

Energy-Efficient Dynamic Task Scheduling Algorithms for DVS Systems

JIANLI ZHUO and CHAITALI CHAKRABARTI

Arizona State University

Dynamic voltage scaling (DVS) is a well-known low power design technique that reduces the processor energy by slowing down the DVS processor and stretching the task execution time. But in a DVS system consisting of a DVS processor and multiple devices, slowing down the processor increases the device energy consumption and thereby the system-level energy consumption. In this paper, we first use system-level energy consideration to derive the 'optimal' scaling factor by which a task should be scaled if there are no deadline constraints. Next, we develop dynamic task scheduling algorithms that make use of dynamic processor utilization and optimal scaling factor to determine the speed setting of a task. We present algorithm *duEDF* which reduces the CPU energy consumption, and algorithm *duSYS* and its reduced preemption version, *duSYS_PC*, which reduce the system-level energy. Experimental results on the Video-Phone task set show that when the CPU power is dominant, algorithm *duEDF* results in up to 45% energy savings compared to the non-DVS case. When the CPU power and device power are comparable, algorithms *duSYS* and *duSYS_PC* achieve up to 25% energy saving compared to CPU energy-efficient algorithm *duEDF*, and up to 12% energy saving over the non-DVS scheduling algorithm. However, if the device power is large compared to the CPU power, then we show that a DVS scheme does not result in lowest energy. Finally, a comparison of the performance of algorithms *duSYS* and *duSYS_PC* show that preemption control has minimal effect on system level energy reduction.

Categories and Subject Descriptors: D.4.1 [Operating Systems]: Process Management—Scheduling

General Terms: Algorithms

Additional Key Words and Phrases: Dynamic task scheduling, energy minimization, optimal scaling factor, DVS system, real-time

1. INTRODUCTION

With the rapid growth in the portable and mobile real-time device market, reducing the energy consumption to extend the battery lifetime has become an important design metric. Dynamic voltage scaling (DVS) is a well-known technique in low power design that trades off performance for power consumption by lowering the operating voltage/frequency. It achieves significant power saving due to the quadratic relationship between voltage and dynamic power.

In recent years, there has been significant amount of work done in energy efficient task scheduling for DVS processors. The work can be classified into static scheduling algorithms [Yao et al. 1995; Quan and Hu 2001; Jejurikar and Gupta 2004; Jejurikar et al.

This research was funded in part by the NSF S/I/UCRC Center for Low Power Electronics (EEC-9523338), and by the NSF (CSR-EHS 0509540).

Authors' addresses: Jianli Zhuo and Chaitali Chakrabarti, Department of Electrical Engineering, Arizona State University, Tempe, AZ 85287; email: jianli@asu.edu, chaitali@asu.edu.

Permission to make digital/hard copy of all or part of this material without fee for personal or classroom use provided that the copies are not made or distributed for profit or commercial advantage, the ACM copyright/server notice, the title of the publication, and its date appear, and notice is given that copying is by permission of the ACM, Inc. To copy otherwise, to republish, to post on servers, or to redistribute to lists requires prior specific permission and/or a fee.

© 20YY ACM 0000-0000/20YY/0000-0001 \$5.00

2004] based on apriori task information, two-phase algorithms [Shin et al. 2000; Krishna and Lee 2000; Gruian 2001; Shin et al. 2001; Jejurikar and Gupta 2005] that operate in two phases: an off-line phase (based on *WCET* or other execution time estimates) followed by an online phase where the slack is greedily absorbed, and purely dynamic algorithms [Yao et al. 1995; Shin and Choi 1999; Kim et al. 2002] that only operate in the online phase. Most of these techniques absorb the system slack greedily to reduce the processor operating voltage and thereby reduce the CPU dynamic power consumption.

Now consider the DVS system shown in Fig 1 which consists of a DVS processor interacting with other devices such as memory, flash drive, wireless interface, etc. Extending the execution time of a task by voltage/frequency scaling results in a reduction in the CPU dynamic power consumption but also results in an increase in the device energy consumption. The system-level energy, which is the sum of the CPU energy and the device energy, is a convex function of the scaling factor [Irani et al. 2003], and thus there exists a speed setting for which the system-level energy is minimum.

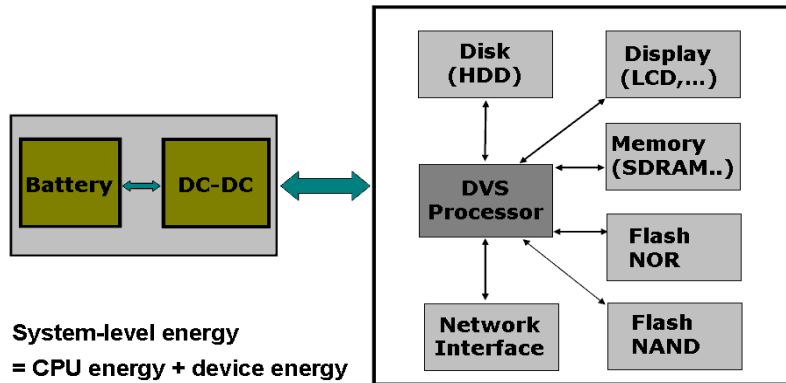


Fig. 1. System schematic

Recently, there has been some effort in developing task scheduling algorithms that minimize the system-level energy consumption [Jejurikar et al. 2004; Jejurikar and Gupta 2005; Kim et al. 2004; Choi and Chang 2004]. These include algorithms that procrastinate the execution of tasks in a static schedule [Jejurikar et al. 2004] and dynamic schedule [Jejurikar and Gupta 2005], an algorithm that reduces the number of preemptions in a dynamic schedule [Kim et al. 2004], and an algorithm that jointly considers the CPU and memory frequency [Choi and Chang 2004].

In this paper, we present low complexity dynamic task scheduling algorithms that minimize system-level energy for a DVS processor based system. This is an extension of the work that was presented in [Zhuo and Chakrabarti 2005]. We first make use of the convexity of the system-level energy curve to derive the *optimal scaling factor* by which a task should be scaled to minimize energy (if there are no deadline constraints). While the same concept has been introduced in [Jejurikar and Gupta 2004; Jejurikar et al. 2004] as *critical speed*, we present a detailed analytical analysis to help understand the contribution of the different components. Next, we introduce a speed setting method based on processor utilization and remaining workload estimation, and use it in conjunction with optimal scaling

factor to derive dynamic task scheduling algorithms. We propose algorithm *duEDF* which reduces the CPU energy consumption more efficiently (up to 45% energy savings compared to non-DVS scheduling) and with lower complexity compared to *lpSEH* [Kim et al. 2002] (which achieves the near optimal energy savings [Kim et al. 2002]). We also propose algorithms *duSYS* and its reduced preemption version, *duSYS_PC*, for system-level energy efficiency. Simulations on a randomized version of the Video-Phone task set show that when the CPU power and device power are comparable, algorithms *duSYS* and *duSYS_PC* achieve large energy savings (up to 25%) compared to the CPU energy-efficient algorithm *duEDF*, and up to 12% over the non-DVS scheduling algorithm. In fact, if the device power is large compared to the CPU power, then we show that a DVS scheme does not result in lowest energy. Finally, a comparison of the performance of algorithms *duSYS* and *duSYS_PC* show that preemption control has minimal effect on system level energy reduction.

The rest of the paper is organized as follows. The paper begins with preliminaries like task definitions, DVS system configuration and calculation of optimal speed setting in Section 2. In Section 3.1, an efficient dynamic task scheduling algorithm for systems with dominant CPU energy is proposed. In Section 3.2, dynamic task scheduling algorithms that reduce system-level energy are presented. The random simulation results are included in Section 4. The related work is described in Section 5. The paper is concluded in Section 6.

2. PRELIMINARIES

2.1 Task Definition

In this paper, we consider periodic tasks in which the relative deadline of a task is equal to its period, and the first instance of each periodic task is released at time 0. Table I lists the task parameters used in this paper. A periodic task $T^{[k]}$ has period $P^{[k]}$ and worst case execution time $WCET^{[k]}$. The execution of $T^{[k]}$ requires one or more devices. We define this set of devices as $\Phi^{[k]}$. The optimal scaling factor that minimizes the system energy consumption while executing $T^{[k]}$ is $\theta^{[k]}$. All task execution times are defined according to the highest frequency of the DVS processor.

There are multiple instances of each task, and each instance has the same worst case execution time and period, but has different arrival time, deadline and actual execution time. To make the notation simple, we relabel all task instances in the scheduling profile by label J indexed by a number indicating the order of execution. For instance, in Fig 2, task J_1 is the first instance of $T^{[1]}$, J_2 is the first instance of $T^{[2]}$, J_3 is the second instance of $T^{[1]}$, etc. J_2 is preempted by task J_3 and resumes after J_3 finishes; the resumed part is still deemed as part of J_2 and not given a new label. Since task J_3 is the second instance of task $T^{[1]}$, its period is $P_3 = 2\text{min}$, $WCET_3 = 0.85\text{min}$, arrival time is $a_i = 2\text{min}$ and deadline is $d_i = 4\text{min}$.

2.2 DVS System Configuration

Consider a typical DVS system that consists of one DVS processor and N devices denoted by D_1, D_2, \dots, D_N . The DVS processor can operate at different frequency and voltage settings in the active mode. For example, the Intel processor PXA270 (based on Xscale) can operate at 5 frequency levels ranging from 208MHz to 624MHz [Intel Corp. 2004]. We assume that a change in frequency is accompanied by a change in voltage, and of course, current. The

Table I. Summary of task parameters

$T^{[k]}$	the k -th periodic task in the task set.
$P^{[k]}$	the period of task $T^{[k]}$.
$WCET^{[k]}$	the worst case execution time of task $T^{[k]}$ when operated at the highest frequency.
$\Phi^{[k]}$	the set of all required devices of task $T^{[k]}$.
$\theta^{[k]}$	the scaling factor that minimizes the system energy consumption while executing task $T^{[k]}$.
J_i	the i -th task in the scheduling profile.
$WCET_i$	the worst case execution time of J_i when operated at the highest frequency, $WCET_i = WCET^{[k]}$ if $J_i \in T^{[k]}$.
P_i	the period of J_i , $P_i = P^{[k]}$ if $J_i \in T^{[k]}$.
Φ_i	the set of all required devices of J_i , $\Phi_i = \Phi^{[k]}$ if $J_i \in T^{[k]}$.
θ_i	the optimal scaling factor of J_i , $\theta_i = \theta^{[k]}$ if $J_i \in T^{[k]}$.
a_i	the arrival time of J_i , $a_i = (x-1) \times P^{[k]}$ if J_i is the x -th instance of $T^{[k]}$.
d_i	the deadline of J_i , $d_i = a_i + P_i$.
AET_i	the actual execution time of J_i when operated at the highest frequency.

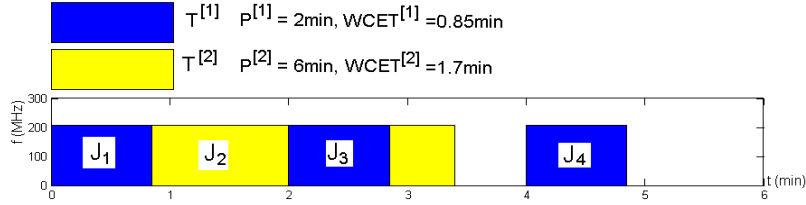


Fig. 2. Illustration of task definitions

devices (e.g. SDRAM, Flash Drive, etc.), on the other hand, operate at a single frequency and voltage in the active mode. Table II lists the parameters used to define the DVS system.

Table II. Parameters of a DVS system

s	the scaling factor of a task running on the DVS processor. $s = \frac{f_{cpu}(1)}{f_{cpu}(s)}$, $s \geq 1$
$f_{cpu}(s)$	the operating frequency of the DVS processor for scaling factor s
$P_{dyn}(s)$	the dynamic power of the DVS processor when operating at frequency $f_{cpu}(s)$
P_{static}	the static power (active state) of the DVS processor
$P_{cpu}(s)$	the power of the DVS processor at frequency $f_{cpu}(s)$, $P_{cpu}(s) = P_{dyn}(s) + P_{static}$
$E_{dyn}(s)$	the processor dynamic energy consumption when executing a task at frequency $f_{cpu}(s)$
$E_{static}(s)$	the processor static energy consumption (active state) when executing a task at frequency $f_{cpu}(s)$
$E_{cpu}(s)$	the total processor energy consumption when executing a task at frequency $f_{cpu}(s)$
Θ	the optimal scaling factor for a processor, i.e. $E_{cpu}(\Theta) = \min(E_{cpu}(s))$.
P_j^{std}	the standby power of device D_j
E_j^{std}	the standby energy of device D_j , $E_j^{std} = P_j^{std} \cdot \bar{\tau}$, where $\bar{\tau}$ is the time during which D_j is on
E_j^{act}	the active energy of device D_j

2.2.1 DVS Processor Energy Model. The power dissipation of a CMOS circuit can be broken down into: (1) *dynamic* power dissipation which is related to the switching activity and the operating voltage, (2) *static* power dissipation which is independent of the switching activity.

The dynamic power is calculated by $P_{dyn} = \alpha C_L V^2 f$, where α is the switching probability, C_L is the load capacitance, V is the voltage and f is the frequency. While frequency scaling in a DVS processor does not affect α and C_L , frequency scaling by a factor of s causes voltage to scale by a factor of s , the dynamic current to scale by a factor of s^2 , and the dynamic power to scale by a factor of s^3 . Thus $P_{dyn}(s) = s^{-3} P_{dyn}(1)$, where $P_{dyn}(1)$ is the dynamic power when there is no frequency scaling (corresponding to $s = 1$).

The static power P_{static} is defined as the sum of gate leakage power and the power consumption due to non-scalable components of the processor such as PLL circuitry and I/O subsystems [Jejurikar and Gupta 2004]. While the leakage power changes linearly with the voltage, the intrinsic power due to PLL circuitry and I/O subsystems is independent of the voltage [Jejurikar and Gupta 2004].

In the following analytical analysis, we make the simplifying assumption that P_{static} is a constant. This simplification is justified by the close match between the data sheet curves of real DVS processors and the analytical curves (as will be shown in Section 2.3). Throughout the paper, we use the data sheet curves, and this simplifying assumption does not impact any of the findings.

2.2.2 Device Energy Model. We assume that the devices have three modes of operations: active, standby(idle) and off. If D_j is required during the processing of a task, then when the task is being executed, the device consumes standby power P_j^{std} as well as active power P_j^{act} , and when the task is preempted, it only consumes standby power P_j^{std} . If D_j is not required, its power consumption $P_j^{off} = 0$. Also, there is an energy overhead when the device transitions from one mode to the other.

Active energy: Assume that an access to device D_j consumes energy E_j^{act} , which is a constant. In reality, E_j^{act} would depend on the type of access. Let $N_{i,j}$ be the number of access to device D_j during execution of task J_i . Then the total active energy of D_j consumed during execution of task J_i is given by $E_j^{act} \times N_{i,j}$. Since E_j^{act} is constant and $N_{i,j}$ is constant, the active energy of a device is independent of the frequency and voltage of the processor.

Standby energy: Since the device operates at a fixed frequency and voltage, the standby power P_j^{std} is assumed to be constant. The standby energy consumption is calculated by $E_j^{std} = P_j^{std} \times \bar{\tau}$, where $\bar{\tau}$ is the time duration for which this device is on. If the frequency of the DVS processor is scaled by a factor of s , the task execution time and $\bar{\tau}$ are extended by s , and standby energy E_j^{std} increases by s .

The synchronous memory in [Choi and Chang 2004] is an example of such a device. If the frequency of the memory is fixed, then the memory energy consumption consists of a part that is independent of the CPU frequency (similar to E_j^{act}) and a part that is linear with the CPU frequency (similar to E_j^{std}) [Choi and Chang 2004].

2.3 Optimal Scaling Factor

The dynamic energy consumption of the CPU decreases with frequency/voltage scaling. But scaling down the voltage does not always translate to lowering the total energy consumption. This is because lowering the voltage also results in increased execution time, which causes greater static energy (for CPU) and standby energy (for device) consumption.

In the rest of this section, we determine the maximum scaling factor of a DVS system

that results in minimum system-level energy when there are no deadline constraints. We call this scaling factor the *optimal scaling factor*. If we only consider the energy consumption of the DVS processor, the optimal scaling factor is only dependent on the CPU characteristics and is denoted by Θ . When we consider the total system energy consumption, the optimal scaling factor is dependent not only on the CPU but also on the devices required by the task. We denote the optimal scaling factor for task $T^{[k]}$ as $\theta^{[k]}$ (and for task J_i as θ_i).

Optimal scaling factor for CPU: Consider scheduling of a task J_i with arbitrary $WCET_i$ and AET_i ($AET_i \leq WCET_i$), but no deadline constraint. When the task J_i is scaled by a factor s , its execution times are extended by the factor s . The corresponding CPU dynamic energy consumption is given by

$$E_{dyn}(s) = P_{dyn}(s) \times (s \times AET_i) = s^{-2} \times P_{dyn}(1) \times AET_i = s^{-2} \times E_{dyn}(1) \quad (1)$$

and the static energy consumption of the CPU is given by

$$E_{static}(s) = P_{static} \times (s \times AET_i) = s \times P_{static} \times AET_i = s \times E_{static}(1) \quad (2)$$

The total energy consumption of the CPU is the sum of the dynamic energy and the static energy.

$$E_{cpu}(s) = E_{dyn}(s) + E_{static}(s) = (s^{-2} \times P_{dyn}(1) + s \times P_{static}) \times AET_i \quad (3)$$

Define $Q_{dyn}(s) = s^{-2} \times P_{dyn}(1)$, $Q_{static}(s) = s \times P_{static}$, and $Q_{cpu}(s) = Q_{dyn}(s) + Q_{static}(s)$. Then $E_{cpu}(s) = Q_{cpu}(s) \times AET_i$ and the value of s that minimizes $Q_{cpu}(s)$ will also minimize $E_{cpu}(s)$. The function $Q_{cpu}(s)$ is convex because its second derivative $Q''_{cpu}(s)$ exists and $Q''(s) \geq 0$ (note that $P_{dyn}(1) \geq 0$ and $s \geq 1$). The optimal scaling factor, Θ , can thus be obtained by setting $Q'_{cpu}(s) = (-2) \times P_{dyn}(1) \times s^{-3} + P_{static} = 0$. The optimal scaling factor is thus

$$\Theta = \left(\frac{2 \times P_{dyn}(1)}{P_{static}} \right)^{1/3} \quad (4)$$

From Eqn (4), we see Θ is a function of $P_{dyn}(1)$ and P_{static} . The dynamic power consumption $P_{dyn}(1)$ is proportional to the switching activity α , which may vary from task to task. Thus strictly speaking, the optimal scaling factor is task dependent. To simplify our analysis, we assume that $P_{dyn}(1)$ and P_{static} are constant, and so Θ is constant for a given processor.

Fig 3 shows $Q_{cpu}(s)$ of a processor with $P_{dyn}(1) = 500\text{mW}$ and $P_{static} = 200\text{mW}$. Convexity similar to Fig 3 have also been reported in [Jejurikar and Gupta 2004], [Irani et al. 2003] and [Zhai et al. 2004]. The optimal scaling factor for this configuration is $\Theta = \left(\frac{2 \times 500}{200} \right)^{1/3} = 1.71$.

For a real processor, we can use the data sheet to determine Θ . Table III shows the $Q_{cpu}(s)$ values of two real DVS processors: processor OMAP5912 from TI [Texas Instruments Inc. 2004] and processor PXA270 from Intel [Intel Corp. 2004]. Based on the frequency and power levels in the data sheets (the data sheet of OMAP5912 has been obtained from actual measurements), the values of $Q_{cpu}(s)$ is obtained by $s \times P_{cpu}(s)$. The $Q_{cpu}(s)$ data of the two processors show very different trends. While $Q_{cpu}(s)$ of OMAP5912 reduces as the scaling factor increases, $Q_{cpu}(s)$ of PXA270 first reduces and then increases. Consequently, the optimal scaling factor of OMAP5912 corresponds to the lowest frequency, and the optimal scaling factor Θ of PXA270 occurs at an intermediate frequency.

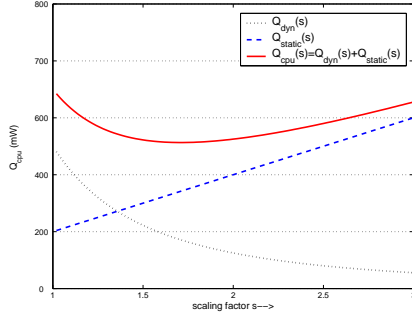


Fig. 3. Plot of $Q_{cpu}(s)$ vs. scaling factor s : $P_{dyn}(1)=500mW$, $P_{static}=200mW$, $\Theta=1.71$

Table III. Q_{cpu} calculation of processors (a) OMAP5912 (b) PXA270

(a) TI OMAP5912					
s	1	1.14	1.33	1.6	2
$f_{cpu}(s)(MHz)$	192	168	144	120	96
$P_{cpu}(s)(mW)$	270	215	160	120	80
$Q_{cpu}(s)(mW)$	270	245.7	213.3	192	160
(b) Intel PXA270					
s	1	1.2	1.5	2	3
$f_{cpu}(s)(MHz)$	624	520	416	312	208
$P_{cpu}(s)(mW)$	925	747	570	390	279
$Q_{cpu}(s)(mW)$	925	896.4	855	780	837

The difference in this trend can be attributed to the relative contribution of Q_{static} and Q_{dyn} . For instance, for the OMAP5912 processor, Q_{static} is significantly smaller than Q_{dyn} in the region of operation, and so the lowest value of Q_{cpu} occurs at the largest value of s .

Optimal scaling factor for CPU + Devices: Now consider a system with CPU and a set of devices. The total energy consumption of the system is $E_{cpu}(s) + E_{dev}(s)$, where $E_{cpu}(s)$ is the CPU energy, and $E_{dev}(s)$ is the total device energy. Since different tasks trigger different sets of devices, the optimal scaling factor for each task is different. Define $\theta^{[k]}$ to be the optimal scaling factor of a task which minimizes the system energy when there are no deadline constraints. $\theta^{[k]}$ is determined off-line and is calculated only once for each periodic task $T^{[k]}$. The procedure is described below.

Assume that a single task instance J_i ($J_i \in T^{[k]}$) is active and it is scaled by factor s . Then the system energy during the execution of J_i is

$$\begin{aligned} E(s) &= (P_{dyn}(s) + P_{static}) \times s \times AET_i + P_{dev}^{[k]} \times s \times AET_i + E^{act} \\ &= (s^{-2} \times P_{dyn}(1) + s \times (P_{static} + P_{dev}^{[k]})) AET_i + E^{act} \end{aligned} \quad (5)$$

where $P_{dev}^{[k]} = \sum_{D_j \in \Phi^{[k]}} P_j^{std}$ is the total standby energy of all required devices, $E^{act} = \sum_{D_j \in \Phi^{[k]}} (E_j^{act} \cdot N_{i,j})$ is the total active energy of all required devices, and $N_{i,j}$ is the number of access to device D_j during execution of task J_i .

Let $Q(s) = s^{-2} \times P_{dyn}(1) + s \times (P_{static} + P_{dev}^{[k]})$. Then $E(s) = Q(s) \times AET_i + E^{act}$. Since AET_i for a task is fixed and E^{act} is constant, the value of s that minimizes $Q(s)$ will also minimize $E(s)$. The function $Q(s)$ is convex and we can easily find its minimum by determining the value of s for which $Q'(s) = 0$. This value of s is referred to as the *optimal scaling factor* $\theta^{[k]}$ for task $T^{[k]}$.

$$\theta^{[k]} = \left(\frac{2 \times P_{dyn}(1)}{P_{static} + P_{dev}^{[k]}} \right)^{1/3} \quad (6)$$

Thus for every task, there is an optimal scaling factor for which the total energy (CPU + device) is minimized. Note that different task instances of the same periodic task may have different density of access, which could lead to different E^{act} (see equation (5)). However the value of E^{act} is independent of scaling, and so it does not impact the optimal scaling

factor.

From equation (6), we see that if $P_{static} + P_{dev}^{[k]} \geq 2P_{dyn}(1)$, then $\theta^{[k]} \leq 1$. In such cases, the voltage/frequency should not be scaled. If $P_{static} + P_{dev}^{[k]} << 2P_{dyn}(1)$, the CPU energy dominates, and we can ignore the device energy during task scheduling. Section 3.1 describes a dynamic task scheduling algorithm for such a scenario. If CPU and device energy are comparable, the optimal scaling factor θ plays an important role. Section 3.2 describes task scheduling algorithms for such cases.

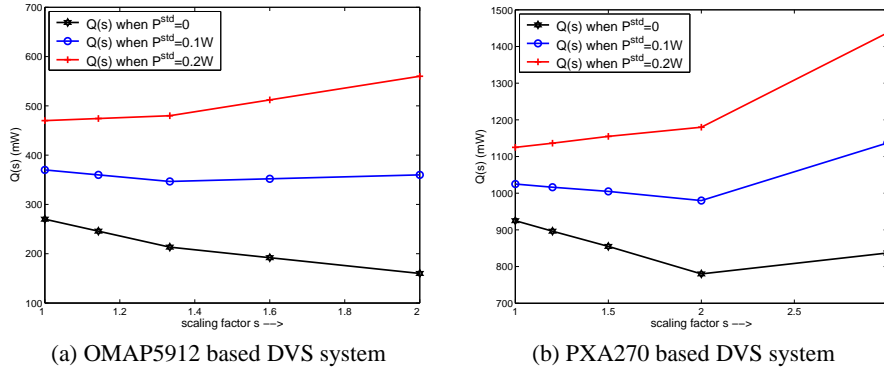


Fig. 4. Plot of $Q(s)$ vs s for different P^{std} : (a) CPU is OMAP5912 (b) CPU is PXA270

Since most DVS processors operate on a limited set of voltage and frequency levels, we can graphically find the optimal scaling point for each task. Fig 4 illustrates the variation of $Q(s)$ with s for a DVS system that consists of a DVS processor and devices with different standby power. For the OMAP5912 based DVS system shown in Fig 4.(a), when $P^{std} = 0$, the optimal scaling factor is actually the CPU optimal scaling factor, i.e. $\theta^{[k]} = \Theta = 2$ (the value of Θ is also given in Table III(a)); when $P^{std} = 0.1W$, the optimal scaling factor is $\theta^{[k]} = 1.33$; when P^{std} is 0.2W (such as SDRAM [Micro Technology Inc.]), $\theta^{[k]} = 1$. Thus we see that as the device standby power increases, the scaling factor reduces. Also if the device standby power is significant, as in PXA270 based system or OMAP5912 based system with $P^{std} = 0.2W$, then a non-DVS setting ($s = 1$) is most energy efficient.

3. ALGORITHM

In this section, we describe task scheduling algorithms, for the case when the CPU energy is significantly higher than the device energy (Section 3.1), and for the case when the device energy consumption is significant (Section 3.2).

We use the following definitions in the algorithm description.

- $RunQ$: the run queue containing all released tasks (with $a_i \leq t$, a_i is arrival time, t is current time point).
- J_{act} : the active task executed by the processor.
- J_{high} : the task with the highest priority in the $RunQ$.
- μ : the static processor utilization (EDF) over m periodic tasks, defined by $\mu = \sum_{k=1}^m \frac{WCET^{[k]}}{p^{[k]}}$.

3.1 Dynamic Task Scheduling With Minimum CPU Energy

In this section, we describe a task scheduling algorithm that minimizes the CPU energy consumption. This is applicable to systems where the CPU energy is significantly higher than the device energy.

3.1.1 Dynamic Processor Utilization. There are two important properties about EDF schedulability and energy efficiency that are used to derive the dynamic utilization $du(t)$. We know that for a periodic task set in which the relative deadline of each task is equal to its period and each periodic task releases the first instance at time 0, the sufficient and necessary condition for EDF schedulability is processor utilization $\mu \leq 1$ [Krishna and Shin 1997]. Furthermore, for a task set satisfying the condition $\mu \leq 1$, the optimal scaling factor to minimize the energy consumption while meeting all deadlines is constant and equal to μ^{-1} [Aydin et al. 2001].

We propose a concept called *dynamic processor utilization* which lets the active task J_{act} absorb all the available slack provided that the remaining tasks can be scaled by a factor not less than the μ^{-1} .

In order to calculate dynamic processor utilization, consider the time interval from current time t to d_{act} (the deadline of the active task J_{act}). Let Γ_{act} be the set of tasks whose arrival time is less than d_{act} ($a_i < d_{act}$). This includes tasks that have been preempted by J_{act} and also tasks that arrive in $[t, d_{act}]$ but does not include J_{act} . In the interval $[t, d_{act}]$, the total workload that needs to be finished is $(WCET_{act} - EX_{act}) + W_\Gamma$, where the term $(WCET_{act} - EX_{act})$ is the remaining execution time of J_{act} , and W_Γ is the workload from task set Γ_{act} .

$$W_\Gamma = \sum_{\forall J_i \in \Gamma_{act}} \max\left(0, \frac{WCET_i}{d_i - a_i} \times (\min(d_{act}, d_i) - a_i) - EX_i\right) \quad (7)$$

where EX_i (of task J_i) is the the workload that has already been executed before time t .

We first guarantee that the workload of Γ_{act} is executed by scaling factor μ^{-1} . Only then can the remaining slack be absorbed by J_{act} . The dynamic utilization when executing J_{act} is given by

$$du(t) = \frac{WCET_{act} - EX_{act}}{d_{act} - t - \mu^{-1} \times W_\Gamma} \quad (8)$$

The scaling factor due to dynamic utilization is given by $s_{du}(t) = \frac{1}{du(t)}$. Note that the above procedure does not push the workload in $[t, d_{act}]$ to be executed after d_{act} , or borrow slack from workload assigned to $> d_{act}$ (this is ensured by the max operation in Equation (8)). Fig 5 illustrates the computation of $du(t)$ for task $J_{act} = T^{[1]}$. Fig 5(a) shows how the task densities of each task contribute to the static utilization μ . Fig 5(b) shows the amount of workload that needs to be computed in duration $[t, d_{act}]$. Note that tasks in the set Γ_{act} whose deadline is greater than d_{act} are also guaranteed utilization μ at $t \geq d_{act}$. Fig 5(c) shows how the slack is absorbed by $T^{[1]}$ after guaranteeing that tasks in Γ_{act} maintain utilization μ .

3.1.2 Algorithm *duEDF*. In this paper, our goal is to develop dynamic task scheduling algorithms that (i) schedule the task set feasibly, and (ii) minimize the CPU energy consumption. Recall that in dynamic scheduling the *AET* of the task set is not known apriori.

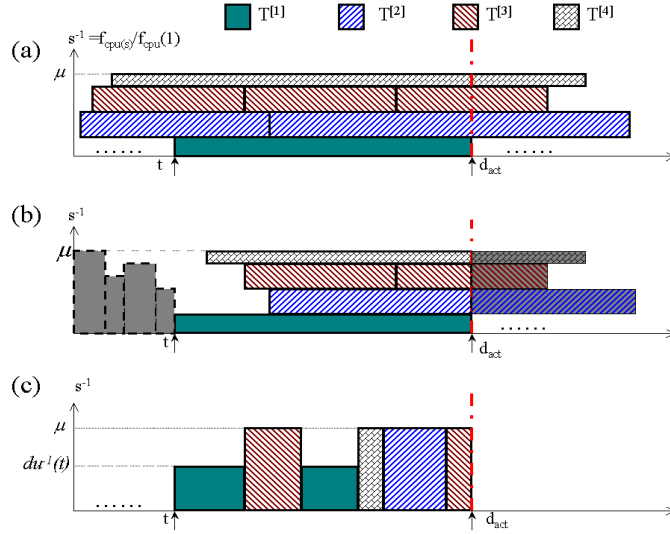


Fig. 5. Illustration of dynamic utilization:(a) The contribution of each task to the static utilization μ ; the height of each task bar is its task density; (b) The total workload which should be finished in duration $[t, d_{act}]$. The plain grey bars are the tasks that have been executed before t . The darker patterned bars are the tasks which could be executed after d_{act} ; (c) Allocation of all the slack to the active task $T^{[1]}$ while keeping the speed of other tasks as μ^{-1} .

We use the Earliest Deadline First (EDF) strategy to determine J_{act} from the $RunQ$. The scaling factor s_{act} of J_{act} is determined based on the deadline/execution time of the tasks in $RunQ$, and is not changed until $RunQ$ is updated. Every time a new task is released or the current J_{act} finishes, $RunQ$ is updated and the active task J_{act} is re-selected.

The scaling factor for J_{act} is given by

$$s_{act}(t) = \min \left(\max (s_{du}(t), \mu^{-1}), \Theta \right) \quad (9)$$

where the first item $s_{du}(t)$ is the scaling factor if only slack during $[t, d_{act}]$ is considered, the second item μ^{-1} is the system-level reference scaling factor based on static utilization, and the third item Θ is the scaling factor if only the CPU energy is considered. Since our goal is to generate a valid schedule that results in minimum energy consumption, we take the minimum of these two terms as the final scaling factor of J_{act} . The dynamic scheduling algorithm is shown in Algorithm 1.

3.1.3 Illustrative Example 1. A simple task set with tasks $T^{[1]}$, $T^{[2]}$, $T^{[3]}$ and utilization $\mu = 1$ is given in Table IV. Each task instance of the same periodic task is assumed to have the same AET. We assume that the CPU can be continuously scaled between $[1, 3]$ and that $P_{dyn}(1) = 500\text{mW}$ and $P_{static} = 200\text{mW}$. For such a processor, $\Theta = 1.71$ (refer to Fig 3 in Section 2.3).

Table V lists the calculation for each task instance, where item t is the start time, $du(t)$ is the dynamic utilization given by equation (8), and s is the scaling factor of the active task. The final task profile is given in Fig 6(e).

Next we compare the performance of $duEDF$ with the following three existing algorithms: 1) $lppsEDF$ [Shin et al. 2000] which implements slack lookahead as well as slack

Algorithm 1: Algorithm *duEDF*

```

1 while there exist tasks do
2   Put all released tasks into RunQ;
3   if RunQ == null then
4     Keep processor idle;
5     Continue;
6   end
7   Find  $J_{high}$  among RunQ;
8    $J_{act} = J_{high}$ ;
9   t = current time;
10  Determine  $s_{act}$  using equation (9);
11  Execute  $J_{act}$  till a new task released or  $J_{act}$  completes;
12  EndTime = current time;
13  if  $J_{act}$  not finished then
14    Increase  $EX_{act}$  by amount  $(EndTime - t)/s_{act}$ ;
15  else
16     $EX_{act} = WCET_{act}$ ;
17  end
18 end
    
```

Task	P	$WCET$	AET
$T^{[1]}$	10ms	4ms	2.4ms
$T^{[2]}$	10ms	4ms	2.4ms
$T^{[3]}$	30ms	6ms	1.2ms

Table IV. Task Specification of Example 1

Task	t	$du(t)$	s
J_1	0ms	1	1
J_2	2.4ms	0.715	1.4
J_3	5.76ms	0.730	1.37
J_4	10ms	0.667	1.5
J_5	13.6ms	0.625	1.6
J_6	20ms	0.667	1.5
J_7	23.6ms	0.625	1.6

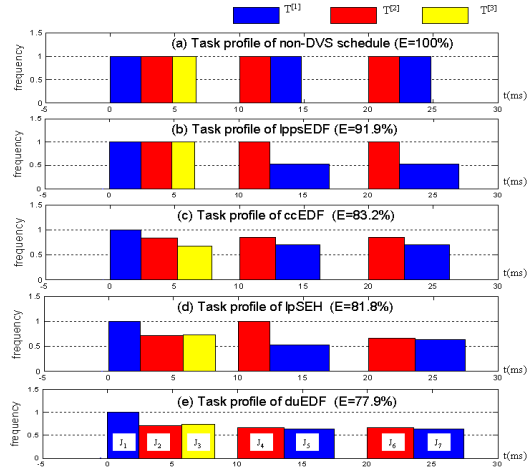
 Table V. Calculations pertaining to *duEDF* of Example 1


Fig. 6. Task profiles for the task set in Table IV

forwarding; 2) *ccEDF* (Cycle-conserving RT-DVS) [Pillai and Shin 2001] which updates the task density dynamically; 3) *lpSEH* algorithm [Kim et al. 2002] which uses slack estimation heuristics to absorb slack. Fig 6 sketches the task profiles for a hyper-period for the competing algorithms. The frequency values are normalized to the highest frequency, and the energy values are normalized to the value of non-DVS scheduling profile. This example shows that our algorithm *duEDF* utilizes slack more efficiently. Fig 6 also states the energy consumption values normalized to the nonDVS EDF schedule. Note that *duEDF* has the lowest energy consumption compared to the three competing dynamic task scheduling algorithms.

3.2 Dynamic Task Scheduling When Device Energy is Significant

In this section, we consider the problem of developing dynamic task scheduling algorithms that minimize the system-level energy (CPU energy + device energy) for cases where the device energy is significant. The algorithms use a combination of (i) *optimal speed* setting, which is the speed that minimizes the system energy for a specific task, and (ii) *limited preemption*, which reduces the number of preemptions in an attempt to reduce the device standby energy.

The skeleton of the proposed dynamic task scheduling algorithms *duSYS* and *duSYS_PC* is given in Algorithm 2.

Algorithm 2: Algorithm outline of *duSYS* and *duSYS_PC*

```

1 while there exist tasks do
2   Put all released task to  $RunQ$  and find  $J_{high}$ ;
3   Determine and execute  $J_{act}$  by duSYS, duSYS_PC;
4   if  $J_{act}$  not finished then
5      $J_{pre} = J_{act}$ ;
6     update  $EX_{pre}$  and put  $J_{pre}$  back into  $RunQ$ ;
7   end
8 end

```

3.2.1 *duSYS - duEDF + θ or ϑ* . In section 3.1, we presented Algorithm *duEDF* which is energy efficient if only CPU energy is considered. Since *duEDF* does not consider device energy when deciding on the scaling factor, we make the following enhancements.

First, we utilize the optimal scaling factor θ which takes into account the CPU and the device energy (refer to section 2.3). After calculating dynamic utilization $du(t)$ by equation (8), we compare $s_{du}(t)$ with the pre-calculated optimal θ_{act} ($\theta_{act} = \theta^{[k]}$ if $J_{act} \in T^{[k]}$) and set the scaling factor to $\min(\max(s_{du}(t), \mu^{-1}), \theta_{act})$.

Second, we consider the standby energy of the devices associated with the preempted tasks in the calculation of the optimal scaling factor. The corresponding scaling factor is referred to as ϑ .

For a DVS processor with discrete voltage / frequency settings, ϑ_{act} can be determined graphically (similar to the method in section 2.3). Since the total device power is larger, $\vartheta_{act} \leq \theta_{act}$, and the execution time of the active task T_{act} reduces in the presence of preempted tasks.

Algorithm 3: Determine J_{act} and s_{act} in *duSYS*

```

1  $J_{act} = J_{high}$ ;
2 if  $J_{pre}$  exists then
3    $s_{act} = \min(\max(s_{du}(t), \mu^{-1}), \vartheta_{act})$ ;
4 else
5    $s_{act} = \min(\max(s_{du}(t), \mu^{-1}), \theta_{act})$ ;
6 end
7 Execute  $J_{act}$  by factor  $s_{act}$  till it finishes, or a new task with higher priority is released;

```

The task selection and scaling factor setting calculation of *duSYS* (line 3 of Algorithm 2) is shown in Algorithm 3.

Since $\theta^{[k]}$ and $\vartheta^{[k]}$ consider the energy contribution of the devices, *duSYS* achieves significant energy savings compared to *duEDF* when device energy is significant.

3.2.2 *duSYS_PC* - Preemption Control. Task preemption helps in better slack utilization and has been extensively used in DVS schemes that are based on CPU energy minimization [Kim et al. 2004]. However, task preemption increases the lifetime of the preempted task, resulting in an increase in the standby device energy consumption of the preempted tasks.

Algorithm 4: Determine J_{act} and s_{act} by *duSYS_PC*

```

1 if  $J_{pre}$  exists then
2    $\tau_{delay} = (WCET - EX_{act}) \times (s_{du}(t) - 1)$ ;
3   if  $\tau_{delay} > 0$  then
4      $J_{act} = J_{pre}, s_{act} = s_{pre}$ ;
5     Execute  $J_{act}$  by factor  $s_{act}$  till  $\tau_{delay}$ , or till it completes, or with higher priority
      than  $J_{high}$  is released;
6   else
7      $J_{act} = J_{high}, s_{act} = \min(s_{du}(t), \vartheta_{high})$ ;
8     Execute  $J_{act}$  by factor  $s_{act}$  till it finishes, or a new task with higher priority is
      released;
9   end
10 else
11    $J_{act} = J_{high}$ ;
12    $s_{act} = \min(s_{du}(t), \theta_{act})$ ;
13   Execute  $J_{act}$  by factor  $s_{act}$  till it finishes, or a new task with higher priority is released;
14 end

```

In this section, we describe a mechanism to reduce the preemption, referred to as *Preemption Control*. At time t , assume that J_{high} (the task with the highest priority) is considering preempting the execution of J_{pre} (the active task in the previous time cycle). If we can delay the execution of J_{high} by a time duration τ_{delay} without deadline violation, and if J_{pre} finishes in τ_{delay} , then we can successfully avoid preemption. In some cases, J_{pre} may not be able to finish in τ_{delay} . By delaying the preemption of J_{pre} , the standby energy of all devices used by J_{pre} is reduced.

The algorithm with preemption control, *duSYS_PC*, is listed in Algorithm 4. The critical parameter is the delay duration τ_{delay} . If τ_{delay} is calculated using the formula $\tau_{delay} = (WCET - EX_{act}) \times (s_{du}(t) - 1)$, the schedulability is not compromised. The argument is along the same lines as [Kim et al. 2004]. If J_{high} can finish before its deadline by operating at scaling factor s ($s = s_{du}(t)$ in this algorithm), it can be delayed by τ_{delay} and still finish before the deadline with $s = 1$. So the slack τ_{delay} can be used to execute J_{pre} .

3.2.3 Illustrative Example 2. A simple task set with tasks $T^{[1]}, T^{[2]}$ is given in Table VI. The processor used in the simulation is same as that in Illustrative Example 1 (Section 3.1.3).

We run the following algorithms for one hyper-period (60 ms) and compare their system-level energy consumptions: (1) Non-DVS EDF scheduling, (2) *duEDF*, (3) *duSYS*, and (4) *duSYS_PC*. Fig 7 shows the normalized frequency profile for each algorithm, and the energy consumption normalized to that of non-DVS EDF scheduling. We see that while *duEDF* exploits the slack efficiently, its energy consumption is the highest. This is because stretching the execution time results in the devices being on for a longer period of time, thereby increasing the device energy. We also see that for this example, preemption control resulted in much greater energy reduction. However, the number of preemptions is typically quite small, and the energy reduction due to preemption control in a longer duration is insignificant.

Task	$T^{[1]}$	$T^{[2]}$
P	20ms	60ms
$WCET$	10ms	20ms
AET	7ms	16ms
Φ	D_1	D_2
$p^{std}(W)$	0.35	0.35
θ	1.22	1.22

Table VI. Task Specification of Example 2

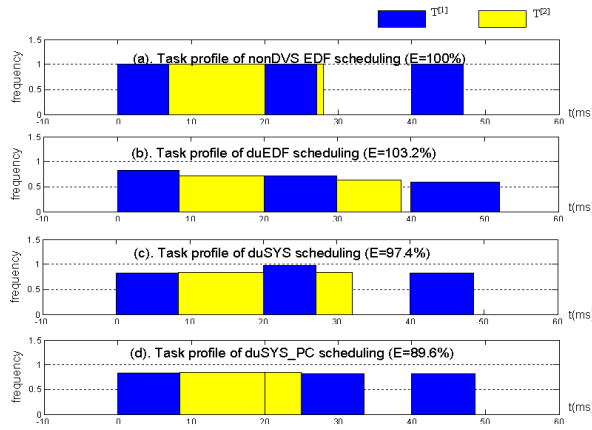


Fig. 7. Task profiles and normalized energy for the task set in Table VI

4. EXPERIMENTAL RESULT

We evaluated the performance of our algorithms by running simulations on a randomly generated task set. The random task set was based on the Video-Phone system [Shin et al. 2001] which consists of 4 periodic tasks: speech encoding, speech decoding, video encoding and video decoding as shown in Table VII. We chose the Video-Phone task set since it has been used by many others to evaluate task scheduling algorithms [Shin et al. 2000; Shin et al. 2001; Shin and Choi 1999; Kim et al. 2002; Choi et al. 2005]. While most of them used the exact task parameters in Table VII, we introduced randomness into the task parameters to get a more general result in our experiments.

Table VII. Task specification of Video-Phone application

Task	video encoding	video decoding	speech encoding	speech decoding
period (ms)	66.667	66.667	40.000	40.000
WCET (ms)	50.386	9.826	1.844	1.383
Ave.ET (ms)	13.099	1.460	0.907	0.680

The period of the tasks is randomly chosen from 10ms to 100ms, $WCET$'s of the tasks are chosen to satisfy the utilization constraint (0.1, 0.2, etc.), AET for each task instance

is evenly distributed in $[0.5WCET, WCET]$. All the task sets are run in a hyper-period (i.e. the L.C.M of all periods). We varied the processor utilization of the tasks from 0.1 to 1.0 with a step of 0.1, and ran 100 random task sets for each utilization value.

Three DVS processors were used in the experiments:

- CPU_A with $P_{dyn}(1) = 500\text{mW}$, $P_{static} = 200\text{mW}$; scaling factor ranging continuously from 1 to 3, $\Theta = 1.71$.
- OMAP5912 with 5 available scaling factors ranging from 1 to 2, and $\Theta = 2$ (see Table III(a)).
- PXA270 with 5 available scaling factors ranging from 1 to 3, and $\Theta = 2$ (see Table III(b)).

When no task is being executed, we assume that the processor stays in the idle mode with power consumption equal to 5% of $P_{cpu}(1)$. The delay of changing the frequency from one value to another (including entering or exiting idle mode) is ignored.

4.1 Performance of duEDF

4.1.1 *Experiment 1.* We first evaluate the performance of *duEDF* with *lppsEDF* [Shin et al. 2000], *ccEDF* (Cycle-conserving RT-DVS) [Pillai and Shin 2001] and *lpSEH* [Kim et al. 2002], with respect to only CPU energy consumption.

Fig 8(a) shows the normalized CPU energy consumption for different utilization values when CPU_A is used as the processor. Algorithm *duEDF* has the lowest energy consumptions for all the utilization values; the algorithm savings are in the order of 10-30% with respect to the non-DVS case. Fig 8(b) shows the complexity evaluated by the CPU time consumed by running the scheduling algorithm (for one hyperperiod). *duEDF* has slightly higher complexity compared to *lppsEDF* and *ccEDF*, but significantly lower complexity compared to *lpSEH* for all utilization factors.

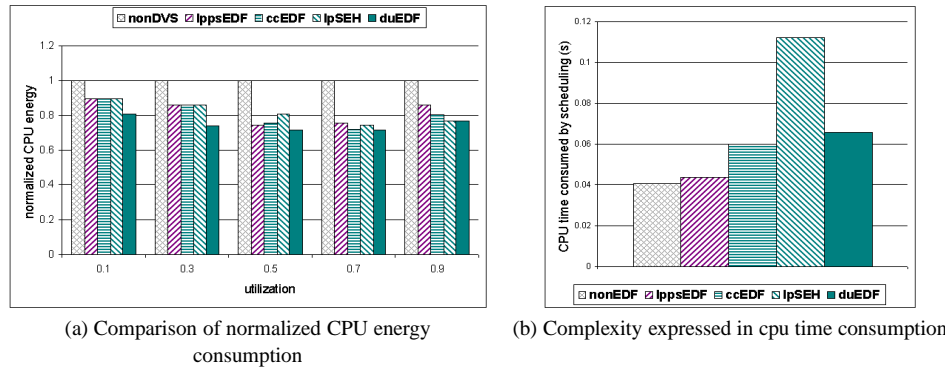


Fig. 8. Experiment 1. Performance of CPU_A with respect to (a) CPU energy, (b) complexity.

Fig 9 shows the CPU energy consumption when OMAP5912 and PXA270 (with only 5 discrete scaling factors) are used. Algorithm *duEDF* achieves up to 45% energy savings compared to the non-DVS scheduling when OMAP5912 is used and only up to 20% savings when PXA270 is used. The low energy saving of PXA270 can be explained with the help of Table III(b) which shows that its largest possible energy saving,

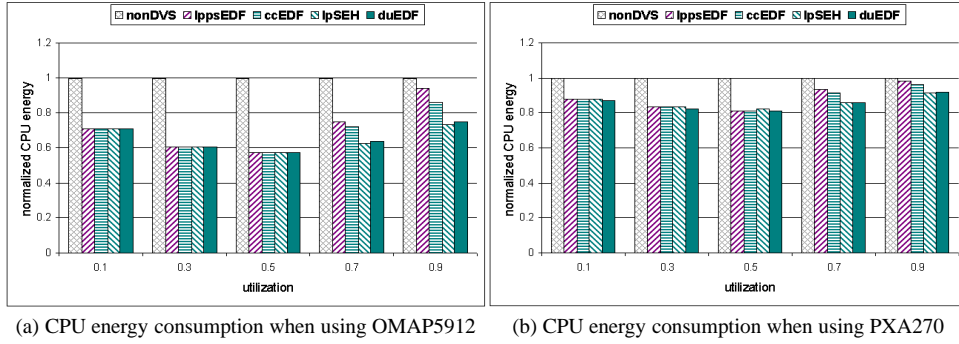


Fig. 9. Experiment 1. Comparison of CPU energy consumption when processors (a) OMAP5912 (b) PXA270 are used.

$1 - Q_{cpu}(\Theta)/Q_{cpu}(1)$, is around 20%. In both cases, the energy savings of *duEDF* and *lpSEH* are comparable. If however the CPU complexity is taken into account (as shown in Fig 8.(b)), the performance of *duEDF* is better.

4.2 Performance of duSYS and duSYS_PC

In this section, we compare the performance of the proposed system level energy-efficient algorithms. We include *duEDF* and *lpSEH_DP* [Kim et al. 2004] in the comparison since *duEDF* is very efficient in reducing the processor energy consumption, and *lpSEH_DP* (which is upgraded from *lpSEH* with delay preemption logic) is the best algorithm (to date) for achieving system level energy efficiency for dynamic task scheduling. All the energy consumption values shown in this section are normalized with respect to non-DVS scheduling.

4.2.1 Experiment 2. The task setting is the same as before but now each task is associated with a set of devices. Specifically, $\Phi^{[1]} = \{D_1\}$, $\Phi^{[2]} = \{D_1, D_2\}$, $\Phi^{[3]} = D_1$ and $\Phi^{[4]} = \emptyset$. The standby power for D_1 is $P_1^{std} = 0.2 \times P_{cpu}(1)$ and for D_2 is $P_2^{std} = 0.3 \times P_{cpu}(1)$. In addition, we assume that Device D_j consumes energy value equal to $P_j^{std} \times 0.01ms$ when it transits from active state to idle state, and $P_j^{std} \times 0.02ms$ vice versa. Furthermore, we add another energy penalty for preemption that is due to storing and retrieving the data of the preempted task. In our experiments, this value is $P_{cpu}(1) \times 0.01ms$ per preemption. All delay penalties during transition are ignored.

Fig 10 and Fig 11 shows the system-level energy consumption of the competing algorithms when run on different processors. Note that the system-level energy includes CPU energy, device standby energy, device state transition energy and preemption energy. On the continuously scalable CPU processor (see Fig 10), while all the algorithms have energy savings of $\sim 20\%$ when only CPU energy is considered, the savings drops to 5-12% when system-level energy is considered. Algorithm *lpSEH_DP* does significantly worse and has higher energy consumption than non-DVS scheduling for low utilization. This is partly because *lpSEH_DP* does not consider the optimal scaling factor.

On the OMAP5912 and PXA270 processors (see Fig 11), Algorithm *lpSEH_DP* and *duEDF* have the worst performance. The bad performance of *duEDF* is attributed to the fact that it only considers the CPU optimal scaling factor and does not take into account the contribution from the device standby power. Compared to the non-DVS scheduling,

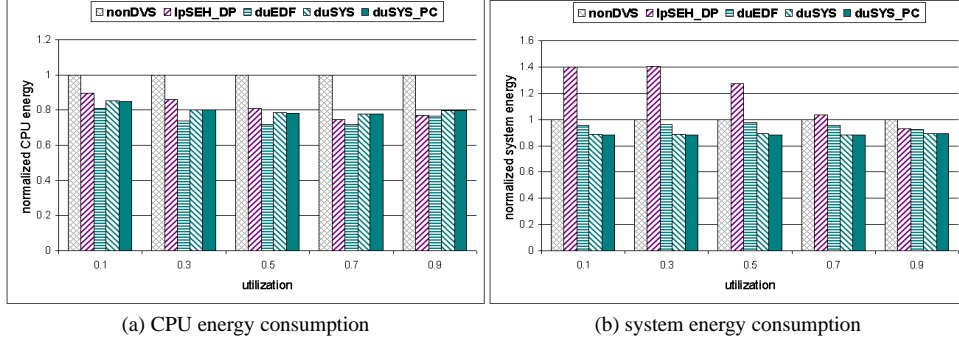


Fig. 10. Experiment 2. Comparison of (a) CPU energy and (b) system-level energy for a CPU_A based system.

the system level energy efficient algorithms *duSYS* and *duSYS_PC* save up to 12% energy when CPU_A is used, up to 15% energy when OMAP5912 is used, and only up to 3% when PXA270 is used. In all three cases, the improvement of *duSYS_PC* over *duSYS* is very limited.

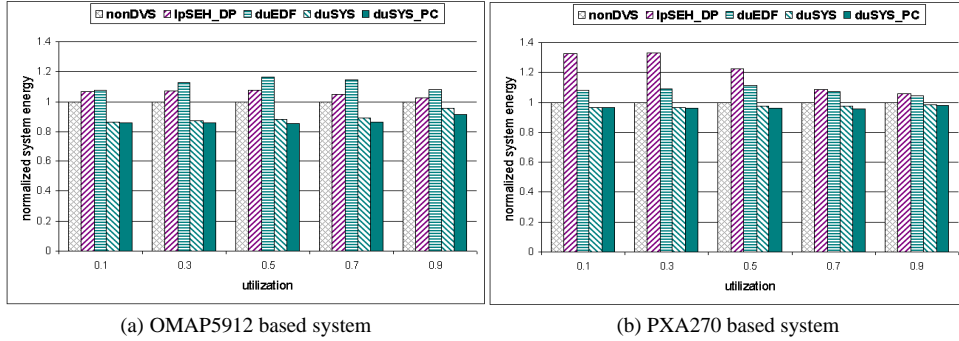


Fig. 11. Experiment 2. Comparison of system-level energy for a system using (a) OMAP5912 and (b) PXA270.

4.2.2 Experiment 3. In this experiment, we study the effect of device power on the system-level energy consumptions. Three cases are considered here: (i) CPU power dominates, (ii) CPU power and device power are comparable and (iii) device power dominates. We use the same task settings as Experiment 2, but vary the device power as follows: Case (i) $P_1^{std} = P_2^{std} = 0.05 \times P_{cpu}(1)$; Case (ii) $P_1^{std} = 0.2 \times P_{cpu}(1), P_2^{std} = 0.3 \times P_{cpu}(1)$; Case (iii) $P_1^{std} = P_2^{std} = 2 \times P_{cpu}(1)$.

Fig 12 compares the system-level energy consumption when the task set is run on the OMAP5912 processor for $\mu = 0.7$. The result shows that while Algorithm *duSYS* and *duSYS_PC* do better than *lpSEH_DP* and *duEDF*, the energy savings achieved by Algorithms *duSYS* and *duSYS_PC* are quite small when the device power increases. When device power dominates (case (iii)), Algorithms *duSYS* and *duSYS_PC* have only up to 5% energy savings compared to non-DVS scheduling. Considering the complexity overhead, in this case, non-DVS scheduling is the best choice. This observation is substantiated by

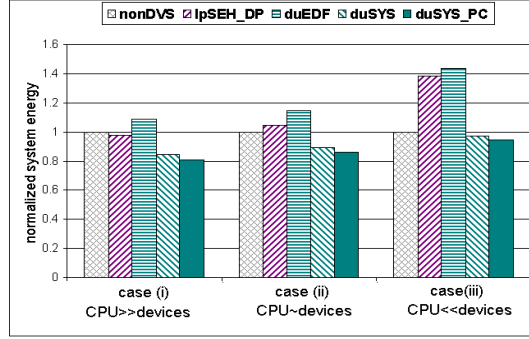


Fig. 12. Experiment 3. Comparison of system-level energy consumptions when running the task set with $\mu = 0.7$ on a OMAP5912 based system for the following cases: (i) CPU power dominates, (ii) CPU power and device power are comparable and (iii) device power dominates.

equation (6). If $P_{static} + P_{dev}^{[k]} \geq 2P_{dyn}(1)$, the optimal scaling factor is $\theta = \left(\frac{2P_{dyn}(1)}{P_{static} + P_{dev}^{[k]}}\right)^{1/3} \leq 1$, implying that DVS should not be used.

5. RELATED WORK

In recent years, there has been significant amount of work done in energy-efficient task scheduling for DVS processors. In this part, we focus on dynamic task scheduling algorithms that handle periodic tasks in single processor systems. Multiple processor scheduling and mixed-task scheduling (periodic task + aperiodic task) problems are beyond the scope of this paper and have not been considered here.

The existing dynamic task scheduling algorithms can be classified into: (1) those that are based purely on slack absorption at run time [Yao et al. 1995; Shin and Choi 1999; Kim et al. 2002], and (2) those that consist of two phases: off-line (based on *WCET* or other execution time estimation) followed by an online phase where the slack from the variation of execution time is greedily absorbed [Shin et al. 2000; Krishna and Lee 2000; Gruian 2001; Shin et al. 2001; Jejurikar and Gupta 2005].

The algorithm *lpfps* proposed in [Shin and Choi 1999] is a power conscious fixed-priority scheduling algorithm. It is based on a simple run-time checking scheme: the processor can either be shut down if there is no released task or adopt the speed such that the current active task is finished at its deadline or the release time of the next task. The advantage of this algorithm is its simplicity which makes it easy to embed into an operating system (OS) kernel. The disadvantage is that the algorithm can not fully exploit the slack because the active task can only be scaled when there is no other released task. The performance of algorithm *lpfps* can be improved by adding an off-line phase which pre-scale all tasks based on processor utilization. The corresponding algorithm, *lpps*, achieves significantly higher energy saving [Shin et al. 2000]. Even though these two algorithms are based on RM scheduling, it is easy to modify them to EDF scheduling by using dynamic priority and EDF processor utilization instead of the original settings.

An online scheduling algorithm called AVR (Average Rate) is proposed in [Yao et al. 1995] where the processor speed is set to be the summation of the densities of all live tasks. The density of a task T_j is defined as $d_j = \frac{R_j}{b_j - a_j}$, where R_j is the number of CPU cycles,

b_j is the deadline, a_j is the arrival time. The competitive ratio with respect to the optimal off-line algorithm is also provided.

The performance of dynamic scheduling is highly dependent on the accuracy of the run-time slack estimation. The algorithms in [Kim et al. 2002; 2003] derive efficient slack estimation algorithms *lpSEH* for dynamic-priority scheduling [Kim et al. 2002], and for static-priority scheduling [Kim et al. 2003]. Three types of slack are identified: the sum of the unused times from higher priority task instances already completed, the currently remaining time of the active task, and the sum of the slack times from the lower priority task instances. The estimation algorithm updates the unused (remaining) time of each task, and uses it to calculate the available slack for the current active task. The algorithm achieves very high slack utilization at the expense of larger run-time complexity. In fact, a comparison of the energy consumption of six EDF scheduling algorithms on DVS processor in [Kim et al. 2002] shows that *lpSEH*[Kim et al. 2002] has near-optimal energy performance.

Most of the above algorithms [Yao et al. 1995; Quan and Hu 2001; Jejurikar and Gupta 2004; Shin et al. 2000; Krishna and Lee 2000; Jejurikar and Gupta 2005; Shin and Choi 1999; Kim et al. 2002; Pillai and Shin 2001; Kim et al. 2003] do not change the processor speed until the current active task is finished or preempted. The slack utilization can clearly be increased if the processor speed can be varied (based on the workload) during the execution of the task. The intra-task scheduling algorithms [Shin et al. 2001; Shin and Kim 2001; Seo et al. 2004] determine the start-up speed off-line by analyzing the execution path, and then scaling down/up the processor speed based on branch selections during run-time. The algorithm in [Shin et al. 2001] uses the worst case execution path delay to calculate the start-up speed, while the algorithms in [Shin and Kim 2001] and [Seo et al. 2004] use the average case execution path delay. These are determined by load-profiling in [Shin and Kim 2001] and by considering branch probabilities in [Seo et al. 2004]. A more accurate intra-task scheduling algorithm based on stochastic estimates is proposed in [Gruian 2001]; the processor speed is determined optimally and can be changed from cycle to cycle.

All of the earlier work on DVS scheduling is based on operating the processor at the lowest possible speed provided the deadlines are not violated. The assumption is that lowering the frequency and voltage lowers the energy consumption. This is certainly true if only dynamic power consumption is considered. However, in sub-micron technology, the leakage power is as important as the dynamic power, and scaling the voltage as low as possible may not result in reduction of total power (dynamic + leakage). This observation has been exploited in recent work [Jejurikar and Gupta 2004; Choi and Chang 2004; Zhai et al. 2004; Rao and Vrudhula 2005].

The method in [Jejurikar and Gupta 2004; Irani et al. 2003] determines the voltage scaling factor (called critical speed) that minimizes the total CPU energy consumption for executing a given task. While analytical results based on the convexity of the curve are presented in [Irani et al. 2003], the results are validated on a 70-nm technology Crusoe processor in [Jejurikar and Gupta 2004]. For a DVS processor with discrete speed settings, [Rao and Vrudhula 2005] shows that a two speed setting is sufficient to achieve the minimum energy consumption. There are other factors that may also affect the critical speed. At the processor level, the increase in temperature causes an increase in the leakage, resulting in lower critical speed. At the system level, components such as DC-DC converter

affect the critical speed as demonstrated recently in [Choi et al. 2005].

In order to reduce the system-level energy consumption, it is important to also consider the energy consumption of other components such as memory, flash drive, wireless interface, etc. In such a system, extending the execution time of a task results in a reduction in the CPU dynamic energy consumption but also causes an increase in the device energy consumption. The method in [Jejurikar et al. 2004] is aimed at reducing the system-level (CPU + device) energy consumption and considers the critical speed of the DVS processor that takes into account the leakage of the devices. While the system in [Jejurikar et al. 2004] assumes that the voltage/frequency of the devices are not scalable, the SDRAM based system in [Choi and Chang 2004] determines the frequency of the memory and the frequency of the DVS processor that result in the minimum system-level energy consumption. A heuristic algorithm that makes use of the critical speed and the procrastination algorithm in [Jejurikar and Gupta 2004] (to reduce the overhead during state transition) is proposed in [Jejurikar et al. 2004] for static scheduling, and in [Jejurikar and Gupta 2005] for dynamic scheduling. We should mention that the definition of *optimal scaling factor* in our work is the same as *critical speed* in [Jejurikar and Gupta 2004; Jejurikar et al. 2004; Irani et al. 2003]. While the key concepts were presented in [Jejurikar et al. 2004], the first part of this paper provided an analytical analysis for better understanding of the factors that contribute to system-level energy consumption. Furthermore, this paper presents dynamic task scheduling algorithms as opposed to static scheduling algorithm in [Jejurikar et al. 2004].

The system-level energy consumption has been reduced by lowering the number of preemptions in [Kim et al. 2004]. The assumption is that voltage and frequency scaling results in increased task preemptions and extended task lifetime which translates to higher energy consumption. The proposed methods include one based on accelerating the current task if a higher priority task is going to be released, another based on delaying the higher priority task even it is already released. The experiments show that delay preemption is an efficient way to reduce the number of preemptions, which in turn reduces the system energy consumption.

6. CONCLUSION

In this paper, we considered the problem of developing dynamic task scheduling algorithms that minimize the system-level energy consumption (sum of CPU and device energy). We first determined the 'optimal' scaling factor by which a task should be scaled to minimize energy when there are no deadline constraints. Then we developed energy efficient dynamic task scheduling algorithms for periodic tasks based on optimal scaling factor and dynamic utilization. When only the CPU energy is considered, the proposed algorithm *duEDF* achieves higher energy saving (up to 45% over the non-DVS scheduling) and has much lower complexity compared to the existing algorithm *lpSEH* [Kim et al. 2002]. When the system energy (CPU energy + device energy) is considered, the proposed algorithms *duSYS* and *duSYS_PC* use a combination of optimal speed setting and limited preemption. For the case when the CPU power and device power are comparable, *duSYS* and *duSYS_PC* achieve large energy savings (up to 25%) compared to the CPU-energy efficient algorithm *duEDF* and up to 15% energy saving over the non-DVS scheduling algorithm. If the device power is large compared to the CPU power, then we show that a DVS scheme does not result in lowest energy. Additionally, the experiments show that the addition of preemption

control logic does not improve the quality of our solution.

Now if the devices in our system also operate at multiple voltage/frequency levels, the device power can be broken up into the dynamic part which is scalable and the static part which is not scalable. While this would change the system-level energy curve, the convexity property would still hold, and there would still be an optimal scaling factor. We are currently looking at extending the proposed dynamic task scheduling algorithms to determine the speed setting of both the CPU and the device for minimum system-level energy consumption.

REFERENCES

- AYDIN, H., MELHEM, R., MOSS, D., AND ALVAREZ, P. M. 2001. Dynamic and aggressive scheduling techniques for power-aware real-time systems. In *Proceedings of the IEEE Real-Time Systems Symposium (RTSS)*. 95–105.
- CHOI, Y. AND CHANG, N. 2004. Memory-aware energy-optimal frequency assignment for dynamic supply voltage scaling. In *Proceedings of the IEEE International Symposium on Low Power Electronics and Design (ISLPED)*. 387–392.
- CHOI, Y., CHANG, N., AND KIM, T. 2005. Dc-dc converter-aware power management for battery-operated embedded systems. In *Proceedings of the IEEE Design Automation Conference (DAC)*. 895–900.
- GRUIAN, F. 2001. Hard real-time scheduling for low energy using stochastic data and dvs processors. In *Proceedings of the IEEE International Symposium on Low Power Electronics and Design (ISLPED)*. 46–51.
- INTEL CORP. 2004. *Intel PXA270 Processor Electrical, Mechanical, and Thermal Specification*. <http://www.intel.com/design/pca/applicationsprocessors/datashts/280002.htm>.
- IRANI, S., SHUKLA, S., AND GUPTA, R. 2003. Algorithms for power savings. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SoDA)*. 37–46.
- JEJURIKAR, R. AND GUPTA, R. 2004. Leakage aware dynamic voltage scaling for real-time embedded systems. In *Proceedings of the IEEE Design Automation Conference (DAC)*. 275–280.
- JEJURIKAR, R. AND GUPTA, R. 2005. Dynamic slack reclamation with procrastination scheduling in real-time embedded systems. In *Proceedings of the IEEE Design Automation Conference (DAC)*. 111–116.
- JEJURIKAR, R., PEREIRA, C., AND GUPTA, R. 2004. Dynamic voltage scaling for systemwide energy minimization in real-time embedded systems. In *Proceedings of the IEEE International Symposium on Low Power Electronics and Design (ISLPED)*. 78–81.
- KIM, W., KIM, J., AND MIN, S. 2002. A dynamic voltage scaling algorithm for dynamic-priority hard real-time systems using slack time analysis. In *Proceedings of the Design Automation and Test in Europe Conference (DATE)*. 788–794.
- KIM, W., KIM, J., AND MIN, S. 2004. Preemption-aware dynamic voltage scaling in hard real-time systems. In *Proceedings of the IEEE International Symposium on Low Power Electronics and Design (ISLPED)*. 393–398.
- KIM, W., KIM, J., AND MIN, S. L. 2003. Dynamic voltage scaling algorithm for fixed-priority real-time systems using work-demand analysis. In *Proceedings of the IEEE International Symposium on Low Power Electronics and Design (ISLPED)*. 396–401.
- KIM, W., SHIN, D., YUN, H., KIM, J., AND MIN, S. 2002. Performance comparison of dynamic voltage scaling algorithms for hard real-time systems. In *Proceedings of the IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. 219–228.
- KRISHNA, C. M. AND LEE, Y. H. 2000. Voltage-clock-scaling adaptive scheduling techniques for low power in hard real time systems. In *Proceedings of the IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. 156–165.
- KRISHNA, C. M. AND SHIN, K. G. 1997. *Real-time systems*. New York : McGraw-Hill.
- MICRO TECHNOLOGY INC. <http://www.micron.com>.
- PILLAI, P. AND SHIN, K. G. 2001. Real-time dynamic voltage scaling for low-power embedded operating systems. In *Proceedings of the ACM Symposium on Operating Systems Principles (SOSP)*. 89–102.
- QUAN, G. AND HU, X. 2001. Energy efficient fixed-priority scheduling for real-time systems on variable voltage processors. In *Proceedings of the IEEE Design Automation Conference (DAC)*. 828–833.

- RAO, R. AND VRUDHULA, S. 2005. Energy optimal speed control of devices with discrete speed sets. In *Proceedings of the IEEE Design Automation Conference (DAC)*. 901–906.
- SEO, J., KIM, T., AND CHUNG, K. 2004. Profile-based optimal intra-task voltage scheduling for hard real-time applications. In *Proceedings of the IEEE Design Automation Conference (DAC)*. 87–92.
- SHIN, D. AND KIM, J. 2001. A profile-based energy-efficient intra-task voltage scheduling algorithm for hard real-time applications. In *Proceedings of the IEEE International Symposium on Low Power Electronics and Design (ISLPED)*. 271–274.
- SHIN, D., KIM, J., AND LEE, S. 2001. Low-energy intra task voltage scheduling using static timing analysis. In *Proceedings of the IEEE Design Automation Conference (DAC)*. 438–443.
- SHIN, Y. AND CHOI, K. 1999. Power conscious fixed priority scheduling for hard real-time systems. In *Proceedings of the IEEE Design Automation Conference (DAC)*. 134–139.
- SHIN, Y., CHOI, K., AND SAKURAI, T. 2000. Power optimization of real-time embedded systems on variable speed processors. In *Proceedings of the IEEE International Conference on Computer Aided Design (ICCAD)*. 365–368.
- TEXAS INSTRUMENTS INC. 2004. *OMAP5912 Multimedia Processor Power Management Reference Guide (Rev. A)*. <http://focus.ti.com/lit/ug/spru753a/spru753a.pdf>.
- YAO, F., DEMERS, A., AND S. SHENKER. 1995. A scheduling model for reduced cpu energy. *IEEE Annual Foundations of Computer Science*, 374–382.
- ZHAI, B., BLAAUW, D., SYLVESTER, D., AND FLAUTNER, K. 2004. Theoretical and practical limits of dynamic voltage scaling. In *Proceedings of the IEEE Design Automation Conference (DAC)*. 868–873.
- ZHUO, J. AND CHAKRABARTI, C. 2005. System-level energy-efficient dynamic task scheduling. In *Proceedings of the IEEE Design Automation Conference (DAC)*. 628–631.

Received September 2005; revised February 2006; accepted September 2006