Power-Aware Scheduling for Pinwheel Task Model

Da-Ren Chen\footnote{Dept. of Information Management Hwa Hsia Institute of Technology}, Chiun-Chieh Hsu\footnote{Dept. of Information Management National Taiwan University of science and Technology} You-Shyang Chen\footnote{Dept. of Computer Science & Information Engineering Hwa Hsia Institute of Technology} and Shu-Ming Hsieh

\{danny, ys_chen, Stanley\}@cc.hwh.edu.tw, cchsu@mail.ntust.edu.tw

Abstract

Dynamic voltage scaling (DVS) is a standard technique for controlling the power consumption of embedded systems. A DVS processor can vary its operating voltage and frequency during runtime by using CMOS circuits whose energy consumption has quadratic dependency on the supply voltage. In this paper, we analyze the problem of DVS scheduling for pinwheel tasks based on Distant-Constraint Task Systems (DCTS). We propose a deterministic scheduling algorithm using smooth voltage-adjusting (SVA) technique that reduces many tasks' speeds to a level by sharing the same slack. SVA gathers new scheduling information off-line for distributing the slack to more tasks than other methods do. Additionally, SVA algorithm does not need a canonical schedule whose length usually depends on the least-common-multiplier (LCM) of task periods, thus tasks can be scheduled on-line only with linear time complexity. A large number of experimental results reveal that SVA algorithm outperforms DRA and AGR in energy-saving and time complexity.

1 Introduction

For hard real-time tasks with scarce energy resources, systems are required to reduce power consumption while still meeting the task timing constraints. This can be accomplished by changing processor speed and voltage online, thereby saving energy by spreading run cycles into idle time \cite{15}. Many researchers point out that many processors can run at a range of possible speeds \cite{5,7}. For example, the Crusoe processor \cite{16} is able to dynamically change clock frequency in 33 MHz steps and up to 200 frequency/voltage changes per second.

A slack is defined as unused processor time due to early completion of a job. Many real-time scheduling algorithms, which reclaim dynamically the available slack, must produce a canonical (standard) schedule before performing a real schedule. To generate a time interval of such schedule, the time and space complexities depend on the least common multiplier of task periods \cite{2,3,11,12}, even if they consider only tasks’ arrival/preemption times.

Pinwheel task systems are first motivated by the performance requirements of satellite-based communications \cite{4}. A pinwheel task \(T_i\) is defined by two positive integers, an execution requirement and a window length, with explanation that the task \(T_i\) need to be allocated to the shared resource for at least \(a\) out of every \(b\) consecutive time units. Additionally, pinwheel scheduling is also applied in the channel assignment policies with buffer and preemptive priority for RT traffics. In the distance-constrained task systems (DCTS), the temporal distance between any two consecutive executions of a job should always be less than a certain value. In DCTS, pinwheel tasks transform the distance-constraints into \(2^n\) multiples of other shorter periods \cite{8,9}, which is not longer than its original distance- constraints by using algorithm \(Sr\) \cite{6}. The advantage of the period transformation is that the produced schedules have regular start, preemption and finish times, and therefore provide good predictability. Suppose, in Figure 2(a), a system have five tasks with distance constraints (periods) \{9.2, 10.6, 21.2, 22.6, 23.4\}, and their execution times are \{1.0, 1.1, 9.98, 0.94, 1.87\}, respectively. After applying \(Sr\), the new task periods \{5.3, 10.6, 21.2, 21.2, 21.2\} are illustrated in Figure 2(b). Because the periods are harmonic numbers, the schedule for each task has no jitter \cite{8}, and its relative starting and ending times are fixed.

In this paper, we focus on the intertask dynamic voltage scaling algorithms for periodic real-time task systems \cite{2}, given that the processor speed can be changed over a continuous range between a pre-defined lower bound and upper bound \cite{2,3,17}. We propose time-efficient and energy-efficient real-time scheduling algorithms, and discuss their performance against other intertask DVS algorithms proposed in \cite{2,3}. The proposed method called Smooth Voltage Adjusting (SVA) can be divided into two parts.
(1) Due to the period specialization [9], we derive new task parameters from the pinwheel schedule using an off-line algorithm in $O(n \log n)$ time, where $n$ denotes the number of tasks in the given task set.

(2) When an early completion takes place in a task, the proposed task parameters before this task deadline and unused execution time are utilized by an on-line scheduling. It evaluates the other tasks and decides which jobs of these tasks can properly share the unused execution, and their speed can be reduced to the same voltage level.

The required time complexity is $O(n)$. In the conventional schemes [2,3,13,14], the length of a canonical schedule depends on the least-common-multiple of task periods. However, this is not desirable because the LCM schedule is of exponential length in general, especially when tasks join and leave the systems frequently, SVA generates necessary tasks information such that the future execution of each task can be predicted rapidly and therefore enables more tasks to share the slack time produced by early-completed tasks. The key motivations of SVA are summarized as follows:

1. **Less frequent voltage scaling and utilization of slack:** In SVA, many successive jobs can be reserved for decreasing simultaneously their execution speed to the same voltage/ speed level and therefore the power consumption can be decreased.

2. **Time-efficient technique:** the execution of each task can be profiled quickly without creating a canonical schedule.

3. **Smooth fluctuations of voltage scaling:** the slack will serve as a long time-interval in the schedule as possible for preventing severe fluctuation of voltage scaling.

The remainder of this paper is organized as follows. Section 2 summarizes the background information and the notational conventions. Our power-aware scheduling algorithms are then presented in Section 3. Section 4 includes experimental results, which presents the time-efficiency and energy-efficiency of the proposed real-time scheduling algorithms. Finally, concluding remarks are given in section 5.

2 Task Model

In this section, model assumption and notations are introduced. We focus on synchronous, preemptive and hard real-time task systems; whenever a new task enters the system, all tasks have to be synchronized at a certain time (say $t=0$). In a periodic task set $T=\{T_{1}, \ldots, T_{n}\}$ of $n$ periodic tasks, each task $T_{i}$ consists of a sequence of jobs $J_{i1}, J_{i2}, \ldots$. A periodic task $T_{i}$ with an execution requirement $T_{i}$.e and a period $T_{i}.p$ has weight $w_{i}=T_{i}.e/T_{i}.p$, where $0<w_{i}<1$. The release time and deadline of each task $T_{i}$ are denoted as $r_{i}$ and $d_{i}$, respectively. Because the first invocation of each task starts at time 0, the release time and deadline of $J_{i}$ are derived from $r_{i}=(i-1)*T_{i}.p$ and $d_{i}=iT_{i}.p$, respectively. In the task model, every task period has been transformed (specialized) as a harmonic number by algorithm $Sr[6]$. Each task $T_{i}$ has a unique priority index $p_{i}^{*}$ (low values denote high priorities) and is scheduled according to preemptive RM policy. A task set $\tau$ contains all tasks in the system and its subsets $\tau_{i}=\{T_{i} \mid T_{i}.p=T_{i}.p$ and $p_{i}<p_{j}^{*}\}$, $\tau_{i}=\{T_{i} \mid p_{i}^{*}>p_{j}^{*}\}$ and $\tau_{i}=\{T_{i} \mid T_{i}.p=T_{i}.p$ and $p_{i}>p_{j}\}$. $U$ and $U_{i}$ denote the total weights of a task set before and after the period specialization [9], respectively. $T_{min}$ and $p_{max}$ denote the highest priority task and the shortest period length among the tasks in $\tau$, respectively. We define the task information of $T_{i}$: $S_{i}$: actual starting time with respect to $r_{i}$, $E_{i}$: actual ending time with respect to $r_{i}$, $Fit_{i}$: the first preemption time with respect to $r_{i}$, $re_{i}(b, \ell)$: the resume execution interval of $T_{i}$ that starts at time $b$ and is with length of $\ell$. $PI\_{i}(b, \ell)$: higher-priority tasks that give rise to preemption interval starting at time $b$ and is with length of $\ell$. $RE_{i}$: a set of $re_{i}(b, \ell)$. $HE_{i}$: the first resume execution of $T_{i}$, i.e., $\{re_{i}(b, \ell) \mid b=S_{i}\}$. $TE_{i}$: the last resume execution of $T_{i}$, i.e., $\{re_{i}(b, \ell) \mid b=E_{i}\}$. $MaxRE_{i}$: the longest length of $re_{i}(b, \ell)$ in $RE_{i}$. $MRE_{i}$: the shortest length of $re_{i}(b, \ell)$ in $RE_{i}$ and excludes the length of $TE_{i}$. $MaxPI\_{i}$: the longest length of $PI\_{i}$ between $[S_{i}, E_{i}]$.

![Figure 1](image-url) Figure 1: The (a) an original RM schedule (b) DC schedule according to RM policy and (c) schedule produced by SVA algorithm.
shown in Figure 1(b), since the lengths of task periods are harmonic numbers, each job of task $T_i$ has fixed starting time $S_i$ and ending time $E_i$ with respect to its job release time. During runtime of a task $T_i$, it could be split into several parts by higher-priority tasks. These parts are classified as preemption interval (PI), head execution (HE), resuming execution (RE) and tail execution (TE), where $TE_i$ is also the last RE in each job. Since the lengths of $re$ and $PI$ are fixed in each job of $T_i$, the lengths of $MaxRE_i$, $MRE$, and $MaxPI_i$ of the jobs are also fixed.

The power/speed model is similar to those proposed in [3]. The current processor speed of $T_i$ is denoted by $p_i$. The initial (default) speed of $T_i$ is denoted by $p_i^0$. In the beginning of every task, the operating system sets $p_i^0 = p_i$, prior to any dynamic speed adjustment. $EC_{i,j}$ denotes the early completion time occurred in job $J_{i,j}$. The power consumption of the CPU in the speed $p_i$ is given by $P(p_i)$, which is assumed to be a strictly increasing convex function denoted by a polynomial function of the third degree. The energy consumed by the processor during the time interval $[t_1, t_2]$ is $E(t_1, t_2) = \int_{t_1}^{t_2} P(p(t))dt$. Given that the processor speed can be changed $\rho$ over time interval $[t, \tau]$, denoted by a polynomial function of the third degree.

The power consumption of the CPU in the speed $p_i$ is given by $P(p_i)$, which is assumed to be a strictly increasing convex function denoted by a polynomial function of the third degree. The energy consumed by the processor during the time interval $[t_1, t_2]$ is $E(t_1, t_2) = \int_{t_1}^{t_2} P(p(t))dt$. Given that the processor speed can be changed $\rho$ over time interval $[t, \tau]$, denoted by a polynomial function of the third degree.

### 3 Smooth Voltage Adjusting Scheme (SVA)

Before presenting SVA scheme, the following notations are produced in runtime.

- $EC_{i,j}$: the absolute early completion time occurred in job $J_{i,j}$.
- $RE_{-counter}_{i,j}$: the number of $re_i$ that has elapsed before $EC_{i,j}$.
- $total_{RE}_{i}$: the total number of $re_i$ in every job of $T_i$.
- $rest_{RE}_{i,j}$: the number of the remaining $re_{i,j}$ after $EC_{i,j}$ is at least $rest_{RE}_{i,j} = \max\{0, total_{RE}_{i} - RE_{-counter}_{i,j}\}$.
- $rest_exec_{i,j}$: denotes the rest of execution time of $J_{i,j}$ after time $\tau$, where $EC_{i,j} = \tau$.

For example, in Figure 4(b), suppose $t = 10.6$, and $EC_{3,1} = 4.0$. $rest_exec_{10.6} = 9.98 - 6.9 = 3.08$.

When $EC_{i,j} = \tau$, the rest of the slack time before $r_{i,j} + E_i$ can be split into two parts:

- **before next $p_{\min}$ ($bnP_{i,j}$)**: denotes the length of time interval $[EC_{i,j}; \left[ \frac{EC_{i,j}}{p_{\max}}\times p_{\max} \right]$.

- **after next $p_{\min}$ ($anP_{i,j}$)**: denotes the length of slack that falls between $bnP_{i,j}$ and $r_{i,j} + E_i$.

which can be defined as $T_{i,j} = \left[ \frac{EC_{i,j}}{p_{\max}} \times p_{\max} \right]$, $r_{i,j} + E_i$.

In Figure 1(c), given $EC_{3,1} = 4.0$, $RE_{-counter}_{3,1} = 0$ and $rest_{RE}_{3,1} = 2$, because $EC_{3,1}$ occurred in $HE_i$. Moreover, $bnP_{3,1}$ denotes the length of $[4.0, 5.3]$, which is $1.3$, and $anP_{3,1}$ denotes the rest of the slack in $J_{1,i}$, which can be rewritten as $9.98 - \left[ \frac{4.0}{5.3} \times 5.3 \right] - (0+2.7) = 7.38$. In general, $anP_{i,j}$ could be utilized by the tasks in $\tau - r_{i,j} - T_i$ and $bnP_{i,j}$ can be applied to the tasks in $r_{i,j} \cup r_{i,j} - T_i$. The use of $bnP_{i,j}$ and $anP_{i,j}$ will be discussed in section 3.3 and 3.4, respectively.

![Figure 2. SVA framework.](image)

**Figure 2. SVA framework.**

SVA framework is composed of four independent procedures introduced in the following subsections. In Figure 2, SVA is triggered by early-completed jobs and decreases the speed of other tasks on-the-fly in order to provide additional power savings and meet the task deadlines.

In Figure 3, procedure $reduce_{speed}(e^a, L, \rho_{k,p})$ utilized by SVA can reduce the speed $\rho_{k,p}$ of $J_{k,p} \in \delta$ by allocating $L$ extra units of CPU time for extending the total length of job executions $e^a$. Thus it decreases the speed of $J_{k,p}$ and is subject to $\rho_{\min}$. This algorithm is quite different from that in [2]. SVA decreases the speed of each $J_{k,p} \in \delta$ to the same speed level, where $\delta$ denotes a job set containing the feasible jobs varying with time. Consequently, SVA reduces the frequency of speed adjustments, and saves more energy than the previous work does.

![Procedure reduce_speed](image)

**Figure 3. Procedure reduce_speed.**

3.1 $\text{Initialized}_{Sys}(\tau)$

The procedure shown in Figure 4 is required in scheduling during system initialization or new tasks arrival, whose time complexity is given in
Lemma 1. Step3 in this procedure computes relative starting and ending times of each task.

Procedure Initial_Syst(t)
Step 1: Sort the tasks in t with respect to their periods, such that $T_r:p_T:p_S < \leq S_r:p_T:p_S < \leq \ldots < \leq p_T:p_S$.
Step 2: By algorithm Sr [8], it transforms the lengths of tasks periods into harmonic number.
Step 3: Computes the $S$ and $E$ of every task by algorithm $Start$. $End$.
End Procedure

3.2 Early_Completion ($t$, $\tau_i$)

When task $T_i$ completes early at time $t$, SVA performs procedure Early_Completion ($t$, $\tau_i$) in Figure 5. Firstly, it computes available slack after time $t$ using Step1 and Step2. It then finds out the task sets that are suitable for sharing the slack using Step3 and Step4.

Procedure Early_Completion($t$, $\tau_i$)
Step 1: Computes the lengths of HE, MRE; and TE; of task $T_i$.
Step 2: Predicting rest_RE in $J_o$.
Step 3: Divide $r$ into two subsets with respect to $p_r$ such that $r_{\tau_i} = \{p_r > \rho_r\}$ and $r_{\tau_i} = \{p_r < \rho_r\}$.
Step 4: If $r_{\tau_i}$ is not empty
   THEN call Higher_Priority_Task($T_i$)
   IF $r_{\tau_i}$ is not empty
   THEN call Lower_Priority_Task($T_i$)
End Procedure

3.3 Lower_Priority_Task($J_{ij}$, $bnP_{ij}$)

The procedure shown in Figure 6 presents an efficient way to distribute slack $bnP_{ij}$ among the tasks in $r_{\tau_i}$.

Procedure Lower_Priority_Task($J_{ij}$, $bnP_{ij}$)
1. Let job set $\delta = \{J_{ij} | T_i < t_{\tau_i} - T_i$, and $r_{n_i} + S_i > EC_i\}.$
2. Find all job $J_{ij}$ such that $J_{ij} \in \delta$.
3. Let $e^* = \sum_{J_{ij} \in J_{ij}} T_i$.
4. For all $J_{ij} \in \delta$, $\text{reduce speed}(e^*, bnP_{ij}, SP_{ij})$.
5. For all $J_{ij} \in \delta$, $\text{set SP}_{ij} = SP_{ij}$.
6. For all $J_{ij} \in \delta$, $\text{change start time} S_i$ of $J_{ij}$ to $S_{ij}$.
7. Change the start time $S_i$ of $J_{ij}$ to $S_{ij}$.
8. Whenever the time is at $S_{ij}$ and $\delta$ is not empty
   SET $SP_{ij} = SP_{ij}$.
End Procedure

Example. In Figure 7, assuming the default task speeds $S_3=5$, $S_4=8.4$, and $S_5=9.8$. Given $EC_{2,1}=4.0$, job set $\delta = \{J_{2,1}, J_{3,1}, J_{3,1}\}$, since $r_{3,1} + S_3$, $r_{4,1} + S_3$, and $r_{5,1} + S_3$ lie between $EC_{2,1}$ and $d_{2,1}$. In line 3, the new speeds of $\rho_{3,1}$, $\rho_{4,1}$, and $\rho_{5,1}$ obtained by reduce_speed in line 4 are $0.857$, $T_{4,1}$, and $T_{5,1}$ obtained in line 5 are $3.97$, $1.634$, and $3.5$, respectively. According to line 6 and line 7, the new starting times of $J_{3,1}$, $J_{4,1}$, and $J_{5,1}$ are $S_{3} - 0.57 = 4.43$, $S_{4} - 0.234 = 8.166$, and $S_{5} - 0.5 = 9.3$, respectively. Similarily, $EC_{2,2}$ takes place at time 15, therefore $\delta = \{J_{2,2}\}$. Only the execution of $J_{3,2}$ can be extended, and thus $\rho_{3,2} = \text{speed} \text{reduce}(3.4, 0.9, 1) = 0.79$. In line 5 and line 6, the new starting times of job $J_{3,2}$ are $r_{3,2} + S_3 - 0.9 = 10.6 + 5.9 - 0.9 = 15.6$. Note that line 8 is triggered whenever a modified starting time $S_{ij}$ has been reached.

3.4 Higher_Priority_Task

When a job $J_{ij}$ completes early, the procedure distributes slack $anP_{ij}$ among the tasks having higher priorities than $p_{ij}$. The lengths of HE, TE, MRE, and MaxPI can be formulated efficiently and the correctness are discussed in Section 4. Let $\delta'$ denote the job set consisting of all jobs that release at time $t$ and are eligible for speed reduction by using $anP_{ij}$. Since $J_{ij} \in \delta'$ and $n_{ij}$ is a multiple of $p_m$, we only consider the speed-adjustment at time $t = r_{ij}$ after $EC_{i,j}$. 

Figure 4: procedure Initial_Syst().

Figure 5: Procedure Early_Completion.

Figure 6: Procedure Lower_Priority_Task($J_{ij}$, $bnP_{ij}$).

Figure 7. The illustration of Lower_Priority_Task().

Figure 8. Procedure Higher_Priority_Task($J_{ij}$, $PI_{ij}$).
In Figure 8, from line 3 to line 10, it deals with the jobs that release from time \( t \) (denotes the multiplier of \( \rho_{\text{max}} \)), and their periods do not overlap with \( TE_{ij} \). Since obtaining the actual lengths of every \( RE_{ij} \) in \( T_e \) is quite time-wasted, the lengths of \( RE_{ij} \) can be predicted more efficiently by replacing them with \( MRE_{ij} \) in line 6. From line 10 to line 16, because the actual length of the rest of \( TE_{ij} \) can be obtained by \( \text{rest}_\text{exec}_{i,j} \), the ending time of the jobs in \( \delta_{i,j}' \) can be postponed by sharing \( \text{rest}_\text{exec}_{i,j}' \).

Notably, from line 3 to line 10 in Figure 8, it will be awaked whenever the starting time \( t \) has been reached.

![Figure 9](image.png)

**Figure 9.** (a) DC schedule according to RM policy and (b) schedule produced by SVA algorithm.

**Example.** Suppose, in the Figure 9(b), the original task speeds are 1 and \( E C_{i1}=4.0 \). The actual lengths of \( anP_{3,1} \) can be derived and utilized by jobs \( J_{1,2}, J_{1,3}, J_{1,4} \) and \( J_{2,2} \) in procedure \( \text{Higher\_Priority\_Task} \).

When time \( t'=5.3 \), in line 4 of Figure 8, job set \( \delta_{3,1}'=[J_{1,2}], r_{1,2}+S_{1,2} \) and \( r_{1,2}+E_{1,2} \) are located in time interval \([5.3, 10.6] \). In line 5, \( e^* = T_{1,e} = 1.0 \), the new speed \( \rho_{1,2} \) derived by the procedure \( \text{reduced\_speed} \) in line 6 is \( T_{1,e} = \frac{1.0}{1.0 + MRE_{3,1}} = 0.278 \), where value 2.6 denotes the length of \( MRE_{3} \). Since deriving the actual length of every \( RE_{3} \) increases substantially the scheduling overhead, we use \( MRE_{3} \) to predict the length of every \( RE_{3} \) in \( T_{3} \). In line 7, since \( T_{1,e} \) is 3.60, the new ending time of \( J_{1,2} \) obtained in line 7 and line 8 is 3.60. Similarly, at time \( t'=10.6 \), we have \( \delta_{1,6}'=[J_{1,3}, J_{2,2}] \), since \( r_{1,3}+S_{1,3} \), \( r_{2,2}+S_{2,2} \), \( r_{1,3}+E_{1,6} \) and \( r_{2,2}+E_{2} \) are located within \([10.6, 15.9] \). Therefore, the new speed of \( J_{1,3} \) and \( J_{2,2} \) is \( \frac{1.0+1.7}{1.0+1.7+2.6} = 0.15 \), and their new ending times are \( E_{1,3}=1.96 \) and \( E_{2}=5.3 \), respectively. Finally, when time \( t'=15.9 \), the jobs beginning during \([15.9, 21.2] \) can share \( TE_{3,1} \), which can be derived accurately by \( \text{rest}_\text{exec}_{3,1} \).

4 Experiments

In the experiments, the original task periods are chosen randomly in the interval \([2.0, 10.000.0] \)ms. Every original task weight has a decimal fraction at most five digits. Before each simulation, the original task periods are specialized according to algorithm \( Sr \), in which the new utilization \( U \), is not greater than one. We vary three parameters in our simulations: (1) number of tasks \( \text{total\_tasks} \) in \( \tau \) from 2 to 20, (2) the probability of an early completion \( \text{prob. of EC} \) for each job, and (3) the \( \text{bcw} \) ratio of \( \text{BCET} \) to WCET, from 0.1 to 0.9. According to the different pair of \( \text{total\_tasks}, U \) and \( \text{bcw} \) in \( T \), we randomly generate \( 20,000 \) task sets. Algorithm \( Sr \) is performed for the 20,000 randomly generated task sets with variable number of tasks in the task sets. In a task set, each task period \( T_{i,p} \) is assigned randomly in the real number range \([1, 100] \)ms with a uniform probability distribution function. The WCET \( T_{i,e} \) of each task is assigned in the real number range \([1, \min\{T_{i,p}-1, 450]\})ms. After giving the values of tasks’ periods and executions, we assign the utilization \( U \) of a task set and rescale the \( T_{i,p} \) of each task such that the summation of the task weights (i.e. \( T_{i,e} / T_{i,p} \)) is equal to a given \( U \). The early completion time of each job in simulations (1) and (2) is randomly drawn from a Gaussian distribution in the range of \([\text{BCET}, \text{WCET}] \), where BC/\( \text{WCET} = 0.1 \). In simulation (3), each experiment was performed by varying \( \text{BCET} \) from 10% to 90% of \( \text{WCET} \). After generating the task set, a duplicate of the task sets is transformed by \( Sr \). Therefore, the duplicate has higher actual utilization than that initially intended. In the transformed tasks, since the simulation tasks still complete early proportionately according to the experiment settings, the slack originated from additional utilization can be utilized by slack-time
analysis algorithms. Because of the range of the period lengths and the regular nature of pinwheel DCTS, we simulate the schedules of length at most 10,000ms.

The processor model we assumed is based on the ARM8 microprocessor core. For all experiments, we assume there are 10 frequency levels available in the range of 10MHz to 100MHz, with corresponding voltage levels of 1 to 3.3 Volts. The assumption of voltage scaling overhead is the same as that in [1], and an idle processor consumes at most 500μW at the processor sleep mode. The energy consumptions of all the experimental results are normalized against the same processor running at the maximum speed without DVS technique (non-DVS). The minimum speed $\rho_{min}$ is set to 10MHz, and default speed $\rho$ is set to $U_r$. In the test bed, the actual speed $\rho_{actual}$ is selected not less than $\rho$ produced by SVA. We measure the average energy-saving per experiment using a cubic power/speed function. The occurrence of an early-completion taken place randomly between $S_i$ and $E_i$ is controlled by an exponential distribution function, where each job has at most one early-completed event.

Figure 10. Effect of variations in the totaltasks (a) tasks without period transformation and (b) tasks with period transformation ($U=60\%$, $bc/wc=0.6$)

4.1 Effect of the sizes of $\tau$

Figure 10 presents the energy consumption of the techniques varying with the sizes of $\tau$, when the probability of the early-completion for a job is 0.5. We change the values of totaltasks between 2 and 20. We have the following observations:

1. The energy consumption of AGR and DRA is rather insensitive to the variations of totaltasks. This is due to the fact that the default speed $\rho$ of a task is very similar to their $U_r$, which is generated randomly between $n \times (2^{1/n} - 1)$ and 1. Although the number of early completion job increases whenever we increase the value of totaltasks, the average generated execution time of a job has to be decreased such that $U_r \leq 1$, which compromises the effect of increased energy saving. However, SVA squeezes further energy saving from the systems with large totaltasks. There are two main reasons. Firstly, in procedure Lower_priority_Task, when an early completion takes place, it attempts to find the adequate jobs as many as possible and reduces their execution speed simultaneously. Contrary to Lower_priority_Task, DRA and AGR could reduce few tasks’ speed in the same period of time. Second, because Higher_priority_Task further utilizes the unused fragments of slack time (i.e. the MREs), the energy saving produced by our scheme gets closer to its saving bound.

2. The energy consumed by SVA is greater than that of DRA. SVA also outperforms AGR when totaltasks is greater than 8. However, from the viewpoint of system implementation, speculating an aggressive level in AGR may not be easy. In a real-time system with unstable load, such as radar systems, portal sites and pilot control systems, it cannot predict their average loading at a particular time. Even if the aggressive level can be reached, it must be updated frequently, which results in a large overhead for scheduling systems.

4.2 Effect ACET/WCET and EC

In Figure 11(a), when the prob. of EC is smaller than 0.8, SVA performs 2%~12% more energy saving compared to AGR and DRA. When prob. of EC is 0.9, the results are not as good as we expect, where there are 1% worse than those of AGR. In the experiments, the totaltasks of each task set is randomly determined from 2 to 20. As the probability of early completion decreases, the differences between SVA and DRA or AGR increase substantially. On the contrary, when most of the jobs complete early, the amount of current available slack would be changed frequently and the voltage levels of other related jobs have to be changed. The effects of frequently voltage change compromise the advantage of SVA that benefits the proper distribution of slack. In Figure 11(b), SVA still saves 1% to 10% energy compared to other methods.
The effect of bc/wc shown in Figure 12(a) and 12(b) confirms our prediction that the energy consumption \((U=0.8, \text{prob. of EC is 0.5})\) would be dependent on the variability of the actual workload. When bc/wc=0.9, the energy consumptions are quite close for all three techniques, as expected. However, once the actual workload continuously decreases, the algorithms are able to reclaim slack time and to save more energy. SVA gives the best energy saving, followed by AGR and DRA. Decreasing the ratio helps further improve the relative performance of SVA because more slack can be distributed properly among a group of future jobs. Once we decrease the bc/wc ratio less than 0.6, SVA saves greater than 8% energy consumption than those of the other two algorithms. In Figure 12(b), the energy consumption of SVA is closer to DRA and AGR than those in Figure 12(a), because transformed periods improve slack computation of these methods.

In a jitterless schedule, more jobs have the same release times and deadlines than those in the schedules with original periods do. Therefore, the appearances of NTAs become regular and the distances between NTAs are longer than those in the schedules without period transformation. At each scheduling point, when the distance between each pair of consecutive NTAs is lengthened, DRA and AGR obtain more energy savings. In addition, when more tasks release at the same time, we can predict the length of slack more precisely and easier. As the example shown in Figure 9(b), a jitterless schedule makes more jobs share available slack than those with original periods do. Although SVA has the penalty for utilization inflations, it still outperforms other methods.

5 Conclusion

SVA is a power-aware hard real-time scheduling algorithm with dynamic voltage scaling. This algorithm supports pinwheel tasks based on DCTS model. As an on-line scheduling algorithm, SVA is time-efficient and still achieves significant energy savings. The previous work in the literature [2,3,8,10,14,16] has to generate a canonical (LCM) schedule before tasks execution. Instead of generating an exponential time schedule in advance, SVA can only attain new parameters of the tasks \((HE, TE, MRE, \text{etc.})\) off-line in \(O(n \log n)\) time and performs speed reduction on-line at job’s starting time.
and early-completion times.

In addition, SVA, which agrees with the concept in [15], reduces the fluctuations of task speeds and leads to further energy saving. In Figure 16 and 17, SVA saves up to 30 percent of the energy over the static algorithm. It also offers an advantage over other state-of-the-art intertask voltage scheduling schemes such as DRA and AGR [2]. Finally, SVA outperforms DRA and AGR up to 19% and 20% percent of energy saving in the considerable size of $\tau$.

Acknowledgment

This work was supported in part by the National Science Council of the Republic of China under grant NSC 99–2221–E–146–011.

Reference