System-Level Energy-Efficient Dynamic Task Scheduling[*]

Jianli Zhuo
Department of Electrical Engineering
Arizona State University, Tempe, AZ, 85287
jianli.zhuo@asu.edu

Chaitali Chakrabarti
Department of Electrical Engineering
Arizona State University, Tempe, AZ, 85287
chaitali@asu.edu

ABSTRACT

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 present dynamic task scheduling algorithms for periodic tasks that minimize the system-level energy (CPU energy + device standby energy). 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 numbers of possible preemptions. For the case when the CPU power and device power are comparable, these algorithms achieve up to 43% energy savings compared to [1], but only up to 12% over the non-DVS scheduling. If the device power is large compared to the CPU power, we show that DVS should not be employed.

Categories and Subject Descriptors

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

General Terms: Algorithms.

Keywords

Dynamic task scheduling, energy minimization, optimal scaling point, DVS system, real-time

1. INTRODUCTION

With the rapid growth in the portable and mobile 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 dynamic 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 [2, 3, 4] based on apriori task information, two-phase algorithms [5, 6, 7, 8] 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 pure dynamic algorithms [2, 9, 10] 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 a DVS system which consists of a DVS processor interacting with other devices such as SDRAM memory, flash drive, wireless interface, etc. Extending the execution time of a task results in a reduction in the CPU dynamic power consumption but it also results in an increase in the device standby energy consumption. Since the device energy consumption can be comparable to that of a DVS processor (such as StrongArm), their contribution cannot be ignored.

Recently, there has been some effort in developing task scheduling algorithms that minimize the system-level energy consumption defined by CPU energy + device energy [11, 1]. These include an algorithm that procrastinates the execution of tasks in a static schedule [11], and an algorithm that reduces the number of preemptions for a dynamic schedule [1].

In this paper, we consider the problem of developing dynamic task scheduling algorithms for periodic tasks that minimize the system-level energy. 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 and thereby reduces the device standby energy. For the case when the CPU power and device power are comparable, the proposed algorithms *duSYS* and *duSYS_PC* achieve large energy savings (up to 30%) compared to the CPU-energy efficient algorithm *duEDF*, and up to 43% energy saving compared to the existing system-level energy efficient algorithm *lpSEH_DP* [1]. However, these algorithms save only up to 12% 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.

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. The new scheduling algorithms are presented in Section 3. The simulation results with random examples are described in Section 4. The paper is concluded in Section 5.

2. BACKGROUND

2.1 Task Definition

In this paper, we consider periodic tasks in which the relative deadline of a task is equal to its period. We denote the $k - th$ periodic task in the task set as $T^{[k]}$, which has period $P^{[k]}$, worst case execution time $WCET^{[k]}$, required device set $\Phi^{[k]}$, and optimal scaling factor $\theta^{[k]}$ which corresponds to the scaling factor that minimizes the energy consumption for executing task $T^{[k]}$ (more details in Section 2.3). 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 $WCET$, P, Φ and θ, but has different arrival time, deadline and execution time. To make the notation simple, we relabel all task instances in the scheduling profile by label J with the index representative of the order of execution. Task instance J_i has parameters $WCET_i$, P_i, Φ_i, θ_i, arrival time a_i and deadline d_i.

2.2 DVS System Configuration

Consider a typical DVS system that consists of one DVS processor (also referred to as CPU) and N devices denoted by D_1, $D_2,..D_N$. The DVS processor can operate at different frequency and voltage settings in the active mode. The devices (e.g. SDRAM, Flash Drive, etc.), on the other hand, operate at a single frequency and voltage in the active mode.

2.2.1 DVS Processor Energy Model

The following parameters are defined for the DVS processor: f_{proc}, P_{proc} represent the operating frequency and power; f_{ref}, P_{ref} represent the reference frequency and power (which are also the highest values of f_{proc}, P_{proc}). The scaling factor is defined as $s = \frac{f_{ref}}{f_{proc}}, s \geq 1$. For long channel devices, scaling the frequency by a factor of s causes voltage to scale by a factor of s, the current to scale by a factor of s^2, and the power to scale by a factor of s^3. Thus $P_{proc} = s^{-3} P_{ref}$. The optimal scaling factor which minimizes the CPU energy is defined as Θ.

2.2.2 Device Energy Model

We assume that the devices have three modes of operation: active, standby and shutdown. The standby power of device D_j is P_j^{std}, and the standby energy is $E_j^{std} = P_j^{std} \cdot \tau$, where τ is the time during which D_j is on. E_j^{atv} is the active energy of D_j, which is the active power P_j^{atv} times the number of access during task execution. E_j^{off} is the energy when the device is shutdown and we assume that $E_j^{off} = 0$. If D_j is required by a task, then the device energy consumption is $E_j^{std} + E_j^{atv}$ when the task is being executed, and E_j^{std} when the task is preempted. If a task is scaled by a factor s ($s \geq 1$), E_j^{std} is increased by a factor s. There is, however, no change in E_j^{atv}.

2.3 Optimal Scaling Point

Our goal is to minimize the total energy consumption of the system $E_{proc} + E_{dev}$, where E_{proc} is the processor energy, and E_{dev} is the device energy given by $E_{dev} = E^{std} + E^{atv}$. The optimal scaling factor for such a system is a function of $E_{proc} + E_{dev}$.

Different tasks trigger different sets of devices, and the optimal scaling factor for each task is different. Let $\theta^{[k]}$ 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

$$E(s) = P_{proc} \cdot s \cdot AET_i + P_{dev}^{[k]} \cdot s \cdot AET_i + E^{atv}$$
$$= (s^{-2} P_{ref} + s \cdot P_{dev}^{[k]}) AET_i + E^{atv}$$

where $P_{dev}^{[k]} = \sum_{D_j \in \Phi^{[k]}} P_j^{std}$ is the summation of standby energy of all devices in the required device set $\Phi^{[k]}$. E^{atv} does not depend on s and is assumed constant during execution of task J_i.

Let $Q(s) = s^{-2} \cdot P_{ref} + s \cdot P_{dev}^{[k]}$. Then $E(s) = Q(s) \cdot AET_i + E^{atv}$. Since AET_i for a task is fixed and E_{atv} is constant, the value of s that minimizes $Q(s)$ will also minimize $E(s)$.

Since $Q(s)$ is a convex function, we can easily find its minimum by calculating the value of s which makes $Q'(s) = 0$. This value is the *optimal scaling factor* $\theta^{[k]}$ for task $T^{[k]}$.

$$\theta^{[k]} = \left(\frac{2P_{ref}}{P_{dev}^{[k]}} \right)^{1/3} \tag{1}$$

Thus for every task, there is an optimal scaling factor for which the total energy (CPU + device) is minimized.

If $P_{dev}^{[k]} \geq 2P_{ref}$, then $\theta^{[k]} \leq 1$. In such cases, DVS should not be employed. If $P_{dev}^{[k]} << 2P_{ref}$, the CPU energy dominates, and we can ignore the device energy during task scheduling. If CPU and device energy are comparable, the optimal scaling factor θ plays an important role.

Since most DVS processors operate on a limited set of voltage and frequency levels, we can also numerically find the optimal scaling point for each task. Let $Q(s) = Q_{proc}(s) + Q_{dev}(s)$, where $Q_{proc}(s) = s \cdot P_{proc}$ and $Q_{dev}(s) = s \cdot Pd^{[k]}$.
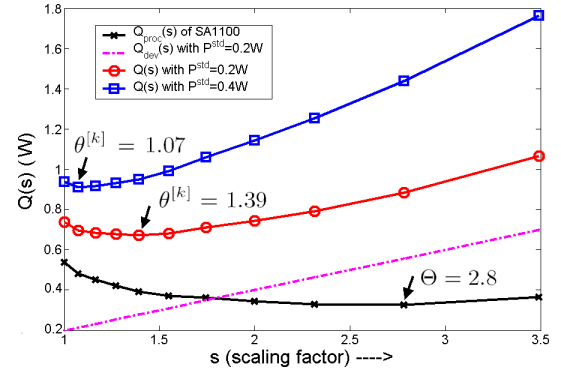

Figure 1: Q(s) vs s for a SA1100 based system

Fig 1 illustrates the variation of $Q_{proc}(s)$, $Q_{dev}(s)$ and $Q(s)$ with s for a DVS system that consists of a StrongArm SA1100 processor [12] and devices with different standby power. Note that $Q_{proc}(s)$ (line with cross) is not a monotonically decreasing function. This trend in $Q_{proc}(s)$ has also been pointed out in [4, 13]. In fact, if only the SA1100 processor is considered, the optimal scaling factor is $\Theta = 2.8$. Note that Θ differs from processor to processor. For the system with $P^{std} = 0.2W$ (such as SDRAM [14]), $Q_{dev}(s)$ is shown by the dotted line, and the corresponding $Q(s)$ is shown by the line with circles. The optimal scaling factor for this case is $\theta^{[k]} = 1.39$, and the corresponding frequency is 148 MHz. For the system when P^{std} is $0.4W$ (such as flash drive [11]), $\theta^{[k]}$ is 1.07 as shown by the line with squares, and the corresponding frequency is 192 MHz. Thus we see that as the device standby power increases, the scaling factor reduces. Note that the optimal scaling factor does not depend on the active power of the device.

3. DYNAMIC TASK SCHEDULING

Next we present task scheduling algorithms that (i) generate feasible task sets, and (ii) minimize the energy consumption (CPU energy + device standby energy) of the system. We consider dynamic scheduling where the AET of the task set is not known apriori.

We use the following definitions in the algorithm description. $RunQ$ is the run queue containing all released tasks, J_{act} is the active task executed by the processor, J_{high} is the task with highest priority in $RunQ$, and J_{pre} is the active task in the previous cycle which is not finished yet.

We first use dynamic utilization information to get the maximum scaling factor of J_{act} at time t. For a hyper-period H (L.C.M of periods of tasks), The maximum scaling factor is

$$du(t) = \frac{H - t - \mu^{-1} \cdot (W - WCET_{act})}{WCET_{act}}, (0 \leq t \leq H) \quad (2)$$

where W is the estimated remaining workload, $\mu = \sum_{k=1}^{m} \frac{WCET^{[k]}}{P^{[k]}}$ is the static utilization in EDF (Earliest Deadline First) scheduling. The speed derived setting by equation (2) lets the active task absorb all the available slack and at the same time ensures that the remaining tasks can be scaled by a factor not less than μ^{-1}.

The skeleton of the proposed EDF based dynamic task scheduling algorithms is given in Algorithm 1.

Algorithm 1 Skeleton of the proposed algorithms

1: $W = H \cdot \mu$;
2: **while** $time() < hyperperiod$ **do**
3:     determine s_{act} and execute J_{act} using s_{act};
4:     **if** J_{act} is not finished **then**
5:         $ExePart = current\_duration/s_{act}$;
6:         $W = W - ExePart$;
7:         $WCET_{act} = WCET_{act} - ExePart$;
8:         $AET_{act} = AET_{act} - ExePart$;
9:     **else**
10:        $W = W - WCET_{act}$;
11:     **end if**
12: **end while**

3.1 duEDF - CPU energy only

Algorithm *duEDF* minimizes the CPU energy (and not the system energy). It is applicable to a system where the CPU energy is dominant. In this algorithm, J_{high} is always selected as J_{act}, and the scaling factor is given by $s_{act} = min(du(t), \frac{d_{act}-t}{WCET_{act}}, \Theta)$. Here the first term exploits the maximum available slack, the second term ensures that the deadline is not violated, and the third term considers the optimal scaling factor of the processor, Θ.

3.2 duSYS - CPU + device energy

Algorithm *duSYS* considers the optimal scaling factor (that takes into account both the CPU energy and device energy) in the derivation of s_{act}.

1: IF J_{pre} exists, THEN $s_{act} = min(du(t), \frac{d_{act}-t}{WCET_{act}}, \vartheta_{act})$;
2: ELSE $s_{act} = min(du(t), \frac{d_{act}-t}{WCET_{act}}, \theta_{act})$; ENDIF

Here ϑ_{act} is the optimal scaling factor when we consider all the devices that are on. These include devices associated with preempted tasks that are in standby and the devices associated with J_{act}. θ_{act}, on the other hand, is the optimal scaling factor when we consider only the devices associated with J_{act}. Since the total

device power when tasks are preempted is larger, $\vartheta_{act} \leq \theta_{act}$, the scaling factor of the active task, s_{act}, is smaller.

3.3 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. However, task preemption increases the lifetime of the preempted task, resulting in an increase in the standby device energy consumption of the preempted tasks.

Here, we describe an algorithm *duSYS_PC* that reduces the preemption of tasks scheduled by *duSYS*. At time t, assume J_{high} is about to preempt the execution of J_{pre}. 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. Note that τ_{delay} has to respect the deadline of J_{high}.

The idea of delay preemption is borrowed from [1] and is used to calculate $\tilde{\tau} = WCET_{high} \times (du(t) - 1)$. In this algorithm, τ_{delay} is given by $\tau_{delay} = \tilde{\tau} \times \sum_{i=1}^{m} x^{(i-1)}$, where $x = \frac{1}{\mu \cdot s_{pre}}$ and m is typically taken to be 10. (details omitted.)

4. EXPERIMENTAL RESULTS

In this section, we compare the performance of the algorithms for randomly generated tasks. We include *lpSEH_DP* [1] in the comparison since it 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 the non-DVS scheduling result.

We vary the processor utilization of the tasks from 0.1 to 0.9 with a step of 0.2 and run 100 random task sets for each utilization value. Each task set consists of 4 periodic tasks, and is generated in the following way: period of the tasks is randomly chosen from 0.1s to 1s, $WCET$s of the tasks are chosen to satisfy the utilization constraint given apriori (0.1, 0.3, etc.), AET for each task instance is given by a Gaussian distribution with mean $m = 0.8 \times WCET$ and variance $\sigma = 0.067 \times WCET$. All the task sets are run in a hyper-period (i.e. the L.C.M of all periods).

The DVS processor is StrongArm SA1100 [12] with 11 frequency values, and power ranging from 0.1W to 0.54W. We assume that the processor does not consume energy when it is idle and the overhead of moving from one voltage to another is negligible.

4.1 Experiment 1

In this experiment, the processor power and the device standby power are comparable. The required device set for each task is fixed: $\Phi^{[1]} = \{D_1\}$, $\Phi^{[2]} = \{D_1, D_2\}$, $\Phi^{[3]} = D_1$ and $\Phi^{[4]} = \emptyset$. The standby power for devices is $P_1^{std} = 0.2W$ and $P_2^{std} = 0.4W$ (typical values for SDRAM and flash drive [11]).

CPU energy: When only CPU energy is considered, algorithms *duEDF* and *lpSEH_DP* have energy saving up to 40% compared to non-DVS scheduling (see Fig 2). Algorithms *duSYS* and *duSYS_PC*, on the other hand, have only 25% energy saving compared to the non-DVS case.

System-level energy: When the processor + device energy is considered, *duEDF* and *lpSEH_DP* have consistently higher energy consumption compared to *duSYS* and *duSYS_PC*. In fact, *duEDF* and *lpSEH_DP* have much higher energy consumption compared to non-DVS scheduling for low processor utilization. For high processor utilization, the increase is not as significant. *duSYS* and *duSYS_PC* have up to 12% energy saving compared to the non-DVS case (see Fig 3).
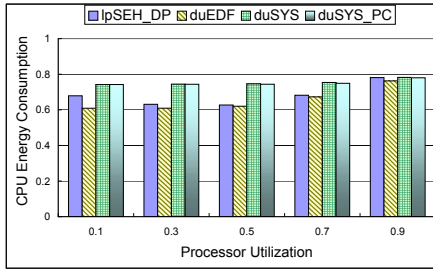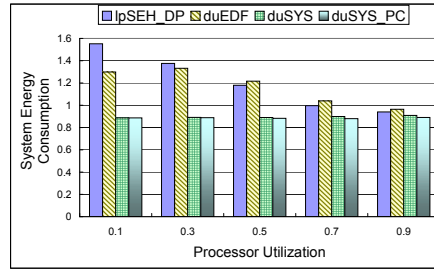
Figure 2: Processor energy (Exp.1)

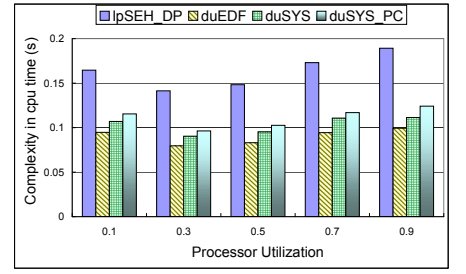**Figure 3: System energy (Exp.1)**

Figure 4: Complexity (Exp.1)

From Fig 3, we also see that the energy consumption of Algorithms *duSYS* and *duSYS_PC* is comparable. This implies that preemption control may not really be required for algorithms that minimize CPU + device energy. If, however, the cost of preemption is accurately taken into account, *duSYS_PC* would have a noticeably lower energy consumption compared to *duSYS*.

Complexity: The complexity of each algorithm shown in Fig 4 is represented by the average processor time spent on running the algorithm. *lpSEH_DP* has significantly high runtime complexity compared to the other algorithms. The proposed algorithms have comparable complexity with *duSYS_PC* being the most complex.

4.2 Experiment 2

In this experiment, we study the effect of standby power on the system level energy consumption. We consider three settings of the standby power: (i) $P_1^{std} = 20mW$ and $P_2^{std} = 40mW$, (ii) $P_1^{std} = 0.2W$ and $P_2^{std} = 0.4W$, and (iii) $P_1^{std} = 2W$ and $P_2^{std} = 4W$. The CPU (SA1100) power ranges from $0.1W$ to $0.54W$. For this experiment, the required device set for each task is the same as that of Experiment 1. The system energy consumption (normalized to non-DVS scheduling) for processor utilization $\mu = 0.5$ is shown in Fig 5.
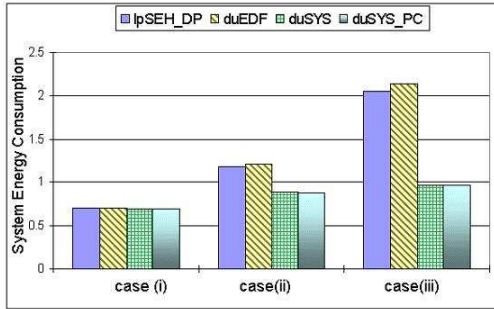

Figure 5: Comparison of system energy for Exp.2

When the processor power dominates, as in Case(i), DVS based schemes reduce the system energy consumption. When the processor power is comparable to the device power (as in Case(ii)), the proposed algorithms *duSYS* and *duSYS_PC*, which explicitly take into account device power, do much better than *duEDF* and *lpSEH_DP*, but just slightly better than the non-DVS case. When the device power dominates as in Case(iii), a DVS based scheme that does not take into account device power, has significantly higher energy consumption. For this case, *duSYS* and *duSYS_PC* have energy consumption that is comparable to the non-DVS case. Thus for such scenarios, non-DVS scheduling is the best choice. This observation is substantiated by equation (1). If the device power $P_{dev}^{[k]} \geq 2P_{ref}$, the optimal scaling factor is $\theta = \left(\frac{2P_{ref}}{P_{dev}^{[k]}}\right)^{1/3} \leq 1$, implying that DVS should not be used.

5. CONCLUSION

In this paper, we consider the problem of developing dynamic task scheduling algorithms for periodic tasks that minimize the system-level energy. We first determine the 'optimal' scaling factor by which a task should be scaled to minimize energy (if there are no deadline constraints). Then we propose scheduling algorithms that use a combination of *optimal speed* setting and *limited preemption*. When the CPU power and device power are comparable, experiments on randomly generated task sets show that the proposed algorithms *duSYS* and *duSYS_PC* achieve large energy savings (up to 43%) compared to existing dynamic scheduling algorithm [1], and up to 12% compared to the non-DVS case. We also show that if the device power is large compared to CPU power, then a non-DVS scheme is the one with the lowest energy consumption.

6. REFERENCES

[1] W. Kim, J. Kim and S. Min, "Preemption-aware dynamic voltage scaling in hard real-time systems," *Proc. ISLPED*, pp. 393–398, 2004.

[2] F.Yao, A.Demers, and S.Shenker, "A scheduling model for reduced cpu energy," *IEEE Annual Foundations of Computer Science*, pp. 374–382, 1995.

[3] G. Quan, and X. Hu, "Energy efficient fixed-priority scheduling for real-time systems on variable voltage processors," *Proc. DAC*, pp. 828–833, 2001.

[4] R. Jejurikar, and R. Gupta, "Leakage aware dynamic voltage scaling for real-time embedded systems," *Proc. DAC*, pp. 275–280, 2004.

[5] Y.Shin, K.Choi, and T.Sakurai, "Power optimization of real-time embedded systems on variable speed processors," *Proc. ICCAD*, pp. 365–368, 2000.

[6] C.M Krishna and Y.H Lee, "Voltage-clock-scaling adaptive scheduling techniques for low power in hard real time systems," *Proc. Real-Time Technology & Applications Symp.*, pp. 156–165, 2000.

[7] F. Gruian, "Hard real-time scheduling for low energy using stochastic data and dvs processors," *Proc. ISLPED*, pp. 46–51, 2001.

[8] D. Shin, J.Kim and S. Lee, "Low-energy intra task voltage scheduling using static timing analysis," *Proc. DAC*, pp. 438–443, 2001.

[9] W. Kim, J.Kim and S. Min, "A dynamic voltage scaling algorithm for dynamic-priority hard real-time systems using slack time analysis," *Proc. DATE.*, pp. 788–794, 2002.

[10] J.Zhuo and C.Chakrabarti, "An efficient dynamic task scheduling algorithm for battery powered dvs system," *Proc.ASP-DAC*, pp. 846–849, 2005.

[11] R. Jejurikar, C. Pereira, and R. Gupta, "Dynamic voltage scaling for systemwide energy minimization in real-time embedded systems," *Proc. ISLPED*, pp. 78–81, 2004.

[12] Intel Corp., *http://www.intel.com*.

[13] B.Zhai, D.Blaauw, D.Sylvester and K.Flautner, "Theoretical and practival limits of dynamic voltage scaling," *Proc. DAC*, pp. 868–873, 2004.

[14] Micro Technology, Inc., *http://www.micron.com*.