An Efficient DVS Algorithm for Pinwheel Task Schedules

Da-Ren Chen* and You-Shyang Chen**

Abstract—In this paper, we focus on the pinwheel task model with a variable voltage processor with d discrete voltage/speed levels. We propose an intra-task DVS algorithm, which constructs a minimum energy schedule for k tasks in O(d+k log k) time. We also give an inter-task DVS algorithm with O(d+n log n) time, where n denotes th e number of jobs. Previous approaches solve this problem by generating a canonical schedule beforehand and adjusting the tasks' speed in O(dn log n) or O(n^3) time. However, the length of a canonical schedule depends on the hyper period of those task periods and is of exponential length in general. In our approach, the tasks with arbitrary periods are first transformed into harmonic periods and then profile their key features. Afterward, an optimal discrete voltage schedule can be computed directly from those features.

Keywords—Hard Real-time Systems, Power-aware Scheduling, Dynamic Voltage Scaling, Pinwheel Tasks

1. INTRODUCTION

In the last decade, energy-aware computing has become widespread not only for portable and mobile devices powered by batteries, but also for large systems in which the cost of energy consumption and cooling is substantial. With dynamic voltage scaling (DVS) techniques [1, 11, 12, 14-16, 27], processors are capable of performing at a range of voltages and frequencies. Since energy consumption is at least a quadratic function of the supply voltage (hence CPU speed), the total energy consumption could be minimized by sharing slack time while satisfying the time constraints of the tasks.

For the DVS hard real-time systems, two categories of algorithms are used: inter-task DVS [1, 11, 21, 24] and intra-task DVS [5, 15, 25]. In the former case, speed assignments are determined at task dispatch or completion times. In other words, when an instance (job) of a task is assigned to a CPU, the CPU speed is not changed until it is preempted or complete. Inter-task scheduling algorithms are often implemented under operating system control, and programs do not need to be modified during their runtime. On the contrary, intra-task DVS algorithms adjust the CPU

※ This work was supported in part by the National Science Council of the Republic of China under Grant NSC 99-2221-E-146-011

Manuscript received March 31, 2011; accepted September 2, 2011.

Corresponding Author: Da-Ren Chen

* Dept. of Information Management, National Taichung University of Science and Technology, Taichung, Taiwan, R.O.C (danny@ ntit.edu.tw, danny063@gmail.com)

** Dept. of Information Management, Hwa Hsia Institute of Technology,Taiwan, R.O.C.

speed within the boundaries of a given task. Intra-task DVS techniques are under software and compiler control by using program checkpoints or voltage scaling points of the target real-time software. Therefore, it exploits all the slack time from the run variations of different execution paths and the CPU speed is gradually increased to assure the timely completion of real-time tasks. However, checkpoints have to be generated at compiling time and indicate places in the code where the processor speed and voltage should be re-calculated. They could increase the complexity of programming and the overhead of real-time systems.

Many theoretical models for DVS only consider the power consumption function with convexity [1, 21, 26, 27]. In these models, the processor must be able to run at any real-speed level in order to achieve optimality. In general, an off-the-shelf processor with variable voltages runs only at a finite number of speed levels. For example, the Intel SpeedStep® technology [28] and AMD Cool'n Quiet® [29] that are currently used in general-purpose mobile and handy devices support 3 and 5 speed levels, respectively. Therefore, an applicable model for DVS scheduling should capture the discrete, rather than continuous, nature of the available speed scale.

There are several works that have addressed the problem of task scheduling and min-energy DVS scheduling. Yao et al. [26] proposed a theoretical DVS model and an $O(n^3)$ algorithm for computing a min-energy DVS schedule in a continuous variable voltages CPU. Ishihara et al. [9] proposed an optimal voltage allocation technique using a discrete variable voltage processor. However, the optimality of the technique is confined to a single task. Kwon et al. [6] proposed an optimal discrete approach, which is based on the continuous version in [26] and therefore requires $O(n^3)$ time. The recent result proposed by Li et al. [15] gives an $O(dn\log n)$ time algorithm, which constructs a minimum energy schedule without first computing the optimal continuous schedule. In the min-energy DVS scheduling algorithms mentioned above, those techniques have to generate certain schedules in advance as the intermediate processing of their algorithms. For example, the algorithm **Bipartition** in [15] has to generate an *s-schedule* and a reversed *s-schedule* in advance. Moreover, algorithm **Alloc-vt** in [14] has to generate a min-energy continuous schedule from [26] prior, in order to perform their algorithm. Since the lengths of such schedules depend on the LCM of task periods, their algorithms could not be completed in polynomial time. Moreover, in the periodic tasks systems, the preprocessing overhead produced by these approaches may become very severe when tasks *join* and *leave* the systems frequently.

In the real-time applications, broadband 3G (B3G) wireless communication systems provide a packet-switched core network to support broadband wireless multimedia services. The resource management policies in the cell of B3G system are to guarantee the quality-of-service (QoS) of real-time (RT) traffics. To guarantee the QoS of RT traffics in a cell, many researchers [2, 13, 18, 19] proposed the pinwheel scheduling algorithms to reduce the jitter of variable bit rate (VBR) traffic in a cell. In other applications, such as the medium access control (MAC) layer of CDMA and TDMA-based wireless networks [10, 22, 23], many pinwheel scheduling schemes are proposed for solving the frame-based packet scheduling problems. These pinwheel methods provide low delay and low jitter for RT traffic and short-queue length for non-RT traffic.

In a network system, *jitter* is the variation in the time between packet arrival, which is caused by network congestion, timing drift, or route changes [16]. In a periodic task schedule, a task's jitters are often caused by the interference of other tasks. A *jitterless schedule* means that the inter-arrival times of successive instances (jobs) of a task are identical. For example, the delay and jitter control in ATM (Asynchronous Transfer Mode), a multimedia stream requires QOS

including end-to-end delay 100ms, and the bandwidth should be allocated approximately to guarantee the delay in network systems. In many real-time applications, tasks must be executed in a distance-constrained manner, rather than just periodically. C. -W. Hsueh et al. propose the *Sr*[6] algorithm to transform the lengths of periods of a pinwheel task into *harmonic,* which is shorter than or equal to the original periods. Moreover, D. -R. Chen et al. [4] give the property that each task in this type of schedule have, a constant relative beginning, and finishing and preemption times, and therefore this schedule provide a good predictability and allow for off-line scheduling optimization.

This paper discusses the theoretical power-aware real-time scheduling. We consider a discrete DVS scheduling problem for periodic task systems given worst-case execution times (WCET). We proposed an algorithm that finds a min-energy intra-task schedule in O(d+k log k) time. We also give an inter-task scheduling algorithm in O(d+$n$log n) time where k, n, and d denote the number of tasks, jobs, and voltage level, respectively. Notably, our approaches are *off-line* and are *truly* polynomial-time algorithms, which can be achieved by the following three phases.

(1) We proof that any slack time can be shared among the tasks with a transformed period.
(2) We also compute the total utilization with tasks speed s_c ($s_1 \geq s_c \geq s_d$). which is first greater than or equal to 1.
(3) Given the speed, s_c, we compute every task's features, such as the relative beginning and finishing times of every task and we adjust the speed of the tasks to prevent missing a deadline.

The rest of the paper is organized as follows: in Section 2, we give the model and the notational conventions. Section 3 presents the properties of a *jitterless* schedule and the algorithm that generates a pinwheel schedule. The DVS algorithms are proposed in Section 4. In Section 5, we present the performance analyses of the proposed algorithms and compare the utilization of transformed task sets with those of their original task sets. Section 6 concludes this paper.

2. TASK MODEL

Pinwheel task systems are first motivated by the performance requirements of satellite-based communications [6]. A pinwheel task T_i is defined by two positive integers, an execution requirement and a window length, with the explanation that the task, T_i, needs 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 many RT applications, tasks must be executed in a distance-constrained manner [17, 20], rather than just periodically. For example, in the wireless sensor network applications, a multi-sensor assessment task is invoked either periodically or is triggered by certain events. This assessment process may have one or more input tasks for collecting data from different sensors. Similar requirements exist in the phased-array radar systems, where the dwell tasks collect device data and recognize the properties of the aircrafts within certain end-to-end deadlines [8]. 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 [7, 8], which are not longer than their original distance-constraints, by using the algorithm of ***Sr*** [6].

The advantage of the period transformation is that the produced schedules have regular start, preemption, and finish times, and therefore provide good predictability.

We focus our attention on synchronous, preemptive, and periodic task systems. In the task set $\tau=\{T_1, \ldots, T_k\}$ of k periodic real-time tasks, every task T_i consists of an infinite sequence of jobs $j_{i,1}, j_{i,2}, \ldots$. A task T_i with a WCET requirement e_i and a period p_i has the weight $w_i=e_i/p_i$, where $0<w_i<1$. A feasible schedule must give each job its WCET between the arrival-time r_i and the deadline d_i. In the task model, we assume that every task period p_i and deadline has been transformed as harmonic that they have been sorted according to their periods, $p_1 \leq p_2 \leq \ldots < p_k$. Because of the jitterless schedule, the relative beginning b_i and finishing time f_i of T_i are fixed and can be efficiently obtained in Section 3.

Denote by $s_1>s_2>\ldots>s_d$ the clock speeds corresponding to d given the discrete voltage levels. The highest speed s_1 is always fast enough to guarantee a feasible schedule for given tasks. Moreover, e_i and e_i^g denote the duration of the execution at speed s_1 and s_g, respectively. The time overhead for varying the supply voltage and clock frequency is negligible. In addition, the power loss for the DC-DC converter is also negligible. Let $U_\tau^g = \sum_{i=1}^{k} w_i^g$ denote the total weight of tasks in τ at speed s_g where $w_i^g = e_i^g / p_i$. For simplicity, U_τ denotes the total weight of τ at highest speed. The power P, or energy consumed per unit time, is a convex function of the processor speed. The energy consumed by the processor during the time interval $[t_1, t_2]$ is $E(t_1, t_2)=\int_{t_1}^{t_2} P(s(t))dt$. We refer to this problem as discrete DVS scheduling (abbreviated to DVS-intra-task). The first goal is to find, for any given task set τ, a feasible schedule produced by DVS-intra-task that minimizes E. In the inter-task version, every job has only one speed during its execution. The second goal is to generate the inter-task (abbreviated to DVS-inter-task) schedules and to lower their energy consumption as that of the schedule produced by DVS-intra-task as possible.

3. JITTERLESS SCHEDULE

In this section, we introduce the concept of an h-schedule produced by *Sr* and propose its important properties. Algorithm *Sr* [6] converts the periods into a set of special periods that are not greater than the original periods, while minimizing the total task utilization increase. For example, in Figure 1(a), a system consists five tasks with periods 9.2, 10.6, 21.2, 22.6 and 23.4, and their execution times are 1.0, 1.1, 9.98, 0.94 are 1,87, respectively. After applying ***Sr***, the new task periods {5.3, 10.6, 21.2, 21.2, 21.2} are illustrated in Figure 1(b). Because the periods are *harmonics*, the schedule for each task has no *jitter*, and their relative starting and ending time are fixed.

3.1 Speed selection

In this subsection, we prove that any slack time in a jitterless schedule with harmonic task periods can be allocated to all jobs. By using this property, we can provide a unique speed for a schedule to minimize energy consumption and the times of speed adjustment.

Definition 1. An *h-schedule,* for τ which conforms to the RM policy and the lengths of the task period, are transformed into harmonics by using ***Sr*** [6].

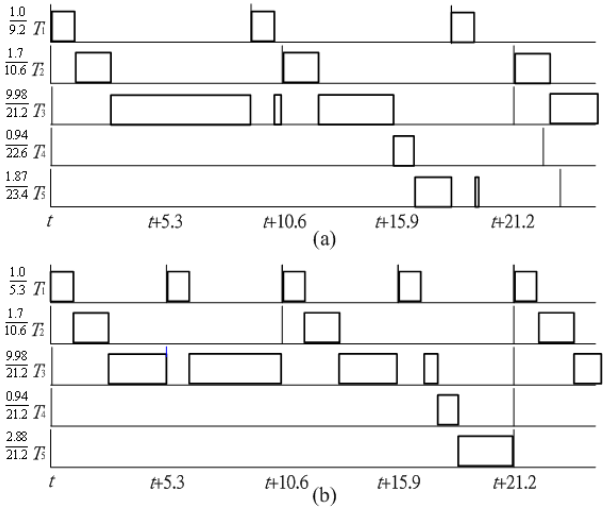

Fig. 1. (a) Periodic and (b) jitterless schedule

Without loss of generality, the length of an *h-schedule* is equal to p_k. Notably, as long as the utilization of the task set after transformation is less than or equal to 1, the task set can be feasibly scheduled.

Lemma 1. If $U_\tau \geq 1$, then there is no slack time in an *h-schedule*.

Proof. We prove it by contradiction. Let $m_i=p_k/p_i$ and $1 \leq i \leq k$, without loss of generality, we discuss *h-schedule* over interval $I=[0, p_k]$. In interval I, the total execution time of tasks in τ is denoted as $\sum_{T_i \in \tau} m_i \times e_i$. Assume that there exists slack time in interval I, the following inequality can be satisfied,

$$\sum_{T_i \in \tau} m_i \times e_i < p_k$$
$$\rightarrow \sum_{T_i \in \tau} \frac{e_i}{p_k} \times \frac{p_k}{p_i} < 1$$
$$\rightarrow U_\tau < 1.$$

This contradicts our assumption. □

Definition 2. In the *h-schedule* for τ, we define a deadline as being *tight* if task T_i is finished just on time at d_i.

Theorem 2. In an *h-schedule* for τ, slack time exists if and only if all jobs in the schedule do not miss their deadlines and have no tight deadline.

Proof. For the "only if" direction, we prove by contradiction.

Case 1: (k=1) When an *h-schedule* contains only one task which has tight deadline, all of its jobs must be finished exactly at their deadlines. Therefore, no slack time exists in the schedule.

Case 2: (k>1) In an *h-schedule*, all jobs of a task have identical relative finishing time. Without loss of generality, assuming that T_k has a tight deadline at the time of t. For all task T_x, $x<k$, have higher priorities than that of T_k. Because task periods are harmonic, we obtain

$$\frac{p_k}{p_x} = m_x,$$

m_x denotes the power of 2.

Moreover, at the time of t-p_k, there are exactly k tasks being released. They must have their jobs' deadlines at the time of t. Therefore, for all T_x, the execution time of their jobs that complete over the interval [t-p_k, t] can be written as follows:

$$\sum_{x=1}^{k} m_x \times e_x = p_k \times \sum_{x=1}^{k} \frac{e_x}{p_x}$$

Since we have assumed $f_k \geq p_k$ and $p_k \times \sum_{x=1}^{k-1} \frac{e_x}{p_x} + e_k$, we can derive

$$f_k \geq p_k \Rightarrow (p_k \times \sum_{x=1}^{k-1} \frac{e_x}{p_x}) + e_k \geq p_k$$
$$\Rightarrow e_k \geq p_k - p_k \times \sum_{x=1}^{k-1} \frac{e_x}{p_x}$$
$$\Rightarrow e_k \geq p_k (1 - \sum_{x=1}^{k-1} \frac{e_x}{p_x})$$
$$\Rightarrow \frac{e_k}{p_k} \geq 1 - \sum_{x=1}^{k-1} \frac{e_x}{p_x}$$

Without loss of generality, T_k has the lowest priority in τ and we derive $U_\tau \geq 1$. According to Lemma 1, it contradicts our assumption.

The "if" direction is easy to see. Suppose the *h-schedule* for τ contains no slack time. Without loss of generality we only discuss the jobs performed in interval I. The total execution time of these jobs can be written as: $\sum_{x=1}^{k} m_x \times e_x$. Since interval I contains no slack, we have:

$$\sum_{x=1}^{k} \frac{p_k}{p_x} \times e_x > p_k \Rightarrow U_\tau \geq 1.$$

When a U_τ>1, *h-schedule* is not feasible, there exists at least one job that is missing its deadline. When U_τ=1, the latest job in interval I must have a tight deadline. This completes the proof. □

Base on Theorem 2, as long as an *h-schedule* that is executing at a constant speed is missing a deadline or has a tight deadline, there is no wasted slack time in the schedule. We give the following definition of eligibility for CPU speed to the *h-schedule*:

CriticalSpeed(τ)

Input: task set τ
Output: critical speed s_c

$g=d$, $U_\tau^g=0$

While $U_\tau^g<1$ **and** $g>1$ **do**

{$U_\tau^g = U_\tau \times (s_1 / s_i)$;
$g=g-1$;}

return s_g.

Fig. 2. Pseudo-code of Algorithm 1

Definition 3. In an *h-schedule* for τ, we define the critical speed s_c to be the highest speed such that all tasks execute at speed s_c and $U_\tau^c \geq 1$.

The purpose of s_c produced by *CriticalSpeed*(τ) in Figure 2 is to rapidly find a suitable *h-schedule* and to reduce the number of speed adjustments.

3.2 Task execution profiling

After computing s_c, we can derive the beginning and finishing times of every task under speed s_c without constructing an actual *h-schedule.* In Figure 3, Algorithm Feature Retrieving obtains b_i and f_i of all the tasks that are in O(k) time.

FeatureRetrieving(τ)

Input: task set τ and speed level s_c
Output: a set of pairs (b_i, f_i) where $0<i\leqq k$

For i=1 **to** k **do** *//scaling all e_i as e_i^c //*

{ $e_i^c = e_i \times \frac{s_1}{s}$ }

$a_1=p_1-e_i^c$

For i=2 **to** k-1 **do** *//compute the lengths of accumulative slack//*

{$a_i=a_{i-1}\times \frac{p_i}{p_{i-1}} - e_i^c$ }

$b_1=0$, $b_2=f_1=e_i^c$

For i=2 **to** k **do** *//compute b_i and f_i of task $T_i\in\tau$//*

{$G_i=\left\lfloor \frac{e_i^c}{a_{i-1}} \right\rfloor \times p_{i-1}$

$L_{Gi}=1-\frac{a_{i-1}}{G_i}\times\left\lceil \frac{e_i^c}{a_{i-1}} \right\rceil$

$f_i=G_i\times L_{Gi}+\frac{e_i^c}{p_i}$, $b_{i+1}=f_i$ }

return $S^*=\{\forall T_i\in\tau \mid (b_i, f_i)\}$

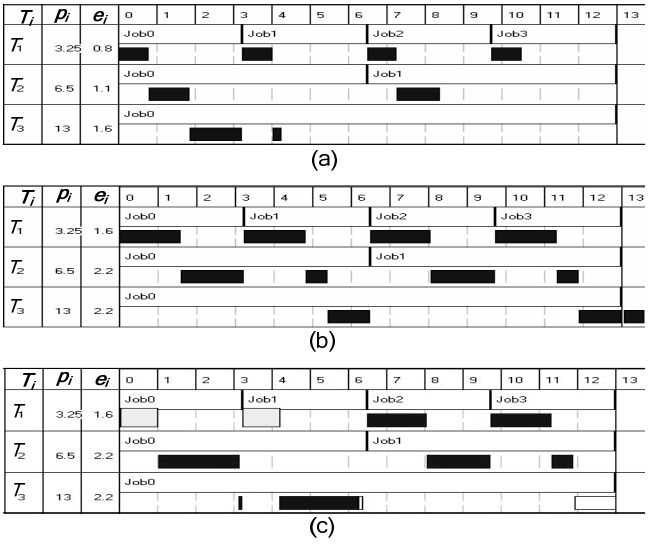

Fig. 4. Examples of *h-schedules*

Example. Consider a 3-task system $\tau=\{T_1, T_2, T_3\}$ with $e_1=0.8$, $e_2=11$, $e_3=1.6$, $p_1=3.25$, $p_2=6.5$ and $p_3=13$. The CPU has three speed levels: $s_1=1$, $s_2=0.8$ and $s_3=0.5$. The *h-schedule* with the highest speed is shown in Figure 4(a) and $U_\tau \fallingdotseq 0.538$. We first search for the critical speed of s_c by using Algorithm 1. Because $U_\tau^2 \fallingdotseq 0.673$ and $U_\tau^3 \fallingdotseq 1.077$, we select s_3 as the critical speed. The schedule with s_c can be illustrated in Figure 4(b). In fact, we can obtain the task features by using Algorithm 2, instead of producing the whole schedule. Therefore, we have $b_1=0$, $f_1=b_2=1.6$, $f_2=b_3=5.4$ and $f_3>12$.

4. SCHEDULING ALGORITHMS

Before presenting the algorithms, we introduce using the following notations throughout this section. For an *h-schedule* under speed s_c, we define:

ε^c: the length of execution time that exceeds p_k. It can be formulated as $(U_\tau^c - 1)\times p_k$.

δ: the shortest execution time with speed s_{c-1} that prevents an *h-schedule* with s_c from deadline missing. It can be denoted as $\dfrac{\varepsilon^c \times s_{c-1}}{s_{c-1}-s_c}$.

For example, in Figure 4(b), we can derive that $\varepsilon^c=(U_\tau^3-1)\times p_3 \cong 1$ and $\delta=\dfrac{\varepsilon^3 \times s_2}{s_2-s_3} \cong 2.67$. That is, the *h-schedule* has to increase the speed of an interval of at least 2.67 in length from s_3 to s_2 for preventing missing a deadline.

4.1. Proposed algorithm for the intra-task schedule

Figure 5 presents an off-line DVS algorithm for the intra-task schedule, which minimizes the

DVS-intra-task (τ)

1. **Input**: task set τ with transformed task periods
2. **Output**: assign speed levels *to h-schedule*
3. Compute the critical speed s_c using Algorithm 1
4. **If** $U_\tau^c \le 1$ **Then** assign s_c to the whole *h-schedule*.
5. **Or Else** { $\varepsilon^c = (U_\tau^c - 1) \times p_n$
6. $\delta = \dfrac{\varepsilon^c \times s_{c-1}}{s_{c-1} - s_c}$
7. Assign speed s_c to the interval $[0, p_k - \delta]$
8. Assign speed s_c to the interval $[p_k - \delta, p_k]$
9. }
10. update b_i and f_i of every task.

End Algorithm DVS-intra-task (τ)

Fig. 5. The pseudo-code of Algorithm 3

number of speed switches and the energy consumption.

Theorem 3. An *h-schedule* for task set τ consumes the minimum energy using the DVS-intra-task(τ) algorithm.

Proof: An *h-schedule* with minimum energy consumption does not contain any idle period. In our scheme, the energy consumption is determined by the total amount of time the processor runs at speed s_c and s_{c-1}, respectively. It is clear that, according to line 7 and line 8 of the DVS-intra-task (τ), an *h-schedule* does not contain any idle period. Moreover, because of the convex of power/speed curve, the wide-gapped speed adjustments will hamper significant energy savings [26]. Consequently, the range of speed adjustments of the DVS-intra-task (τ) is at most one speed level, when an *h-schedule* under speed s_c incurs missing a deadline. Clearly, the speed adjustment for *h-schedule* minimizes energy consumption, and the theorem follows: □

The algorithm in Figure 5 runs in O(d+$k$log$k$) time, where k denotes the number of task in τ. In the algorithm, its input periods have to be transformed in advance by *Sr* [6], which runs in O(k log k) time, and Algorithm 1 (in line 3) needs O(d) time for finding a critical speed. Notably, the time complexity of the algorithm in [15] is O(dnlog n) while n denotes the number of jobs. In the task model, the number of jobs is far greater than that of task. Therefore, our scheme outperforms their algorithms, even if we ignore the overhead of producing the whole schedule in their method.

4.2. Proposed algorithm for the inter-task schedule

In an *h-schedule* produced by a DVS-inter-task(τ), every job has a unique execution speed whenever it performs. However, the decision for speed adjustment is similar to that of the *knapsack* problem with an unbroken object and identical value. More precisely, given that δ is derived from the *h-schedule*, we have to decide which jobs can be placed into the interval in such a way that the sum of their execution time is greater than δ and their differences are minimal. Unfortunately, optimization algorithm for this decision problem runs in pseudo-polynomial time [3]. In other words, the time complexity depends on the length of an output schedule.

The objective of our algorithm is to avoid the fluctuation of execution speeds as much as

DVS-inter-task (τ)

01. **Input**: task set τ with transformed task periods
02. **Output**: the assignment of speed levels to all jobs
03. Obtain the critical speed s_c using Algorithm 1
04. Obtain the f_i and b_i of T_i in τ using Algorithm 2
05. Sort the jobs' f_i in increasing order in $J_L=\{j_1,\ldots,j_k\}$
06. $\varepsilon^c=(U_\tau^c-1)\times p_n$ and $\delta=\dfrac{\varepsilon^c\times s_{c-1}}{s_{c-1}-s_c}$

    $J=\varnothing$, $e_{min}=\infty$
07. **For** i=1 **to** n **do**
08. {choose j_i and obtain its execution length e_i^c
09. **If** $\delta \geqq e_i^c$ **Then** $J_L=J_L-j_i$ and $J=J\cup j_i$
10. Or **ElseIf** $e_i^c<e_{\min}$

    Then $e_{\min}=e_i^c$ and $j_{\min}=j_i$
11. }
12. $J_L=J_L-j_{\min}$, $J=J\cup j_{\min}$
13. Assign the jobs in J_L to speed level s_c
14. Assign the jobs in J to speed level s_c+1
15. Update b_i and f_i of job j_i in J

Fig. 6. The pseudo-code of Algorithm 4

possible. Thus, the proposed method has to know the task features produced by Algorithm 2. In order to choose the successive jobs for increasing their speed, we sort all jobs by their finishing times and it takes O($n$log n) time. From lines 7 to 11, they search for the suitable jobs according to this order and books the nearest job with the minimum length of execution. Hence, the cost is at most O(n). For the remaining part of the complexity, period transformation takes O($k$log$k$), finding the critical speed needs O(d) and to compute the f_i and b_i of every task O(k) is needed. Because $n>>k$ in a periodic task model, the total running time of Algorithm 4 is O(d+$n$log$n$).

Example: In Figure 4(c), the first job of T_1 is placed in job set J and other jobs cannot be picked, because of the for-loop of Algorithm 4 in Figure 6. Finally, in line 12, the second job of T_1 is assigned to j_{min} and included in J.

5. PERFORMANCE ANALYSIS

Besides the proposed methods, we discussed the performance of the techniques in [9, 14, 15, 26]. Because these methods belong to DVS-intra-task (τ) scheduling and are optimal solutions in power consumption, we can only compare their time complexities. In addition, the energy efficiency of our DVS-inter-task (τ) method is presented.

The complexity of given methods are shown in Table 1. The second and third rows present the time complexity of DVS-intra-task(τ) and DVS-inter-task(τ) techniques, respectively. The fourth row shows the space complexity of each method. Notations n, k, and d denote the number of *jobs*, *tasks,* and *speed levels*, respectively. Notably, L denotes the length of an input schedule, and "X" denotes that the corresponding method does not provide such a technique. In a periodic

Table 1. Time and space complexities

	Proposed.	Ishihara [9]	Li [15]	Kwon[14] and Yao[26]
Intra-task	O(d+$k$log$k$)	O(dn)	O(dnlog$n$)	O(n^3)
Inter-task	O(d+$n$log$n$)	X	X	X
Space	O(d+n)	O(d+n)	O(L)	O(L)

task system, the number of jobs is far greater than that of tasks and our DVS-intra-task (τ) algorithm outperforms those of others.

Notably, the method proposed by Ishihlara et al. is formulated as a linear programming problem and has at least O(dn) time complexity [9]. However, the optimality of the technique is confined to a single task. Therefore, the optimality does not hold for the practical case in which every task has different execution speeds. In addition, the techniques proposed in [14, 15] have to generate a "canonical" schedule before voltage/speed adjustments. Their space complexities depend on the length of such schedule and could not be generated in polynomial time. For example, in Figure 1(a), we have to generate a canonical schedule with length of the Least Common Multiplier (LCM) of these five tasks. In Figure 1(b), the length of the canonical schedule is at most 21.2.

Our experiments were carried out in two respects: (1) the inflation of U_τ caused by *Sr* and (2) the energy-efficiency of the DVS-inter-task(τ) scheme. Because of period transformation, U_τ is greater than its original utilization and the difference between them is called *inflation*. In Figure 7, *Sr* is performed in the 20,000 randomly generated task sets with varying sizes. For example, when every τ contains 9 tasks, the average inflation per task set is 0.14. The figure indicates that inflation will rise at a rate that is proportionate to the size of task set.

Since the DVS-intra-task(τ) scheme produces a min-energy schedule, it can be a yardstick by which we can assess the energy efficiency of the DVS-inter-task(τ). In Figure 8, it presents the percentage of normalized deviation produced by the DVS-inter-task(τ) to that of an optimal solution versus the sizes of task set. Because the previous works [9, 14, 15, 26] are optimal in energy consumption, we do not need to compare them again with the proposed DVS-inter-task(τ). The figure shows that the energy consumption of the DVS-inter-task(τ) does not deviate more than 6% from that of optimal solution. It also presents that energy consumption is rather sensitive to the varying sizes of task sets. The deviation is less than 4% when the sizes of task

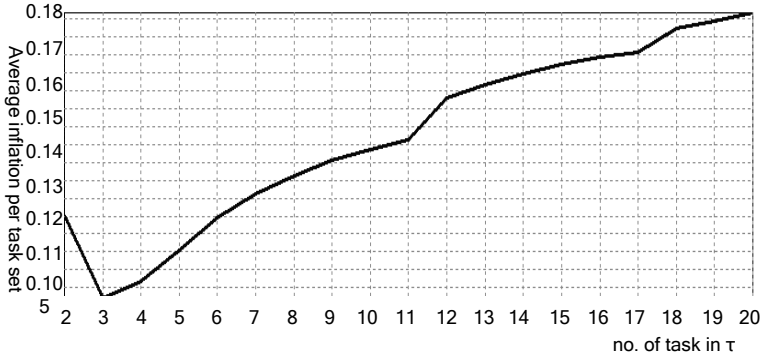

Fig. 7. Average inflation versus the number of tasks

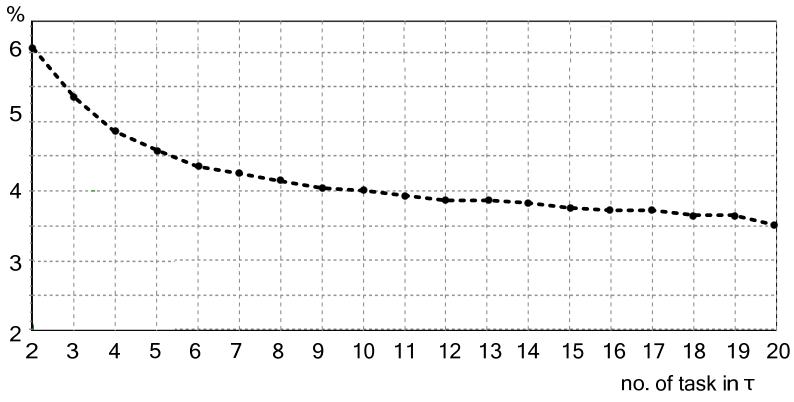

Fig. 8. Normalized deviation produced by the DVS-inter-task

sets are greater than 10. The main reason is that, when the number of tasks increases, the number of candidate tasks for sharing the slack time is also increased.

6. CONCLUSIONS

In this paper, we considered the pinwheel task scheduling on a variable voltage processor with d discrete voltage/speed levels. In the assumption of harmonic task periods, we gave an intra-task scheduling algorithm, which constructs a minimum energy schedule for k periodic tasks in $O(d+k\log k)$ time. We also gave an inter-task scheduling algorithm, which decreases the number of speed/voltage switching in $O(d+n\log n)$ where n denotes the number of jobs. Because of the periodic task model, the number of jobs is far greater than that of tasks. Our schemes outperformed the scheme in [15] even if the task is equivalent to the job. Moreover, since the schedule is obtained without first generating an actual schedule, our schemes are *truly* polynomial time algorithms. We also proposed some fundamental properties associated with jitterless schedules. The useful properties may provide some new insights for jitterless task scheduling.

REFERENCES

[1] H. Aydin, R. Melhem, D. Mosse and P. Mejia-Alvarez. "Power-Aware Scheduling for Periodic Real-Time Tasks." *IEEE Trans. Comput.,* 53(5): 584-600, May 2004.

[2] Sanjoy K. Baruah, Azer Bestavros, *"Pinwheel Scheduling for Fault-Tolerant Broadcast Disks in Real-time Database Systems," Proceedings of the IEEE International Conference on Data Engineering* ICDE 1997: 543-551.

[3] G. Brassard and P. Bratley. *Fundamentals of Algorithmics*. Prentice Hall International, 1996.

[4] D.-R. Chen. *"Scheduling Methods for Periodic Tasks in Real-Time Systems." Doctoral Dissertation National Taiwan University of Science and Technology,* 2005.

[5] F. Gruian. *"Hard Real-Time Scheduling Using Stochastic Data and DVS Processors." In Proceedings of the International Symposium on Low Power Electronics and Design*, 46-51, August 2001.

[6] C.-C. Han, K.-J. Lin and C.-J. Hou. Distance-Constrained Scheduling and Its Applications to Real-Time Systems. *IEEE Trans. Comput.* 45(7): 814-826, July 1996.

[7] C.-W. Hsueh and K.-J. Lin, "Scheduling Real-Time Systems with End-to-End Timing Constraints Using the Distributed Pinwheel Model," *IEEE Trans. Computers*, Vol.50, No.1, pp.51-66, January 2001.

[8] C.-W. Hsueh, K.-J. Lin and C.-J. Hou, "Distance-Constrained Scheduling and Its Applications to Real-Time Systems," *IEEE Trans. Computers*, Vol.45, No.7, pp.814-826, July 1996.

[9] T. Ishihara and H. Yasuura. *"Voltage scheduling problem for dynamically variable voltage processor." In Proceedings of International Symposium on Low Power Electronics and Design,* 197-202, 1998.

[10] M. Kang, D.-I. Kang, J. Suh and J. Lee, "An energy-efficient real-time scheduling scheme on dual-channel networks," *Information Sciences*, Vol.178, Issue 12, 15th June 2008, pp.2553-2563.

[11] W. Kim, J. Kim, and S. L. Min. A. *"A Dynamic Voltage Scaling Algorithm fro Dynamic-priority Hard Real-Time Systems Using Slack Time Analysis." In Proceedings of Design, Automation and Test in Europe (DATE'02)*, 788-794, March 2002.

[12] W. Kim, D. Shin, H.S. Yun, J. Kim, and S.L. Min. *"Performance Comparison of Dynamic Voltage Scaling Algorithms for Hard Real-Time Systems." Proc. Eighth Real-Time Technology and Applications Symp.*, 219-228 2002.

[13] S. Kim and P. K. Varshney, *"An Adaptive Bandwidth Reservation Algorithm for QoS Sensitive Multimedia Cellular Network," Proceedings of the IEEE VTC2002-Fall*, Sept. 2002, Vancouver, Canada, pp.1475-1479.

[14] W.-C. Kwon and T. Kim. "Optimal voltage allocation techniques for dynamically variable voltage processors." *ACM Trans. Embedded Comput. Sys.*, 4(1):211-230, February 2005.

[15] M. Li and F. F. Yao. "An efficient algorithm for computing optimal discrete voltage schedules." *SIAM J. Comput.*, 35(3): 658-671, 2006.

[16] Jane W.S. Liu. *Real-Time Systems*. Prentice Hall, 2000.

[17] S.-A. Li, C.-C. Hsu, C.-C. Wong and C.-J. Yu *"Hardware/software co-design for particle swarm optimization algorithm," Information Sciences*, *In Press, Corrected Proof*, *Available online* 2 August 2010.

[18] L. –L. Lu, J. –L. C. Wu and W. –Y. Chen, *"The study of handoff prediction schemes for resource reservation in mobile multimedia wireless networks," Proceedings of the of AINA 2004*, March 2004. Fukuoka, Japan, Vol.2, pp.379-384.

[19] M. Marsan, S. Marano, C. Mastroianni, and M. Meo, "Performance analysis of cellular mobile communication networks supporting multimedia services," *Mobile Network and Applications*, Vol.5, No.3, pp.167-177, March 2000.

[20] S.-J. Park and K.-H. Cho, "Real-time preemptive scheduling of sporadic tasks based on supervisory control of discrete event systems," *Information Sciences*, Vol.178, Issue 17, 1st September 2008, pp.3393-3401.

[21] P. Pillai and K. G. Shin. *"Real-Time Dynamic Voltage Scaling for Low-Power Embedded Operating Systems." In Proceedings of 18th ACM Symposium on Operating Systems Principles (SOSP'01)*, 89-102, October 2001.

[22] M. Saleem, G. A. Di Caro, M. F. Swarm *"Intelligence based routing protocol for wireless sensor networks: Survey and future directions", Information Sciences*, *In Press, Corrected Proof*, *Available online* 23 July 2010.

[23] P. S. Sausen, M. A. Spohn and A. P. Broadcast, " Routing in wireless sensor networks

with dynamic power management and multi-coverage backbones" *Information Sciences*, *Volume 180, Issue 5*, *1 March 2010*, pp.653-663.

[24] Y. Shin, K. Choi, and T. Sakurai. *"Power Optimization of Real-Time Embedded Systems on Variable Speed Processors." In Proceedings of the International Conference on Computer-Aided Design*, 365-368, November 2000.

[25] D. Shin, J. Kim, and S. Lee. "Intra-Task Voltage Scheduling for Low-Energy Hard Real-Time Applications." *IEEE Design and Test of Computers,* 18(2): 20-30, March 2001.

[26] F. Yao, A. Demers, and S. Shenker. "*A Scheduling Model for Reduced CPU energy." In Proceedings of the 36th Annual IEEE Symposium on Foundations of Computer Science*, 374-382, 1995.

[27] D. Zhu, D. Mosse and R. Melhem. "Power-Aware scheduling for and/or graphs in Real-time systems." *IEEE Trans. Parallel and Distributed Sys.*, 15(9): 849-864, September 2004.

[28] Intel Corporation, Wireless Intel SpeedStep Power Manager – Optimizing power consumption for the intel PXA27x processor family, *Wireless Intel SpeedStep® Power Manager White pape*r, 2004. http://download.intel.com/ pressroom/kits/pxa27x/wp_wireless_ speedstep.pdf.

[29] Advanced Micro Devices Corporation, *AMD Athlon 64 Processor Power and Thermal Datasheet*, 2006. http://www.amd.com/us-en/assets/content_type/white_papers_and_tech_ docs/30430.pdf.

Da-Ren Chen

received his B.E., M.E., and Ph.D. degrees from the National Taiwan University of Science and Technology, in 1999, 2001, and 2006, respectively. Presently, he is an Associate Professor in the Department of Information Management at the National Taichung University of Science and Technology. His research interests include real-time systems, fault-tolerance systems, parallels and distributed computing, and scheduling algorithms.

You-Shyang Chen

is now an Assistant Professor in the Department of Information Management at the Hwa Hsia Institute of Technology and majors in Information Management. He was a practitioner in the financial industry and manufacturing industry for over 20 years. He received his Bachelor's degree in Industry Management from the National Taiwan University of Science and Technology in 1988, and his Master's degree and Ph.D. degree in Information Management from the National Yunlin University of Science & Technology in 2006 and 2009, respectively. His major research interests are focused on in financial analysis, soft computing, and rough set theory. He has published more than 20 papers.