Journal of Applied Research and Technology
ISSN: 1665-6423
jart@aleph.cinstrum.unam.mx
Centro de Ciencias Aplicadas y Desarrollo Tecnológico
México

Da-Ren, Chen; Young-Long, Chen; You-Shyang, Chen
Time and Energy Efficient DVS Scheduling for Real-Time Pinwheel Tasks
Journal of Applied Research and Technology, vol. 12, núm. 6, diciembre, 2014, pp. 1025-1039
Centro de Ciencias Aplicadas y Desarrollo Tecnológico
Distrito Federal, México

Time and Energy Efficient DVS Scheduling for Real-Time Pinwheel Tasks

Chen Da-Ren*[1], Chen Young-Long[2] and Chen You-Shyang[3]

[1] Department of Information Management
National Taichung University of Science and Technology
Taichung city, Taiwan, R.O.C.
*danny@nutc.edu.tw
[2] Department of Computer Science and Information Engineering
National Taichung University of Science and Technology
Taichung city, Taiwan, R.O.C.
[3] Department of Information Management
Hwa Hsia University of Technology,
New Taipei city, Taiwan, R.O.C.

ABSTRACT
Dynamic voltage/frequency scaling (DVFS) is one of the most effective techniques for reducing energy use. In this paper, we focus on the pinwheel task model to develop a variable voltage processor with d discrete voltage/speed levels. Depending on the granularity of execution unit to which voltage scaling is applied, DVFS scheduling can be defined in two categories: (i) inter-task DVFS and (ii) intra-task DVFS. In the periodic pinwheel task model, we modified the definitions of both intra- and inter-task and design their DVFS scheduling to reduce the power consumption of DVFS processors. Many previous approaches have solved DVFS problems by generating a canonical schedule in advance and thus require pseudo polynomial time and space because the length of a canonical schedule depends on the hyperperiod of the task periods and is generally of exponential length. To limit the length of the canonical schedules and predict their task execution, tasks with arbitrary periods are first transformed into harmonic periods and their key features are profiled. The proposed methods have polynomial time and space complexities, and experimental results show that, under identical assumptions, the proposed methods achieve more energy savings than the previous methods.

Keywords: Hard real-time systems, Power-aware scheduling, Dynamic voltage scaling, Pinwheel tasks.

1. Introduction

In the last decade, energy-aware computing has been widely applied not only for portable electronic devices, but also for large systems which incur large costs for energy and cooling. With dynamic voltage/frequency scaling (DVFS) techniques [3, 7, 8, 9, 10, 11, 12, 15, 16, 17, 18, 19], processors can perform at a range of voltages and frequencies. In the CMOS processors, the energy consumption is at least a quadratic function of its supply voltage (and hence the processor frequency) [3, 7, 8, 19, 20, 21], and total energy consumption could be minimized by sharing slack time while satisfying the time constraints of the tasks. For DVFS hard real-time systems, we defined two categories of DVFS scheduling: inter-task DVFS and intra-task DVFS. In the former, 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 does not change until it is preempted or completed. This definition is somewhat different from the inter-task definition in [3, 13, 16, 20, 23] wherein the speed of each tasks is fixed and cannot be changed between different instances (jobs). Inter-task DVFS 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 DVFS algorithms adjust the CPU speed within the boundaries of a task. Intra-task DVFS techniques are kept under software and compiler control using program checkpoints or voltage scaling points of the target real-time software. 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 the compiling time and indicate places in the code where the processor speed and voltage should be re-calculated. These checkpoints could increase programming complexity and the overhead of real-time systems. The previous definition of intra-task scheduling [24, 18, 25] is similar to our definition of inter-task DVFS.

Many studies have addressed the problem of task scheduling with minimum energy DVFS. Yao et al. [21] proposed a theoretical DVFS model and an $O(n^3)$ algorithm for computing a min-energy DVFS schedule in a continuous variable voltage CPU. Ishihara et al. [14] 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. [17] proposed an optimal discrete approach based on the continuous version in [21] and therefore requires $O(n^3)$ time. Li et al. [18] proposed an $O(dn \log n)$ time algorithm which constructs a minimum energy schedule without first computing the optimal continuous schedule. The abovementioned min-energy DVFS scheduling algorithms have to generate certain schedules in advance as intermediate processing steps. For example, the Bipartition algorithm in [18] has to generate an s-schedule and a reversed s-schedule in advance. Moreover, the Alloc-vt algorithm in [17] has first to generate a min-energy continuous schedule from [21]. Since the lengths of such schedules depend on the LCM of task periods, their algorithms could not be completed in polynomial time. In addition, in the periodic tasks systems, the preprocessing overhead produced by these approaches may become unsustainable when tasks frequently join and leave the system. Many theoretical models for DVFS only consider the power consumption function with convexity [3, 20, 21, 22]. In these models, the processor must be able to run at infinite real-number speed levels to achieve optimality, while an off-the-shelf processor with variable voltages runs only at a finite number of speed levels. For example, Intel's SpeedStep® technology [25] and AMD's Cool'n Quiet® [27] are currently used in general-purpose mobile devices and respectively support 3 and 5 speed levels. Therefore, an applicable model for DVFS scheduling should capture the discrete, rather than continuous, nature of the available speed scale.

In network systems, jitter or packet delay variation (PDV) is defined as the variation in the time between successive packet arrivals caused by network congestion, time drift or route changes [19, 28]. PDV is an important quality of service factor to evaluate network performance. One of the most widely-used techniques to improve PDV is pinwheel scheduling [1, 5, 29, 30, 31, 32, 33]. A pinwheel task τ_i is characterized by two positive integer parameters, an execution requirement and a window-length with the understanding that the shared resource needs to be allocated to task i for at least an out of every b consecutive time units. Pinwheel task systems were developed to meet the performance requirements of satellite-based communications. In more recent 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 a B3G system guarantee the quality-of-service (QoS) of real-time (RT) traffic. To guarantee the QoS of RT traffic in a cell, many researchers [1, 2, 5, 4, 29, 32, 34] have proposed using pinwheel scheduling algorithms to reduce the jitter of variable bit rate (VBR) traffic in a cell. In addition, pinwheel scheduling has also been applied in channel assignment policies with buffer and preemptive priority for RT traffic. In other applications, such as the medium access control (MAC) layer of CDMA and TDMA-based wireless networks [19, 35, 36], many pinwheel scheduling schemes have been proposed to solve frame-based packet scheduling problems. These pinwheel methods provide low delay and low jitter for RT traffic and a short-queue length for non-RT traffic.

This paper discusses theoretical power-aware real-time scheduling. We consider a discrete DVFS scheduling problem for periodic task systems given worst-case execution times (WCET). We propose an algorithm that finds a min-energy intra-task DVFS schedule in $O(d+k \log k)$ time. An inter-task DVFS method is also proposed, with time and space complexities of $O(d+n \log n)$ and $O(d+n)$, respectively. Notations k, n and d respectively denote the number of tasks, jobs and voltage levels. In section 2, we present the model and the notational conventions. Section 3 introduces DCTS and the proposed task profiling algorithms. The

DVFS scheduling algorithms are proposed in Section 4. In Section 5, we present the performance analyses of the proposed algorithms and compare the consumed utilization of transformed task sets with those of their original task sets. Section 6 concludes this paper.

2. Task model

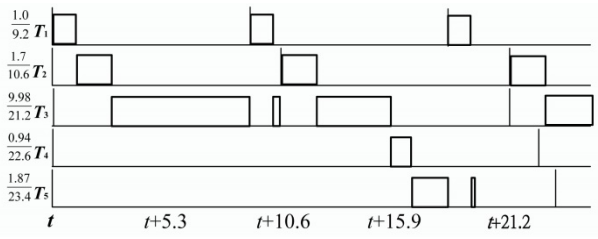
A pinwheel task T_i is defined by two positive integers, an execution requirement and a window length, with the understanding that the task T_i needs to be allocated to the shared resource for at least a out of every b consecutive time units. In the distance-constrained task systems (DCTS) [13], they stipulated that temporal distance between any two consecutive executions of each job in the pinwheel schedules should always be less than a certain value. DCTS modifies the distance-constraints of longer task periods as a multiplier of a power of two (abbr. to *harmonic* number) shorter periods, neither of which is longer than its original distance-constraints using the *Sr* algorithm [13]. The advantage of the period transformation is that the produced schedules have regular start, preemption and finish times, and therefore provide good predictability.

The target model focuses 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 a 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 pi and deadline (as well as their distance constraints) has been transformed as harmonic in 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 bi and finishing time f_i of T_i are fixed and can be efficiently obtained in Section 3.

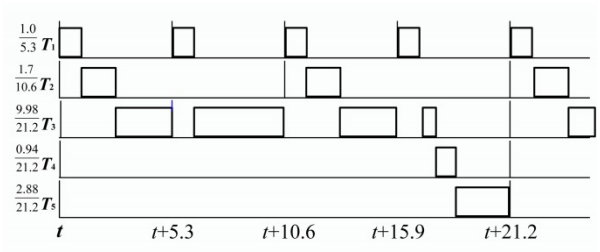
The clock speeds corresponding to d given discrete voltage/speed levels are denoted by $s_1>s_2>\ldots>s_d$. The highest speed s_1 is always fast enough to guarantee a feasible schedule for the given tasks. Moreover, e_i and τ_i^g respectively denote the duration of the execution at speeds s_1 and s_g. 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 the highest speed. The power P, or energy consumed per unit of 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 DVFS scheduling (abbreviated to IntraDVFS). The first goal is to find, for any given task set τ, a feasible schedule produced by IntraDVFS 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 InterDVFS) schedules and to reduce their energy consumption level as close as possible to that of the schedule produced by IntraDVFS.

3. Distance-constrained task systems

This section introduces the concept of an *h*-schedule produced by algorithm *Sr* mentioned by DCTS [13] and discusses its important properties. Algorithm *Sr* [31] converts the periods into a set of specialized periods that are not greater than the original periods, while minimizing the increased weight of the total task set . For example, in Fig. 1(a), a system consists of five tasks with periods 9.2, 10.6, 21.2, 22.6 and 23.4, for which the corresponding execution times are 1.0, 1.1, 9.98, 0.94 and 1.87. After applying ***Sr***, the new task periods {5.3, 10.6, 21.2, 21.2, 21.2} are illustrated in Fig. 1(b). Because the lengths of the task periods are *multiples* of the power of 2, the schedule for each task has no *jitter*, and their relative starting and ending times are fixed.



(a)



(b)

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

3.1 Base speed selection

In this section, 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 provide a speed factor for a WCET schedule to minimize energy consumption and the speed adjustment times.

Definition 1. An *h-schedule,* for which τ conforms to the RM policy and the lengths of the task period, is transformed into harmonics using ***Sr*** [31].

Without loss of generality, the length of an *h-schedule* is equal to the longest task period 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. Let $U_\tau \geq 1$; there is no slack time in an *h-schedule*.

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

$$\sum_{\forall i \in \tau} m_i \times e_i < p_k$$
$$\rightarrow \sum_{\forall i \in \tau} \frac{e_i}{p_k} \times \frac{p_k}{p_i} < 1 \quad (1)$$
$$\rightarrow U_\tau < 1$$

This contradicts our assumption.

Definition 2. In the *h-schedule* for τ, we define a deadline as being *tight* if task T_i finishes 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 the deadline is not tight.

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 times. Without loss of generality, assume T_k has a tight deadline at time t. For all T_x, $x<k$, thus it has higher priority than T_k. Because task periods are *harmonic*, we obtain

$$\frac{p_k}{p_x} = m_x \quad (2)$$

m_x denotes the power of 2.

Moreover, at the time of t-p_k, exactly k tasks are released, and 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:

$$\sum_{x=1}^{k} m_x \times e_x = p_k \times \sum_{x=1}^{k} \frac{e_x}{p_x} \quad (3)$$

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

$$f_k \geq p_k \Rightarrow (p_k \times \sum_{x=1}^{k-1} \frac{e_x}{p_x}) + e_k \geq p_k$$
$$\Rightarrow e_k \geq p_k - p_k \times \sum_{x=1}^{k-1} \frac{e_x}{p_x}$$
$$\Rightarrow e_k \geq p_k (1 - \sum_{x=1}^{k-1} \frac{e_x}{p_x}) \quad (4)$$
$$\Rightarrow \frac{e_k}{p_k} \geq 1 - \sum_{x=1}^{k-1} \frac{e_x}{p_x}$$

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

The proof of the "if" direction is easily found. 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:

$$\sum_{x=1}^{k} \frac{p_k}{p_x} \times e_x > p_k \Rightarrow U_\tau \geq 1 \quad (5)$$

When a $U_\tau > 1$, *h-schedule* is not feasible, at least one job is missing its deadline. When $U_\tau = 1$, the latest job in interval *I* must have a tight deadline.

This completes the proof.

Based 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 define the suitable processor speed for *h-schedule* as follws:

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

The function of speed s_c produced by *CriticalSpeed*(τ) in Fig. 2 generates the base speed for the tasks to produce a suitable *h-schedule* and to reduce the number of speed adjustments.

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.

Figure 2. Pseudo-code of Algorithm 1.

3.2 Task execution profiling

After computing s_c, the beginning and finishing times of each task under speed s_c can be derived without constructing an actual *h-schedule*. In Fig. 3, Algorithm TaskProfiling obtains b_i and f_i of all the tasks that are in $O(k)$ time.

TaskProfiling(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*
    $a_i=a_{i-1} \times \frac{p_i}{p_{i-1}} - e_i^c$; *accumulative slack//*
  $b_1=0$, $b_2=f_1=e_i^c$;
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 $S^* = \{\forall T_i \in \tau \mid (b_i, f_i)\}$;

Figure 3. Pseudo-code of Algorithm 2.

4. Scheduling algorithms

For an *h-schedule* under speed s_c, we define the following:

ε^c: the length of execution time that exceeds p_k and is derived from $(U_\tau^c - 1) \times p_k$.

δ: the shortest execution time under execution speed s_{c-1} that prevents an *h-schedule* under speed s_c from missing deadlines and denotes $\frac{\varepsilon^c \times s_{c-1}}{s_{c-1} - s_c}$.

For example, in Fig. 4(a), we can derive $\varepsilon^c = (U_\tau^3 - 1) \times p_3 \cong 1$ and $\delta = \frac{\varepsilon^3 \times s_2}{s_2 - s_3} \cong 2.67$. That is, the *h-schedule* in Fig. 4(b) has to increase the speed of an interval of at least 2.67 in length from s_3 to s_2 to ensure deadlines are met.

4.1 Proposed algorithm for the intra-task schedule

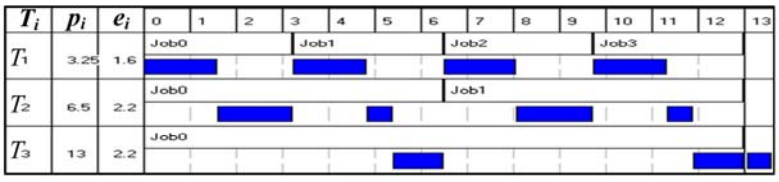
Figure 5 presents an DVFS algorithm for the intra-task schedule, which minimizes the number of speed/voltage transitions and energy consumption.

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

Proof. An *h-schedule* with minimum energy consumption has no idle period. In the algorithm IntraDVFS(*τ*), energy consumption under speed s_c and s_{c-1} is determined by the total processor run time. According to lines 8 and 9 in IntraDVFS (*τ*), an *h-schedule* with WCET clearly contains no idle period. Moreover, because of the convex property of the function related to power speed, the wide-gapped speed adjustments will make it difficult to obtain significant energy savings [43]. Consequently, the range of speed adjustments of the IntraDVFS (*τ*) is at most one speed level, when an *h-schedule* under speed s_c misses a deadline. Therefore, the speed adjustment for *h-schedule* minimizes energy consumption, and this completes the proof.

The algorithm in Fig. 5 has a time complexity of $O(d+k \log k)$ time, where k denotes the number of tasks in τ. In the algorithm, its input periods have to be transformed in advance by *Sr* [19], which runs in $O(k \log k)$ time, and line 3 in algorithm *CriticalSpeed*() needs $O(d)$ time to find a critical speed. Notably, the time complexity of the algorithm in [30] is $O(dn \log n)$ where n denotes the number of jobs. In a task system, the number of jobs is far greater than the number of tasks. Therefore, our scheme outperforms their algorithms, even if we ignore the overhead incurred by producing the whole schedule in the method presented in [30].

(a)



(b)

(c)

Figure 4. Examples of h-schedules.

IntraDVFS (τ)

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

End Algorithm IntraDVFS (τ)

Figure 5. The pseudo-code of Algorithm 3.

4.2. Proposed algorithm for the inter-task schedule

In an *h-schedule* produced by a InterDVFS(τ), each 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, the optimization algorithm for this decision problem runs in pseudo-polynomial time [14]. In other words, the time complexity depends on the length of an output schedule.

The objective of our algorithm is to minimize the fluctuation of execution speeds. Thus, the proposed method has to know the task profiles produced by Algorithm TaskProfiling( ) in Fig. 3. Jobs are arranged in order of increased speed by sorting all jobs by their finishing times and therefore interDVFS takes $O(n \log n)$ time. In Fig. 6, from lines 7 to 10, we they search for suitable jobs according to this order and book the nearest job with the minimum execution time, thus the time complexity is at most $O(n)$. For the time complexity of the rest of algorithm InterDVFS( ), the remaining period transformation takes $O(k \log k)$ time, finding the critical speed takes $O(d)$, and computing the values of f_i and b_i of each task takes $O(k)$. In general, the value of n is much larger than that of k in a periodic task model, and the running time of algorithm InterDVFS( ) is $O(d+n\log n)$.

InterDVFS (τ)

01. **Input**: task set τ with transformed task periods
02. **Output**: the assignment of speed levels to all jobs
    Whenever a task completes early
03. Compute the critical speed s_c using *CriticalSpeed*();
04. Compute the f_i and b_i of T_i in τ using *TaskProfiling*(τ);
05. Sort the jobs according to 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. **Elself** $e_i^c < e_{\min}$ **Then** $e_{\min} = e_i^c$ and $j_{\min} = j_i$;
11. $J_L = J_L - j_{\min}$, $J = J \cup j_{\min}$;
12. Assign the jobs in J_L to speed level s_c ;
13. Assign the jobs in J to speed level s_c+1 ;
14. Update b_i and f_i of job j_i in J ;

End Algorithm InterDVFS (τ)

Figure 6. The pseudo-code of Algorithm 4.

Example. In Fig. 4(c), the first job of T_1 is included in job set J due to the for-loop in algorithm InterDVFS( ). In line 12, the second job of T_1 is finally assigned to j_{min} and included in J.

5. Performance analysis

We discuss the performance of the techniques proposed in [14, 17, 18, 21]. Because the methods belong to intraDVFS scheduling and are optimal power consumption solutions, we can only compare their time complexities.

The levels of complexity for the given methods are shown in Table 1. The second and third rows present the time complexities of InterDVFS and IntraDVFS techniques. Notations *n*, *k* and *d* respectively denote the numbers for *job*, *task* and *speed levels*. Notably, *L* denotes the length of an input schedule. In a periodic task system, the number of jobs is far greater than that of tasks and our intraDVFS algorithm outperforms the others.

	Proposed.	Ishihara[9]	Li [18]	Kwon [12]
Inter-tasks	$O(d+n\log n)$	$O(dn)$	$O(dn\log n)$	$O(n^3)$
Intra-tasks	$O(d+k\log k)$	N/A	N/A	N/A
space	$O(d+n)$	$O(d+n)$	$O(L)$	$O(L)$

Table 1. Time and space complexities.

Notably, the method proposed by Ishihlara et al. is formulated as a linear programming problem and has at least $O(dn)$ time complexity [12]. 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 [16, 18] have to generate a "canonical" schedule before voltage/speed adjustments. The memory space required by the above-mentioned methods is dominated by the length of such schedule and cannot be generated in polynomial time.

The simulations results in Fig. 7 presentes the inflation of U_τ caused by *Sr* and (2) the energy-efficiency of interDVFS scheme. Because of period transformation, U_τ is greater than its original utilization and the difference between them is called *inflation*. In Fig. 7, *Sr* is performed in the 20,000 randomly generated task sets with varying sizes. For example, when every τ contains 6 tasks, the average inflation per task set is 0.14. The figure indicates that inflation will rise at a rate proportionate to the size of task set.

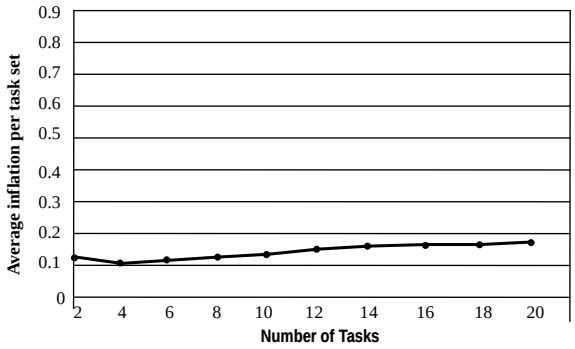


Figure 7. Average inflation versus the number of tasks.

Since the intraDVFS scheme produces a min-energy schedule, it can serve as a yardstick by which to assess the energy efficiency of interDVFS. Figure 8 presents the percentage of the normalized deviation produced by interDVFS as compared to that of the optimal solution versus task set size. Because previous works [9, 12, 14, 15] optimized energy consumption, no comparison need be made with the proposed interDVFS. The figure shows the energy consumption of interDVFS does not deviate more than six percent from that of the optimal solution. It also indicates that energy consumption is rather sensitive to variation of task set size. The deviation is less than 4% when task set size is greater than 10. The main reason is that, when the number of tasks increases, the number of candidate tasks to share the slack time is also increased.

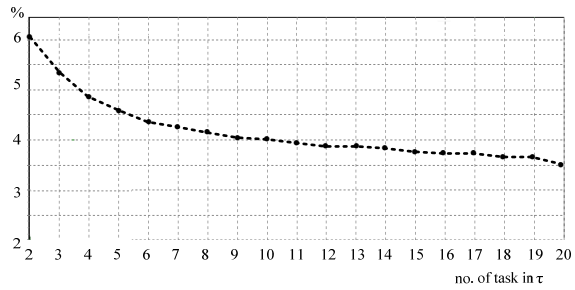


Figure 8. Normalized deviation produced by task.

In additioin, we evaluate the effectiveness of InterDVFS on randomly generated task sets and compare its energy consumption with DRA, AGR [1] and lpSHE [13]. Both DRA and AGR are modified to account for time transition overhead. Each task is characterized by its worst-case execution wc_i, its period p_i, and its deadline d_i, where d_i=p_i. We vary three parameters in our simulations: (1) number of tasks *totaltasks* in τ from 2 to 20 in two task increments, (2) the probability of an early completion (prob. of EC) for each job, and (3) the *bc*/*wc* ratio of BCET to WCET, from 0.1 to 0.9. For any given pair of *totaltasks*, *U* and *bc*/*wc* in *T*, we randomly generate 1000 task sets, and the experiment result is the average value over these 1000 task sets. In a task set, each task period p_i is assigned randomly in the real number range [1, 100]ms with a uniform probability distribution function. The execution time wc_i of each task is assigned in the real number rang [1, min{p_i-1, 450}]. After giving the values of tasks' periods and executions, we assign the utilization *U* of a task set and rescale the p_i of each task such that the summation of the task weights (i.e. wc_i/p_i) is equal to a given *U*. The early completion time of each job in simulations (1) and (2) was randomly drawn from a Gaussian distribution in the range of [BCET, WCET], where BC/WC=0.1. In simulation (3), each experiment was performed by varying BCET from 10% to 90% of WCET. The experiment calculates the energy consumption of each task set in an interval of time *I*=2000. After generating the task set, a duplicate of the task sets are transformed to have powers of 2 periods. Therefore, the duplicate has higher actual utilization than initially intended. In the period transformation tasks, since the simulation tasks still complete early proportionately according to the experiment settings, the slack originated from additional utilization still be utilized by slack-time analysis algorithms.

The processor model we assumed is base 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 [24], and an idle processor consumes at most 500μW at the processor sleep mode. The energy consumptions of all the experiment results are normalized against the same processor running at maximum speed without DVFS technique (non-DVFS). Table 2 is a summary of our simulated ARM8 processor core [37].

Frequency	Voltage	Power
Sleep (idle)	0.5V	<500μW
10MHz	0.7V	4.5mW
20MHz	0.75V	11.2mW
30MHz	0.85V	21.9mW
40MHz	0.96V	36.8mW
50MHz	1.08V	57.5mW
60MHz	1.2V	85.8mW
70MHz	1.33V	123.2mW
80MHz	1.48V	174.4mW
90MHz	1.65V	244.8mW
100MHz	1.82V	330mW

Table 2. Power specification [33, 34].

The overheads considered in the simulations are as follows.

(1) Algorithm execution time and energy

The energy overhead is obtained under the assumption of 80% of the maximum power [24].

(2) Voltage transition time and energy

The assumption of voltage scaling overhead is the same as that in [24] and the transition time is at most 70us between maximum transitions [38]. The energy consumed during each transition is:

$$\Delta E = n(1-\lambda)\cdot C\cdot\left|V_{dd2}-V_{dd1}\right| \quad (6)$$

where λ denotes the efficiency of DC-DC converter.

For the voltage scaling from V_{dd1} to V_{dd2}, the transition time is:

$$\Delta t \approx \frac{2\cdot C}{I_{\max}}\cdot\left|V_{dd2}-V_{dd1}\right| \quad (7)$$

where *C* and I_{max} denote the charge to the capacitor and the maximum output current of the converter.

(3) Context-switch time and energy

The context-switch is assumed to be 50μs at the highest speed S_{max} as presented by David in [40].

Figures 9 (b), 10 (b) and 11 (b) have been done under the assumption that the lengths of task periods in the non-DVFS, DAR, AGR and lpSHE are transformed to the power of 2, while those in Figures 9 (a), 10(a) and 11 (a) are not transformed. As shown in Figure 9 (a), InterDVFS reduces the energy consumption up to 5% over DRA. The utilization of a given task set is 60%. As the number of tasks from 8 to 20, the energy consumption of InterDVFS is increased steadily. The reason for this fact is that InterDVFS focuses on distributing the slack on as many tasks as possible. When the number of tasks increases, the available slack can be shared by many tasks due to the jitterless schedule. Therefore InterDVFS decreases the number of speed/voltage scaling and benefits the energy saving. In Figure 9 (b), DRA, AGR and lpSHE have closer energy consumption with each other than those in Figure 9 (a). InterDVFS still outperforms other methods especially in the large *totaltask*.

In Figure 10(a), when prob. of EC is smaller than 0.8, InterDVFS performs 2%~12% more energy saving compared to AGR and DRA. When prob. of EC is 0.9, the results are not good as we had hoped, they are 7% worse than those of AGR. In the experiment, the *totaltasks* of each task set is randomly determined from 2 to 20. As the probability of early completion decreases, the differences between InterDVFS and DRA or AGR increase substantially. On the contrary, when the largely 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 harmful effects of frequently voltage change compromise the advantage of InterDVFS that benefits the evenly distribution of slack. In Figure 10(b), InterDVFS still outperforms up to 8% energy saving compared to other methods.

The effect of *bc*/*wc* shown in Figure 11(a) and 11(b) confirms our prediction that the energy consumption (*U*=0.8, prob. of *EC* is 0.5) would be highly dependent the variability of the actual workload. When *bc*/*wc*=0.9, the energy consumptions are quite close for all four techniques, as expected. However, once the actual workload decreases, the algorithms are able to reclaim slack time and to save more energy. Algorithm InterDVFS gives the best energy saving, followed by AGR DRA and lpSHE. Decreasing the ratio helps further improve the relative performance of InterDVFS due to the equally assigned voltage/speed level to the future jobs.

In a jitterless schedule, more jobs have the same release times and deadlines as those in the schedule with original periods. Therefore, the appearances of next task arrival (NTA) become regular and the distances between NTAs are longer than that in the schedule without period transformation. At each scheduling point, when the distances between each pair of NTAs are longer, DRA, AGR and lpSHE obtain more energy savings. This is because they can derive longer slack between the NTAs. In addition, when more tasks release at the same time, they can predict the length of slack more precisely and easier. As the example shown in Figure 1(b), a schedule without jitters makes more jobs share available slack than those generated by original periods. Although InterDVFS has the penalty for utilization inflations, it still outperforms other techniques.

Since non-DVFS does not as sensitive as DRA, AGR and lpSHE to period transformation, its energy savings is modest while other methods have relatively less energy consumption. It is likely to that these methods gain more savings by predicting the length of slack in the future interval. Moreover, in the Figures 9(b), 10(b) and 11(b), the energy consumptions of DRA, AGR and lpSHE are closer with each other than those without period transformation in Figures 9(a), 10(a) and 11(a). The experiments show that jitter-controlled schedule can decrease the energy consumptions of up-to-date DVFS algorithms. Therefore, it is a promising technique in many real-time applications.

Figure 12 shows how the number of speed transition changes with the DVFS algorithms. The number of transitions is measured with varying the rates of *bc/wc*. The length of generated schedules and task periods is up to 2000ms and 100ms, respectively. Figure 12 also considers the results

for a clairvoyant algorithm, named bound, which checks all possible scheduling points in the whole schedule as well as looks for the best speeds, start and completion times. In fact, bound is extremely time-consuming for finding a combination with the minimum transition times. The results illustrate that lpSHE, DRA and AGR have higher frequencies of transitions than that of InterDVFS

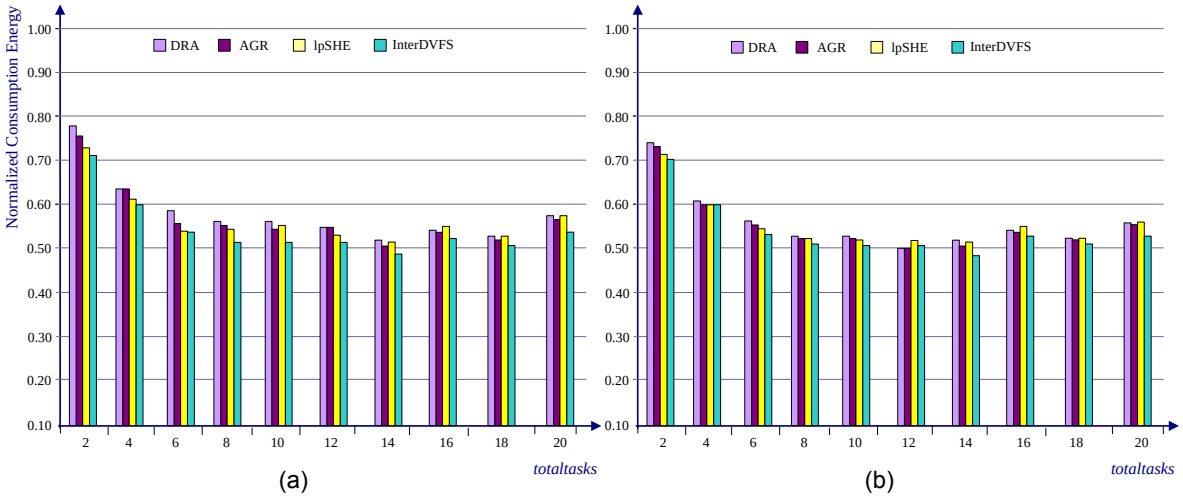


Figure 9. Energy consumption under different totaltasks (a) tasks without period transformation and (b) tasks with period transformation (U=60%, bc/wc=0.5).

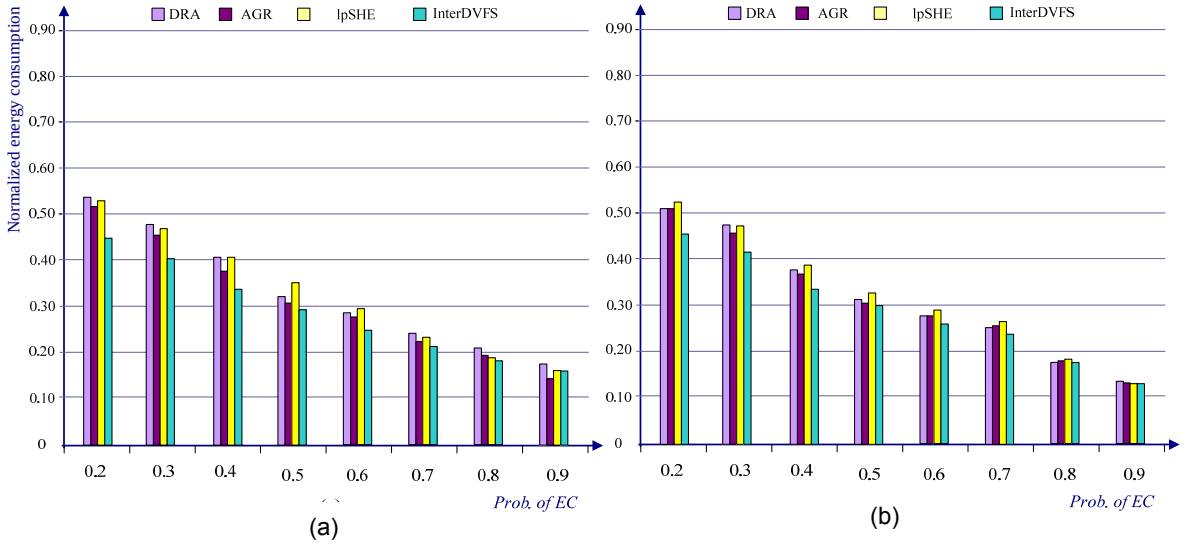


Figure 10. Effect of variations in the prob. of EC (a) tasks without period transformation and (b) tasks with period transformation (20 tasks, U=60%, bc/wc=0.5).

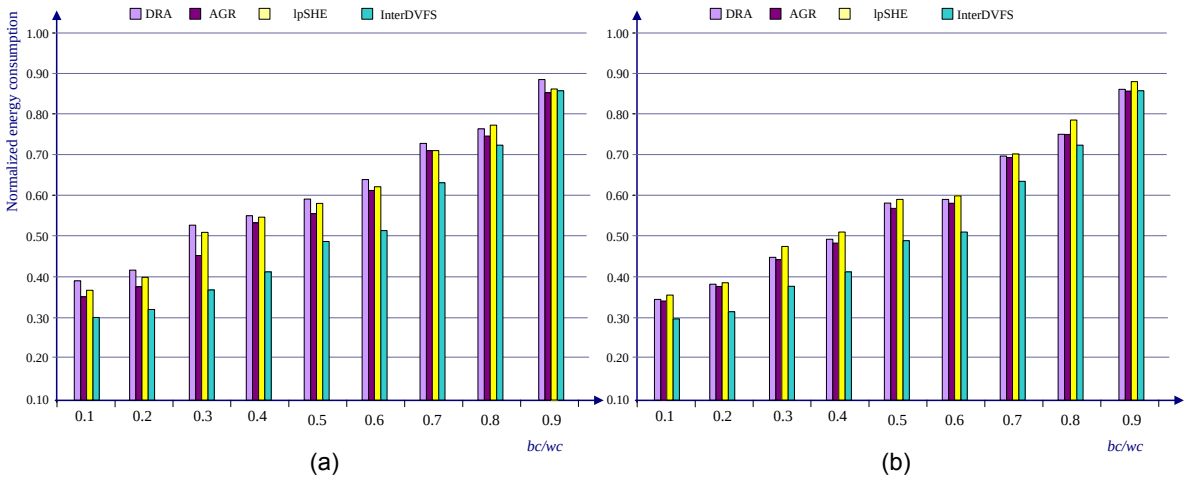


Figure 11. Energy consumption under different bc/wc ratio sets (a) tasks without period transformation and (b) tasks with period transformation (2~20 tasks, U=60%, prob. of EC is 0.5).

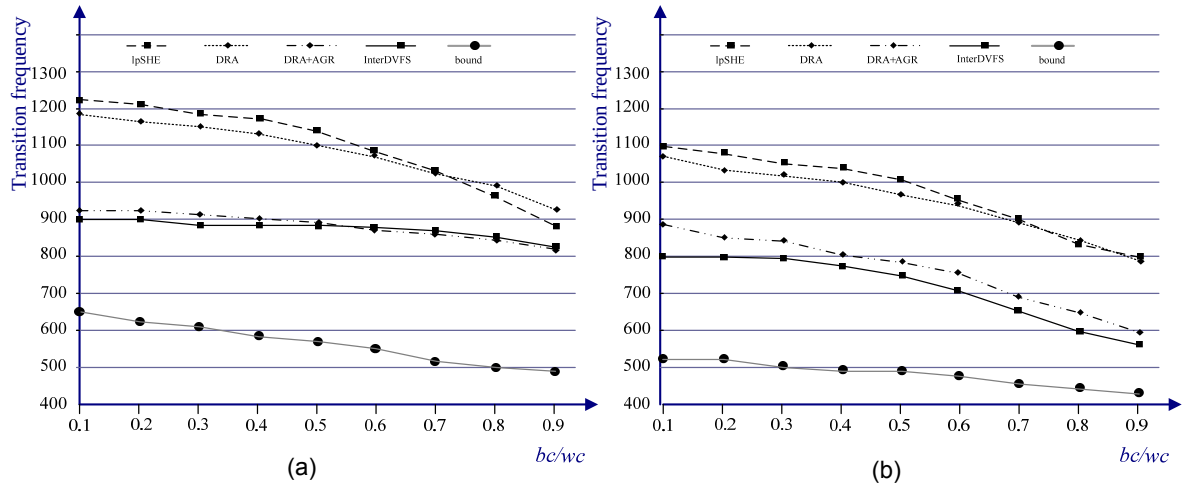


Figure 12. (a) Average Transitions for the task sets versus bc/wc at U=0.5 and d=10 and (b) Average Transitions for the task sets versus bc/wc at U=0.5 and d=4.

6. Conclusion

In this paper, we consider the pinwheel task model on a variable voltage processor with d discrete voltage/speed levels. On the assumption of harmonic task periods, we propose an intra-task scheduling algorithm which constructs a minimum energy schedule for k periodic tasks in $O(d+k \log k)$ time. We also propose an inter-task scheduling algorithm which decreases the number of speed or voltage switching events in $O(d+n \log n)$ where n denotes the number of jobs. Our schemes outperform the scheme in [18] even though the number of tasks is equivalent to that of given number of jobs. Moreover, since the schedule is obtained without generating an actual schedule in advance, our schemes are true polynomial time algorithms. We also propose some fundamental properties associated with jitterless schedules which may provide new insights for jitterless tasks scheduling.

Aknowledgments

The author would like to thank the National Science Council of the Republic of China, Taiwan, for financially supporting this research under Contract No. NSC 102-2221-E-025-003.

References

[1] A. Menéndez-Leonel de Cervantes and H. Benítez-Pérez. "Scheduling strategy for real-time distributed systems," Journal of Applied Research and Technology, vol.8, no.2, pp.177-184, Aug. 2010.

[2] Medina-Santiago, J. L. Camas Anzueto, M. Pérez-Patricio and E. Valdez-Alemán, "Programming Real-Time Motion Control Robot Prototype," Journal of Applied Research and Technology, vol.11, no. 6, pp.297-931, Dec. 2013.

[3] H. Aydin, R. Melhem, D. Mosse and P. Mejia-Alvarez, "Power-Aware Scheduling for Periodic Real-Time Tasks," IEEE Trans. Comput., vol. 53, no.5, pp.584-600, May, 2004.

[4] S. K. Baruah and Azer Bestavros, "Pinwheel Scheduling for Fault-Tolerant Broadcast Disks in Real-time Database Systems," in Proceedings of the IEEE International Conference on Data Engineering, Burlington, VT, USA, ICDE, 1997, pp.543-551.

[5] B. Ernesto and R. Monroy "Real-Time Verification of Integrity Policies for Distributed Systems." Journal of Applied Research and Technology, vol.11, no.6, pp.831-843, Dec. 2013.

[6] K. Varghese Cibu, K. Shankar, "Identification of Structural Parameters Using Combined Power Flow and Acceleration Approach in a Substructure," International Journal of Engineering and Technology Innovation, vol. 1, no. 1, pp. 65-79, 2011.

[7] B. Zhao, H. Aydin and D. Zhu, "Shared recovery for energy efficiency and reliability enhancements in real-time applications with precedence constraints," ACM Transactions on Design Automation of Electronic Systems (TODAES), vol. 18, no. 2, 23, 2013.

[8] G. M. Tchamgoue, K. H. Kim and Y. K. Jun, "Dynamic Voltage Scaling for Power-aware Hierarchical Real-Time Scheduling Framework," in Computational Science and Engineering (CSE), 2012 IEEE 15th International Conference on, Dec. 2012, pp. 540-547.

[9] M. Zakarya, N. Dilawar, M. A. Khattak and M. Hayat, "Energy Efficient Workload Balancing Algorithm for Real-Time Tasks over Multi-Core," World Applied Sciences Journal, vol. 22, no.10, pp. 1431-1439, 2013.

[10] G. Terzopoulos and H. D. Karatza, "Dynamic voltage scaling scheduling on power-aware clusters under power constraints," in Distributed Simulation and Real Time Applications (DS-RT), 2013 IEEE/ACM 17th International Symposium on, Delft, Netherlands, Oct. 2013, pp. 72-78.

[11] W. Wang, P. Mishra and S. Ranka, "Energy-Aware Scheduling with Dynamic Voltage Scaling," in Dynamic Reconfiguration in Real-Time Systems, Springer New York, 2013, pp. 85-127.

[12] F.Gruian, "Hard Real-Time Scheduling Using Stochastic Data and DVS Processors," in Proceedings of the International Symposium on Low Power Electronics and Design (ISLPED), Lund, Sweden, Aug. 2001, pp.46-51.

[13] C.-C.Han, K.-J Lin. and C.-J.Hou, "Distance-Constrained Scheduling and Its Applications to Real-Time Systems," IEEE Trans. Comput., vol. 45, no.7, pp. 814-826, July 1996.

[14] T. Ishihara and H. Yasuura, "Voltage scheduling problem for dynamically variable voltage processor," in Proceedings of International Symposium on Low Power Electronics and Design (ISLPED), Monterey, CA, USA, 1998, pp.197-202.

[15] 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, pp.2553-2563, June 2008.

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

[17] W.-C. Kwon and T. Kim, "Optimal voltage allocation techniques for dynamically variable voltage processors," ACM Trans. Embedded Comput. Sys., vol. 4, no. 1, pp.211-230, Feb. 2005.

[18] M. Li and F. F. Yao, "An efficient algorithm for computing optimal discrete voltage schedules," SIAM J. Comput., vol. 35, no.3, pp. 658-671, 2006.

[19] W.S. Liu Jane, Real-Time Systems. Prentice Hall PTR Upper Saddle River, NJ, USA, 2000.

[20] 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), Oct., 2001, pp.89-102.

[21] 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, Milwaukee, WI, 1995, pp.374-382.

[22] D. Zhu, D. Mosse and R. Melhem, "Power-Aware scheduling for and/or graphs in Real-time systems," IEEE Trans. Parallel and Distributed Sys., vol. 15, no. 9, pp. 849-864, Sep. 2004.

[23] 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, San Jose, CA, USA, Nov., 2000, pp. 365-368.

[24] T. D. Burd and R. W. Brodersen, "Design issues for dynamic voltage scaling," in Proceedings of the International Symposium on Low Power Electronics and Design (ISLPED), 2000, pp. 9-14.

[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, vol. 18, no. 2, Mar. 2001, pp.20-30.

[26] Intel Corporation, "Wireless Intel SpeedStep Power Manager-Optimizing power consumption for the intel PXA27x processor family," Wireless Intel SpeedStep® Power Manager White paper, 2004. from:http://download.intel.com/pressroom/kits/pxa27x/wp_wireless_ speedstep.pdf.

[27] Advanced Micro Devices Corporation, "AMD Athlon 64 Processor Power and Thermal Datasheet," 2006, from:http://www.amd.com/usen/assets/content_type/white_papers_and_tech_ docs/30430.pdf.

[28] M. Yishay and P.-S. Boaz, "Jitter Control in QoS Networks," IEEE/ACM Trans. Networking, vol. 9, no. 4, pp.492-502, Aug., 2001.

[29] R. Holte, A. Mok, L. Rosier, I. Tulchinsky and D. Varvel, "The Pinwheel: A Real-Time Scheduling Problem," in Proc. 22nd Hawaii Int'l Conf. System Science, Kailua-Kona, HI, Jan. 1989, pp. 693-702.

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

[31] H.-H. Lin and C.-W. Hsueh, "Applying pinwheel scheduling and compiler profiling for power-aware real-time scheduling," Real-time systems, vol.34, pp.37-51, 2006.

[32] 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.

[33] C. Oliveria, J. B. Kim and T. Suda, "An adaptive bandwidth reservation scheme for high-speed multimedia wireless networks," IEEE JSAC, vol. 16, no. 6, pp. 858-873, Aug. 1998.

[34] 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,” International Journal of Communication Systems, vol. 17, no. 6, pp. 535-552, March, 2004.

[35] J.-C. Chen, K. M. Sivalingam, R. Acharya and P. Agrawal, “Scheduling multimedia services for a low-power MAC in wireless and mobile ATM networks,” IEEE trans. on multimedia, vol. 1, no. 2, pp.187-201, June, 1999.

[36] S. Dennett, “The cdma2000 ITU-R RTT Candidate Submission,” Tele. Industry Association(TIA), June 2, 1998.

[37] ARM 8 Data-Sheet, “Document Number ARM DDI0080C,” Advanced RISC Machines Ltd, July 1996.

[38] T. D. Burd, T. Pering, A. Stratakos and R. Brodersen, “A dynamic voltage scaled microprocessor system,” IEEE journal of Solid-State Circuits, vol, 35, no. 11, pp. 1571-1580, Nov. 2000.

[39] T. Pering, T. Burd and R. Brodersen, “The Simulation and Evaluation of Dynamic Voltage Scaling Algorithms,” in Proceedings of the the 1998 international symposium on Low power electronics and design, ACM, Monterey, CA, USA, 1998, pp. 76-81.

[40] F. David, J. Carlyle and Roy Campbell, “Context-switch overheads for Linux on ARM platforms,” in Proceedings of the 2007 workshop on Experimental computer science, ACM 2007, pp. 3.