we are more interested in the jobs that arrive later when processor is idle, not the ones that have been completed. Specifically, we have the following definition.

Definition 1. A job set is called a **J_n-job set** (denoted as $\mathcal{J}_n$) if every job J_i in the set satisfies $r_i \geq r_n$.

In addition, delaying the execution of a job set should not cause any future job to miss its deadline, we use the following definition to capture this characteristic for a job set.

Definition 2. The ***latest starting time of a job***, e.g., $J_n \in \mathcal{J}$, (denoted as $t_{LS}(J_n)$) is the latest time such that, if the execution of J_n or the higher priority jobs starts no later than $t_{LS}(J_n)$, J_n will meet its deadline. The ***latest starting time(LST) of a job set***, e.g. $\mathcal{J}$, (denoted as $T_{LS}(\mathcal{J})$) is the latest time such that, if the execution of any jobs in $\mathcal{J}$ starts no later than $T_{LS}(\mathcal{J})$, all jobs will meet their deadlines.

Algorithm **DVSLK** (Algorithm 1) sketches the general idea of our approach. First, the optimal dynamic voltage schedule $S_{dyn}(\mathcal{J})$ is constructed according to [20] that can minimize the dynamic energy. The required processor speed for each job to meet its deadline is thus obtained. When the processor is not idle, it will run the jobs in the ready queue according to EDF and use s_{th} when necessary (line 5). When the processor is idle, Algorithm 1 dictates if the processor should be shut down or be kept idle depending on if the predicted idle interval is long enough. The key to the success of Algorithm 1 is the computation of the latest starting time for the coming job set when the processor is idle (line 8), which will be addressed next.

Algorithm 1 Algorithm to reduce both dynamic and leakage power consumption for real-time systems scheduled according to EDF (Algorithm **DVSLK**)

```
1: Input: J, s_th, and T_min.
2: Compute the dynamic voltage schedule according to [20]
   and s_n, n = 1, 2, ..., N;
3: // s_n is the minimal speed for J_n to meet its deadline
4: if processor is not idle then
5:    Run job J_i in the ready queue according to EDF, using
      s = max(s_i, s_th);
6: else
7:    Let J_n be the next coming job;
8:    Compute T_LS(J_n);
9:    if T_LS(J_n) − t_cur > T_min then
10:      // t_cur is the current time
11:      Shut down the processor and set up the wake up
         timer to be T_LS(J_n) − t_cur;
12:   end if
13: end if
```

3.2 Computing LST for a job set

Delaying execution of jobs helps to extend the idle interval length. At the same time, however, it will also potentially cause a job to miss its deadline. To guarantee the deadline of a job set, Mochocki *et. al.* [16] introduced a method that can be applied to find out the longest time that the future jobs can be delayed if the job set is to be run with a constant speed. Their method is based on the following lemma.

LEMMA 2. *[16] Let job set ($\mathcal{J}$) be executed with a constant speed s^*. Then,*

$$t_{LS}(J_n) = d_n - \sum_{J_k \in hp(J_n)} \frac{c_k}{s^*}, \tag{8}$$

where $hp(J_n)$ is the jobs with the same or higher priorities than that of J_n. Furthermore,

$$T_{LS}(\mathcal{J}) = \min_n t_{LS}(J_n). \tag{9}$$

The rationale behind Lemma 2 is that if the accumulated workload from a job and *all* the higher priority jobs can be finished between its starting time and deadline, the deadline of this job can be satisfied. It is optimal in the sense that delaying execution of jobs any further will cause at least one

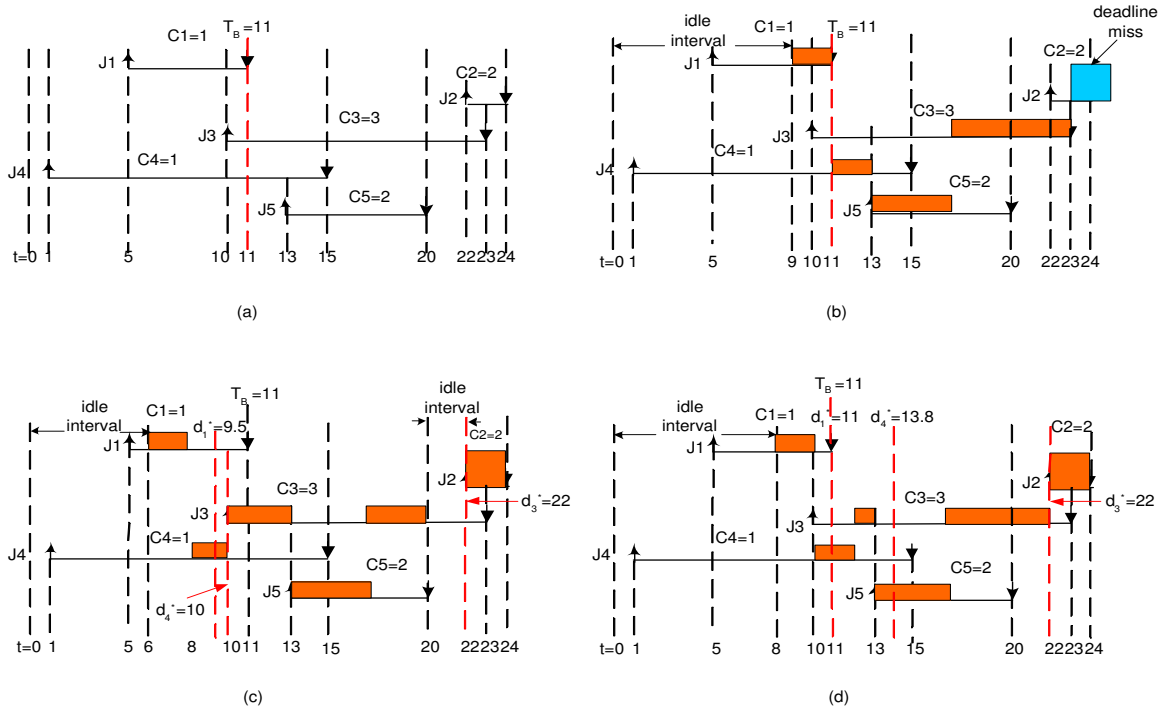

Figure 2: (a) The delay bound for a J_4-job set. (b) With $s_{th} = 0.5$, using the minimum of the latest start times of the jobs arriving before T_B as LST for the job set causes J_2 to miss its deadline. (c) The idle intervals are not effectively merged using $T_{LS} = 6$, computed based on the completion time of the jobs according to the DVS voltage schedule. (d) With $T_{LS} = 8$, all the idle intervals are merged into one single interval and all jobs can meet their deadlines.

deadline miss. When considering the case when different jobs can be run at different speeds, *e.g.,* according to the DVS voltage schedule computed based on [20], Lemma 2 can be readily revised as follows.

LEMMA 3. *Let job $J_n \in \mathcal{J}$, $n = 1, 2, .., N$ be executed with speed s_n. Then, we have*

$$t_{LS}(J_n) = d_n - \sum_{J_k \in hp(J_n)} \frac{c_k}{s_k}, \tag{10}$$

and,

$$T_{LS}(\mathcal{J}) = \min_n t_{LS}(J_n). \tag{11}$$

Lemma 3 follows the general principle as that for Lemma 2 and thus can guarantee the schedulability of the job set. However, a straightforward implementation of Lemma 3 has computational complexity as $O(N^2)$, where N is the total number of jobs. For a periodic job set, N is the number of all the jobs within the *least common multiple* (LCM) of the periods. It can be considerably large, especially when the periods of the tasks are co-prime. Therefore, due to its complexity, this strategy cannot be applied on-line or can be extremely costly for large periodic task sets.

Several observations help to reduce the computational cost significantly in the computation of the latest starting time for a job set. We use Figure 2 to illustrate these observations. Figure 2(a) shows a J_4-job set (the same job set as that shown in Figure 1(a)). Recall that the jobs with speed requirement higher than s_{th} cannot be delayed at all, the processor must be activated no later than the earliest arrival of such jobs. Moreover, the processor must also be waken up before the deadline for any one of the incoming jobs. Therefore, we can immediately set up an upper bound for the LST of the job set. We call this bound as the *delay bound*, and denoted as T_B. Specifically, we have

$$T_B = \min_n (d_n, \min(r_i)), \ \textit{where } s_i > s_{th}. \tag{12}$$

For example, from Figure 2(a), we have $T_B = 11$. Since the execution of the job set cannot be delayed over T_B, the execution of at least one of the jobs that arrive before T_B must be started before T_B. Therefore, one intuitive method is to use the minimum of the latest starting times of these jobs as the LST for the job set. Unfortunately, computing LST in this way may cause jobs to miss their deadlines. For example, with the job set in Figure 2(a), we have $t_{LS}(J_1) = 9$, $t_{LS}(J_3) = 9$, $t_{LS}(J_4) = 11$. However, as shown in Figure 2(b), if we let LST be 9, J_2 will miss its deadline. This is because the feasibility guaranteed by Lemma 3 requires the LST for a job set be computed based on the latest starting time for *all* jobs in the job set. More careful analysis must be performed to identify the maximal delay that a job set can tolerate based on only a sub set of these jobs.

3.3 Determining LST based on a partial job set

Lemma 3 ensure the validity of LST of the job set by examining the latest starting time for every job. If we want to identify the LST by examining the latest starting time for a limited number of jobs, these *latest starting times* need to guarantee not only the feasibility of these jobs themselves

but also that of the jobs that are not examined.

Note that equation (10) guarantees the schedulability of job J_n by requiring J_n be finished *at* its deadline. However, even though J_n can meet its deadline, other lower priority jobs may miss their deadlines if J_n completes exactly at d_i. For example, as seen in Figure 2(b), J_3 has deadline of $d_3 = 23$. However, assuming J_3 to finish at $t = 23$ will block the execution of the lower priority job, *i.e.*, J_2, during the interval $[22, 23]$ and cause it to miss its deadline. This indicates that to guarantee the deadlines for the lower priority jobs, a higher priority job may need to finish much earlier than its deadline. In addition, since the deadlines of the jobs arriving earlier than T_B can be guaranteed by equation (10), only the jobs arriving later than T_B may possibly miss their deadlines. In what follows, we want to identify when a higher priority job has to be finished, *i.e., the effective deadline*, such that other lower priority jobs arriving later than T_B can also meet their deadlines. Specifically, we have the following definition.

Definition 3. Let $J_n \in \mathcal{J}$ be any job that arrives before T_B and let s_n be the required speed for J_n to meet its deadline. The *effective deadline* for J_n is the time $t = d_n^*$ such that if J_n finishes by d_n^*, J_n and all the jobs that arrive later than T_B can meet their deadlines.

Given the fact that, based on the DVS voltage schedule that optimizes the dynamic power consumption, all the jobs can meet their deadlines. One intuitive strategy to determine the effective deadline is to use the completion time of a job based on this voltage schedule. A job completes at its effective deadline defined in this way will ensure that it will not cause any lower priority job to miss its deadline. Accordingly, Figure 2(c) shows the effective deadlines for the jobs arriving earlier than T_B, i.e., J_1, J_3, J_4. Since the schedulability of all the jobs can be guaranteed as long as all the jobs complete before their completion times according to the DVS voltage schedule, we can therefore apply Lemma 3 and replace d_i with d_i^* in equation (10) to estimate the latest starting time for the job set. Figure 2(c) shows the LST computed based on the jobs arriving earlier than $T_B = 11$, *i.e.*, J_1, J_3, J_4. It can be readily verified that all the jobs can meet their deadlines.

Recall that the jobs with required speeds less than s_{th} are executed with s_{th}. This implies that the effective deadline for a job can be in fact much later than that computed using the strategy stated above, especially when the threshold speed is much larger than the required speeds for the jobs. According to equation (10), the bigger the effective deadlines are, the later the job set can be delayed. Delaying job set not late enough may result in small intervals that are not effectively merged. For example, two idle intervals appear in the actual schedule as shown in Figure 2(c). A better strategy, as shown in Figure 2(d), can merge all the idle intervals into one single interval. In what follows, we present a strategy to more accurately estimate the effective deadline for each job. The strategy takes consideration of the speed requirements as well as the threshold speed and therefore can achieve a better result. Algorithm 2 illustrate the general idea of this strategy.

In Algorithm 2, the completion times (line 5) for the jobs can be obtained by a linear scanning algorithm. Since we only need to determine the completion time for the jobs arriving earlier than T_B, we can perform the linear scanning from the interval of the earliest arrival time to the maximal deadline of these jobs, which has a complexity of $O(N')$, where N' is the total number of jobs within this interval. The complexity of the rest of the algorithm is also linear related to N'. Since N' is usually very small for a periodic task set, Algorithm 2 has a very low computation complexity. To demonstrate the effectiveness of Algorithm 2, we have the following Lemma.

Algorithm 2 Compute the effective deadline of a job

1: **Input:** $\mathcal{J}_n$, s_i, $i = 1, 2, ..., N$ according to the optimal DVS voltage schedule [20], T_B, and s_{th};
2: **Output:** the effective deadline of J_n, *i.e.,* d_n^*
3: Let $s_k^* = \max(s_k, s_{th}), k = 1, ..., N$;
4: Let $\delta_k = (\frac{c_k}{s_k} - \frac{c_k}{s_k^*}), k = 1, ..., N$;
5: Let $f_k, k = 1, ..., N$ be the completion time of J_k according to the DVS voltage schedule;
6: $d_n^* = d_n$;
7: **for** J_i with $T_B < r_i < d_n^*$ and $d_i > d_n$ **do**
8: **if** $s_n > s_i$ **then**
9: continue;
10: **else if** $s_n < s_i$ **then**
11: **if** $d_n^* < d_i$ **then**
12: $d_n^* = \min(d_n^*, r_i + \delta_i)$;
13: **end if**
14: **else**
15: $//s_n = s_i$;
16: $d_n^* = \min(d_n^*, \max(f_n + \delta_i, r_i + \delta_i))$;
17: **end if**
18: **end for**

LEMMA 4. *Given a J_n-job set $\mathcal{J}_n$, the schedulability of J_n and that of all jobs that arrives later than T_B can be guaranteed if J_n finishes no later than d_n^*, the effective deadline computed with Algorithm 2.*

PROOF. To prove this Lemma, we first briefly review the algorithm, LPEDF [20], that constructs the DVS voltage schedule. LPEDF generates the DVS voltage schedule by iteratively identifying the *critical intervals*, *i.e.*, an interval within which the highest speed is required so that the jobs within this interval can be executed with no idle time. According to [20], the speeds for the critical intervals are monotonically decreasing during each iteration. Moreover, a critical interval can only be completely contained in another critical interval when it has a higher speed, or no two critical intervals can intersect each other. Algorithm 2 identify the effective deadline for a job based on the critical interval it belongs to. For ease of our presentation, we use I_n to denote the critical interval that contains J_n .

Given $J_n \in \mathcal{J}_n$, for any other J_i with $T_B < r_i < d_n$, if J_i has a higher priority than that of J_n, delaying the execution of J_n will not cause J_i to miss its deadline. Thus, we only consider the schedulability of J_i when J_i has a lower priority than that of J_n. Given $J_i \in \mathcal{J}_n$ with $T_B < r_i < d_n$ and with priority lower than J_n, we have three possibilities, *i.e.*, $s_n > s_i$, $s_n < s_i$ and $s_n = s_i$. We consider each case separately.

- $s_n > s_i$.

  According to LPEDF, the starting point of I_i cannot be earlier than the deadline of J_n. Since J_i can be schedulable as long as its execution starts no later than

the starting point of corresponding critical interval, the effective deadline of J_n can be as late as its original deadline (line 8-9).

- $s_n < s_i$.

  If I_n and I_i do not intersect, according to LPEDF, the ending point of I_n can be no later than r_i. Therefore, J_n needs to be finished before the arrival of J_i or it may block the execution of J_i and cause J_i to miss its deadline. On the other hand, if J_i is to be executed with s_{th} which is higher than s_i, the corresponding slack δ_i for executing J_i (defined in line 4) can be exploited by J_n as J_n has a higher priority than J_i. Therefore, J_n can be completed no later than $r_i + \delta_i$ (line 12).

- $s_n = s_i$.

  I_n and I_i may be the same or different intervals. Two strategies can be applied to ensure that the execution of J_n will not cause J_i to miss its deadline: one is to assume that J_n completes before the arrival of J_i, the other is to assume that J_n finishes before the *completion time* according to the DVS voltage schedule by LPEDF. In addition, when J_i is executed with s_{th} which is higher than s_i, J_n can be further delayed δ_i as shown in line 16 of Algorithm 2.

□

Figure 2(d) shows the effective deadlines for the jobs that arrives earlier than T_B. Note that the newly identified effective deadlines for J_1 and J_4 are larger than the ones shown in Figure 2(c). With these effective deadlines, we can therefore compute the corresponding latest starting times for the jobs arriving earlier than T_B, and take the minimum as the latest starting time for the job sets. Specifically, we have the following theorem.

THEOREM 1. *Given job set $\mathcal{J}$, the required speed s_n for each job J_n, delay bound T_B, and the threshold speed s_{th}, then the execution of $\mathcal{J}$ can be delayed to $\tilde{T}_{LS}(\mathcal{J})$ with no job missing its deadline, where*

$$\tilde{T}_{LS}(\mathcal{J}) = \min(T_B, \min_{J_i \in \mathcal{J}_s}(d_i^* - \sum_{J_k \in hp(J_i)}(\frac{c_k}{s_k^*})) \quad (13)$$

where $\mathcal{J}_s$ consists of jobs with arrival times earlier than T_B, $s_k^* = \max(s_{th}, s_k)$, d_i^* is computed based on Algorithm 2 and $hp(J_i)$ are the jobs with the same or higher priorities than that of J_i.

PROOF. Let $\mathcal{J}$ be a J_n job set. Then $J_n \in \mathcal{J}_s$. Let

$$\tilde{t}_{LS}(J_n) = d_n^* - \sum_{J_i \in hp(J_n)} \frac{c_i}{s_i^*}. \quad (14)$$

Since $\tilde{T}_{LS}(\mathcal{J}) \leq \tilde{t}_{LS}(J_n)$, J_n and all jobs that arrive later than T_B can meet their deadlines. For any job J_i $(i \neq n)$ that arrives before T_B, since $\tilde{T}_{LS}(\mathcal{J}) \leq \tilde{t}_{LS}(J_i)$, the schedulability of J_i is guaranteed. Therefore, $\tilde{T}_{LS}(\mathcal{J})$ can guarantee the deadlines of all the jobs. □

Figure 2(d) shows latest starting time for the job set computed according to Theorem 1, as well as the actual schedule. As shown in Figure 2(d), with a better estimation of the effective deadlines, all the scattered idle times are merged into one single idle interval. In next section, we use experiments to demonstrate the energy efficiency of our approach.

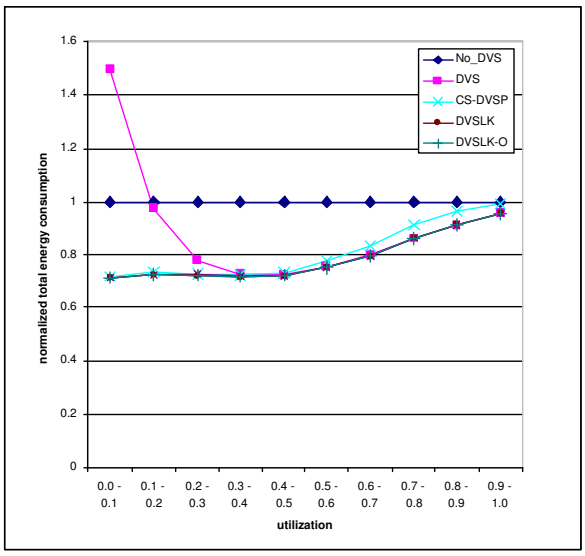

Figure 3: The average total energy consumption by the different approaches.

4. EXPERIMENTAL RESULTS

In this section, we evaluate the energy saving performance of our approach with simulations. Specifically, we compare the energy saving performance for the following five approaches (We do not compare our approach with **LC-EDF** [13] since it is proved [11] that **CS-DVSP** outperforms **LC-EDF**).

- **No-DVS** The task sets are scheduled with the non-DVS scheme, *i.e.*, all jobs are executed using the highest possible speed, and the processor is shut down when the idle interval is long enough.

- **DVS** The task sets are scheduled with pure DVS according to [20]. There is no consideration of either the leakage (i.e., the threshold speed) or delaying the execution of task instances.

- **CS-DVSP** This is the approach presented in [11]. The task instances from the same task have the same processor speed. A task instance with speed less than the threshold speed is executed with the threshold speed. When processor is in idle, the execution of later task instances may be delayed.

- **DVSLK** The task sets are scheduled with DVS according to [20] with consideration of both the leakage(*i.e.* the threshold speed) and job execution delay. The maximal delay for the task set is computed based on Theorem 1.

- **DVSLK-O** Same as that for **DVSLK**. The only difference is that the maximal delay for the task set is computed based on Lemma 3.

The power model and technology parameters of the processor used in our simulation were adopted from [15] (Based on this processor model, the threshold speed is around 0.4 [11]).

For the processor power down/up overhead, we used the same values as that used in [11], *i.e.*, $P_{idle} = 240mW$, $E_o = 483\mu J$, and $t_o = 2ms$. The periodic task sets tested in our experiments were randomly generated with the periods and the worst case execution times of these tasks randomly chosen in the range of $[10ms, 50ms]$ and $[1ms, 10ms]$, respectively. The deadlines of the tasks were set to be 80% of their periods. To investigate the energy performance of different approaches under different workload, we divided the total utilization into intervals of length 0.1 and randomly generate 50 feasible task sets for each interval. The average energy consumptions within each interval was normalized to that by No-DVS.

Figure 3 shows the average total energy consumptions by different approaches. It is interesting to see that in Figure 3, the *DVS* approach (**DVS**) may actually increase the total energy consumptions when the utilization is low. For example, when the utilization is less than 0.1, the overall energy consumption of that by **DVS** can be as high as 1.5 times of that by **No-DVS**. This is because, when the utilization is low, the processor is running at a very low speed and consumes more energy due to the large leakage current. This result clearly demonstrates that reducing the dynamic power consumption may not effectively save the overall energy consumption for the embedded system as technology continues to evolve. Leakage power consumption must be taken into consideration to make overall power efficient system.

On the other hand, as shown in Figure 3, when the utilization is high, the energy consumption by DVS is very close to that by DVSLK and DVSLK-O. This is because when the utilization is high, most of the processor speeds, which optimize the dynamic power consumption, become higher than the threshold speed. Therefore DVS, DVSLK, and DVSLK-O tend to use the same processor speeds in most cases. Moreover, due to the optimum of the DVS algorithm [20], there are much less idle periods that can be exploited to further save the energy.

With the scaling of IC technology and reduction of the supply voltage, the threshold speed of a processor can be very close to its maximal supply voltage. A large number of idle intervals may be generated even for task sets with high utilizations as higher-than-necessary processor speeds are applied for these tasks. Moreover, with the dramatic increasing of the leakage, the energy consumption during the processor idle time will soon become a significant part of the overall energy consumption. We are therefore interested in investigating how our approach can help to reduce this part of energy consumption compared with other approaches.

Figure 4 shows the average idle energy consumptions by four different approaches, *i.e.*, **No-DVS**, **CS-DVSP**, **DVSLK**, and **DVSLK-O**. Note that, our approach, *i.e.* **DVSLK** can always lead to better average idle energy savings than the previous approach **CS-DVSP**. As seen in Figure 4, **DVSLK** can save as much as 80% idle energy compared with **CS-DVSP**. This is because that, with a job-based analysis, the delay for the job set can be more accurately identified, and therefore the smaller idle intervals can be more effectively merged. Moreover, we can see that the energy savings obtained by **DVSLK** are very close to that by **DVSLK-O**. **DVSLK-O** determines the maximal delay for a job set by computing the latest starting time for all the jobs in the job set, which is not always viable in practice. **DVSLK**, on the other hand, can achieve almost the same energy saving performance but with a greatly reduced computation complexity. In general, our experiment results demonstrate that our proposed approach is not only effective but also computation efficient, and therefore, has a great potential for future embedded system in reducing both the dynamic and static power consumption.

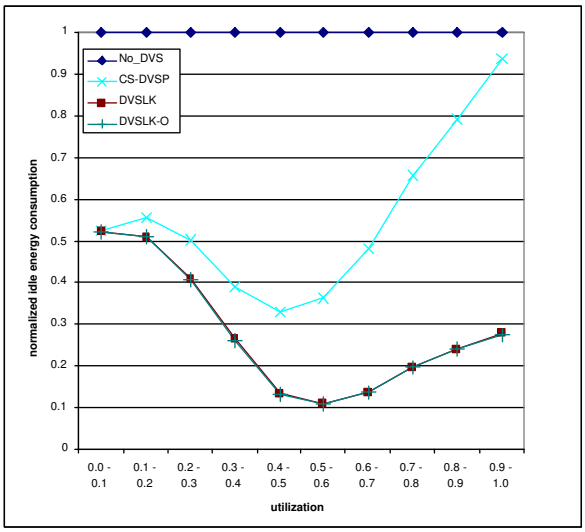

Figure 4: The average idle energy consumption by the different approaches.

5. CONCLUSIONS AND FUTURE WORK

Power efficient technology is the critical enable technique in the design of future real-time embedded systems. As the IC technology continues to evolve, leakage power consumption is becoming a more and more significant part of the overall power consumption. In this paper, we investigated the problem of applying scheduling techniques to reduce both the dynamic and leakage energy consumption.

As demonstrated in our experiments, applying the DVS based voltage schedule without the consideration of leakage power consumption cannot effectively reduce the overall energy consumption, and may even increase the total energy consumption for the embedded systems. A leakage power conscious DVS voltage schedule may require the processor to adopt a speed higher-than-necessary to avoid the rapidly increasing leakage current at low voltage level. This may result in a large number of small idle intervals during task executions. We proposed an efficient approach to merge these idle intervals by delaying the execution of task instances so that the processor shutdown overhead can be reduced and the overall energy performance can be improved. Our experiments show that our approach can outperform the related previous approach as much as 80% in reducing the idle energy consumption.

Finally, it is worth mentioning that our approach is a greedy approach. It is greedy in the sense that the job set is always delayed as late as possible which might not be globally optimal for the overall energy reduction. How to achieve the optimal overall energy performance is yet another very interesting problem and needs further study.

6. REFERENCES

[1] F. Assaderaghi, D. Sinitsky, S. A. Parke, J. Bokor, P. Ko, and C. Hu. Dynamic threshold-voltage mosfet (dtmos) for ultra-low voltage vlsi. *IEEE Trans. on Elec. Dev.*, 44(3):414–422, Mar 1997.

[2] H. Aydin, R. Melhem, D. Mosse, and P. Alvarez. Dynamic and aggressive scheduling techniques for power aware real-time systems. *RTSS*, 2001.

[3] B. H. Calhoun, F. A. Honore, and A. Chandrakasan. Design methodology for fine-grained leakage control in mtcmos. *ISLPED*, pages 104–109, 2003.

[4] A. P. Chandrakasan, S. Sheng, and R. W. Brodersen. Low-power cmos digital design. *IEEE Journal of Solid-State Circuits*, 27(4):473–484, April 1992.

[5] S. Duarte, Y. Tsai, N. Vijaykrishnan, and M. Irwin. Evaluating run-time techniques for leakage power reduction. *VLSID'02*, 2002.

[6] J. Halter and F. Najm. A gate-level leakage power reduction method for ultra low power cmos circuits. *CICC*, pages 475–478, 1997.

[7] Intel. *PXA250 and PXA210 Applications Processors Design Guide*. Intel, 2002.

[8] S. Irani, S. Shukla, and R. Gupta. Algorithms for power savings. *ISDA*, 2003.

[9] T. Ishihara and H. Yasuura. Voltage scheduling problem for dynamically variable voltage processors. *ISLPED*, pages 197–202, August 1998.

[10] ITRS. *International Technology Roadmap for Semiconductors*. International SEMATECH, Austin, TX., http://public.itrs.net/.

[11] R. Jejurikar, C. Pereira, and R. Gupta. Leakage aware dynamic voltage scaling for real-time embedded systems. *DAC*, pages 275 – 280, 2004.

[12] M. Johnson, D. Somasekhar, and K. Roy. Leakage control with efficient use of transistor stacks in single threshold cmos. *DAC*, pages 442–445, 1999.

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

[14] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard real-time environment. *Journal of the ACM*, 17(2):46–61, 1973.

[15] S. Martin, K. Flautner, T. Mudge, and D. Blaauw. Combined dynamic voltage scaling and adaptive body biasing for lower power microporcessor under dynamic workloads. *ICCAD*, 2002.

[16] B. Mochocki, X. Hu, and G. Quan. A realistic variable voltage scheduling model for real-time applications. *ICCAD*, 2002.

[17] C. Neau and K. Roy. Optimal body bias selection for leakage improvement and process compensation over different technology generations. *ISLPED*, pages 116–121, 2003.

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

[19] L. Yan, J. Luo, and N. K. Jha. Combined dynamic voltage scaling and adaptive body biasing for heterogeneous distributed real-time embedded systems. *ICCAD*, 2003.

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