System-Level Energy-Efficient Scheduling for Hard Real-Time Embedded Systems

Linwei Niu
Department of Math and Computer Science
Claflin University
Orangeburg, SC 29115
linwei.niu@claflin.edu

***Abstract*—In this paper, we present a system level dynamic scheduling algorithm to minimize the energy consumption by the DVS processor and multiple non-DVS peripheral devices in a hard real-time system. We show that the previous work which adopts the *critical speed* as the lower bound for scaling might not be most energy efficient when the energy overhead of shutting-down/waking-up is not negligible. Moreover, the widely used statically defined *break even idle time* might not be overall energy efficient due to its independence of job execution situations. In our approach, we first present an approach to enhance the computation of *break even idle time* dynamically. Then a dynamic scheduling approach is proposed in the management of speed determination and task preemption to reduce the energy consumption of the processor and devices. Compared with existing research, our approach can effectively reduce the system-level energy consumption for both CPU and peripheral devices.**

I. Introduction

Energy minimization has become one of the major goals in embedded systems design and power aware scheduling has proven to be an effective way to reduce the energy consumption for today's pervasive computing systems. Two main types of techniques are reported in the literature. The first one is called *dynamic voltage scaling* (DVS) which updates the processor's supply voltages dynamically. The second one is commonly known as the *dynamic power down* (DPD), *i.e.*, to shut down a processing unit when it is not in use.

Extensive researches on power aware scheduling have been conducted for reducing the processor energy consumption. While the processor is one of the major power hungry units in the system, real-time embedded systems are often implemented with more than one hardware resources. Recently, a number of researches (e.g. [1], [2], [3], [4], [5]) are reported to reduce the energy consumption for systems consisting of both core DVS processors and peripheral devices. Jejurikar and Gupta [1] introduced a heuristic search method to find the so called *critical speed* to balance the energy consumption between the processor and peripheral devices. With critical speed as the threshold for scaling, more researches [3], [6] were conducted to reduce the system-wide energy. However, the critical speed may require the processor to run at $higher-than-necessary$ speeds to execute a given set of real-time tasks, which can result in a large number of idle intervals. To effectively reduce the energy consumption during these idle intervals, more advanced techniques [6] based on procrastination of jobs have been proposed to merge/prolong the idle intervals and then shut down the processor/devices so long as the predicted idle interval length is larger than the so called *break even time* [3]. Zhuo *et al.* [2] considered controlling the inter-task preemptions in order to reduce the active period of the devices and therefore their energy consumption. But they assumed the energy and time overheads of mode transitions of the devices are negligible. In [6], Niu *et al.* proposed preemption control techniques for real-time systems with non-negligible shutdown overheads for peripheral devices. In [7], an off-line approach is proposed to compute the task-level *energy efficient speed* which was then used as the threshold speed for online slack reclaiming. In [3], an online approach was proposed to reduce the system-wide energy consumption with the system energy-efficient processor speed as the lower bound for speed scaling. In [4], more advanced techniques were provided to minimize the system level energy consumption by exploring the interplay between DVS and DPD for real time systems. Their approach targets the so called "frame-based" system which is a special case of real-time applications that all tasks have the same deadlines and periods.

In this paper, we study the problem of reducing the system-level energy consumption for hard real-time systems. We first show how to leverage the *critical speed* strategy and the traditional DVS strategy with the dynamic determination of the *break even time* to minimize the system energy consumption. Then based on it, we present a dynamic scheduling algorithm that combines *critical speed* strategy and DVS strategy adaptively to achieve higher efficiency in energy savings. The novelty and effectiveness of this approach are demonstrated with extensive simulation studies.

The rest of the paper is organized as follows. Section II introduces the system models and some preliminaries. Section III presents our dynamic algorithm to reduce the system energy. In section IV, we presents our experimental results. Section V draws the conclusions.

II. Preliminaries

A. System Models

The real-time system that we are interested in consists of N independent periodic tasks, $\mathcal{T} = \{\tau_1, \tau_2, \cdots, \tau_N\}$, scheduled according to the earliest deadline first (EDF) scheme. Each task, $\tau_i = (C_i, D_i, P_i)$, is characterized by its worst case execution

978-3-9810801-7-9/DATE11/©2011EDAA

time C_i, deadline D_i, and period P_i. We assume $D_i \leq P_i$. Each periodic task consists of a sequence of instances, called jobs. The j^{th} job of task τ_i is represented with $J_{ij} = (r_{ij}, c_{ij}, d_{ij})$, where r_{ij}, c_{ij}, and d_{ij} are the arrival time, actual execution cycles, and absolute deadline, respectively.

The system architecture consists of a core DVS processor and n devices, $M_0, M_1, ..., M_{n-1}$. The DVS processor used in our system can operate at a finite set of discrete supply voltage levels $\mathcal{V} = \{V_1, ..., V_{max}\}$, each with an associated speed s_i, which is normalized to the speed corresponding to V_{max}. The devices don't have DVS capability and can only support the DPD mechanism.

Each task τ_i is associated with a subset of peripheral devices $\Phi^i = \{M_0, M_1, ..., M_k\}$. Φ^i can be further divided into shared part Φ^i_{share} and non-shared-part $\Phi^i_{nonshare}$. Φ^i_{share} is the subset of the devices that can be accessed by τ_i and all other tasks, while $\Phi^i_{nonshare}$ is the subset of devices dedicated to τ_i. At any particular time, at most one task can access the same device.

Each peripheral device can be in one of the two states: *active state* and *shut-down state*. Energy overhead (E_o) and time overhead (t_o) need to be consumed to shut-down and later wake up the processor or devices. And the *break even time* (denoted as t_{be}) is widely used to reflect whether it is worthwhile to shut down the processor (or the device) during the idle interval or not. The general idea is: if we assume that the power consumption of the processor (or the device) in its idle state is P_{idle}, then the processor (or the device) can be shut down with positive energy gains only if the length of the idle interval is larger than the *break even time*, which is defined as $t_{be} = \max\{\frac{E_o}{P_{idle}}, t_o\}$.

Given the system architecture above, the power consumption when the system is active can be divided into two parts: the speed-dependent part $P_{dep}(s)$ and the speed-independent part P_{ind} [5]. And the critical speed ($s_{crit} = \sqrt[3]{\frac{P_{ind}}{2\alpha}}$ [5]) is widely used to balance the processor and device power and minimize the system active energy consumption. We call it *system level critical speed*. Note that, when the system is active at different time, the set of active devices in the system can be different. So the *system level critical speed* is not a static value and should be recomputed dynamically depends on the value of P_{ind} at run time. On the other hand, since the devices associated with each task is a fixed set in this paper, each task τ_i will also be associated with a static *task level critical speed*, represented as s^i_{crit}, which can be computed off-line in a similar way.

B. Motivations

Given the system models above, a popular approach in literature is to adopt the *critical speed* as the lower bound for speed reduction and shut down the processor or the device(s) if the predicted idle interval length is larger than the *break even time*(s) for the processor or the corresponding device(s). However, although seems reasonable, is this approach really energy beneficial?

Consider a job $J = (0, 9ms, 31ms)$ to be executed on a system with $P_{dep}(s) = 1.52s^3$ and $P_{ind} = 0.28Watt$. Such a system can be used to model a system consisting of an Intel XScale processor [8] and a peripheral device with active power of $P_{dact} = 0.2Watt$. (According to [8], the power consumption function for Intel XScale [8] can be modeled approximately as $P_{act}(s) = 1.52s^3 + 0.08$ Watt by treating 1GHz as the reference speed 1). And the derived critical speed s_{crit} normalized to the reference speed in such a system model is about 0.45 (at 452 MHz) with power consumption 0.42Watt. We assume the shut down overhead for the core processor to be $E^c_o = 0.8mJ$. If we assume the minimal processor speed is 0, the idle power for the processor P^c_{idle} will be 0.08 Watt and the break even time for the processor will be $t_{be} = 10ms$. Also for presentation convenience, we assume the shut down overhead for the device to be $E^d_o = 2mJ$ such that it has the same length of break even time as the core processor.

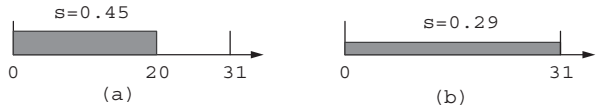

Fig. 1. (a) Job $J = (0, 9ms, 31ms)$ scheduled with critical speed; (b) The same job scheduled with DVS.

The job execution time after it is scaled to s_{crit} is 20ms and the corresponding schedule is shown in Figure 1(a). Since the job finishes at $f_1 = 20ms$, the idle period between its completion time and deadline is $11ms (> t_{be})$. So the processor and the device can be shut down between f_1 and its deadline. The energy consumption under this schedule is 11.17 mJoule. However, if we apply the traditional DVS strategy between its arrival time and deadline, as shown by the schedule in Figure 1(b). The scaled speed for job J will be 0.29. The energy consumption under the new schedule in Figure 1(b) is 9.8 mJoule, which is 12% lower than the previous case.

The reason why the *critical speed* with *shut down* strategy consumes more energy than *DVS* strategy mainly comes from two aspects: first, *critical speed* can only optimize the active energy to execute the job; second, the break even time t_{be} computed above is completely independent of the job execution situation and thus cannot reflect the energy consumed by executing the job at different speed (which will affect the idle interval length and shut-down decision correspondingly) at all. Obviously, in determining the energy beneficial break even time, the job speed scaling and device idle/shut-down should not be separated from each other and need to be considered simultaneously.

C. Compute the energy beneficial break even idle interval I_{eb}

Given the information above, the problem becomes how to determine the system-wide energy beneficial break even time. In this part, we will try to solve this problem analytically, based on the concept of *feasible interval* defined as followed:

Definition 1: Given a job $J_i = (r_i, c_i, d_i)$, the **feasible interval** of job J_i is a close interval $[t_s, t_e]$ during which J_i can be executed continuously without violating its deadline. And without loss of generality, we assume $t_s \geq r_i$ and $t_e \leq d_i$.

With the definition of *feasible interval*, the energy minimization problem for job J_i can be formulated as follows:

Problem 1: Given a job $J_i = (r_i, c_i, d_i)$ to be scheduled continuously in a feasible interval $[t_s, t_e]$, choose the most energy efficient speed for J_i such that the total energy consumption within the interval is minimized.

Obviously if the workload c_i of job J_i is larger than $s_{crit} \times (t_e - t_s)$, the feasible interval will be fully utilized and

the lowest feasible speed is the unique optimal speed. So we only need to make decision when $c_i < s_{crit} \times (t_e - t_s)$. In the latter case, according to [5], the most energy efficient speed for executing job J_i should be chosen between the *critical speed* and the lowest DVS speed . However, between the *critical speed* strategy and DVS strategy, which one is better will depend on the length of the idle interval following the completion of job J_i. In order to help solve the problem, we have the following definition.

Definition 2: Given a job $J_i = (r_i, c_i, d_i)$ to be executed with critical speed continuously, the *energy beneficial idle interval* (denoted as I_{eb}) is defined to be the minimal length of idle interval following the completion of job J_i such that the energy consumed by the *critical speed* with *shutdown* strategy is less than the DVS strategy.

By realizing the optimality of the critical speed s_{crit} in minimizing the active power, I_{eb} can be understood and computed as follows: Since the job active energy is minimized when it is executed with critical speed s_{crit}, each time when we scale the speed of job J_i to $s_i' < s_{crit}$, there will be an increment in its active energy, *i.e.*, $\Delta E = E(s_i') - E(s_{crit})$, where $E(s_i')$ is the active energy of job J_i under the new speed s_i' and $E(s_{crit})$ is the active energy under its critical speed. Since the function of active energy is convex with regard to speed s, ΔE is monotonically increasing with the reduction of s_i'. Since the *critical speed* with shut-down strategy will incur an energy overhead E_o, when $\Delta E < E_o$, DVS strategy will outperform the *critical speed* strategy. However, whenever ΔE reaches/surpasses E_o with the scaling of s_i', *critical speed* strategy will begin to outperform DVS strategy. And the difference between the completion times of J_i under these two strategies at the break even point will be I_{eb}.

Based on the concept of *energy beneficial idle interval*, in next section we will present our dynamic scheduling algorithm to minimize the system-wide energy consumption.

III. The dynamic scheduling algorithm

Our algorithm consists of two phases. During the off-line phase, the speed for each task τ_i is scaled as low as s^i_{crit} at the task level first. Then during the online phase, we dynamically compute the energy beneficial break even time I_{eb} for each job according to the run-time conditions and shut down the processor/devices when necessary. Before introducing our approach in detail, we first introduce the following theorem (the proof is omitted due to page limit):

Theorem 1: Given task set $\mathcal{T} = \{\tau_1, \tau_2, ..., \tau_N\}$ with tasks ordered by increasing value of D_i, all task deadlines can be guaranteed if any job J_i of task τ_i is procrastinated by no more than Δ_i time units, where Δ_i (called the *procrastination time* of task τ_i) is computed by

$$\Delta_i = \min\{t - \sum_{j \le i} \lfloor \frac{t + T_j - D_j}{T_j} \rfloor C_j,\ t \in \mathcal{S}\} \tag{1}$$

where $\mathcal{S} = \bigcup_{j=1}^{i} \mathcal{S}_j$, $\mathcal{S}_j = \{pT_j + D_j : p \in Z, D_i \le t \le \lfloor \frac{L_i}{T_j} \rfloor T_j + D_j\}$, and L_i is the length of the first deadline-based $level-i$ busy period, *i.e.*, the busy period in which only instances of tasks with relative deadlines less than or equal to D_i execute.

Based on Theorem 1, we have the following corollaries:

Corollary 1: Let $\mathcal{T}$ be the task set scheduled according to EDF and $up(J_i)$ be the upcoming job set in which each job J_k has arrival time $r_k > r_i$. If J_i is the only job in the ready queue, all jobs in $\mathcal{T}$ can meet their deadlines if the starting execution time of $up(J_i)$ is delayed to t_{LS}, where

$$t_{LS} = \min_{J_k \in up(J_i)} (r_k + \Delta_k) \tag{2}$$

Corollary 2: Let $\mathcal{T}$ be the task set scheduled according to EDF. Assume that J_i is the current job to be executed at any time t and $hp(J_i)$ is the upcoming job set in which each job J_l has arrival time $r_l > t$ and deadline $d_l < d_i$. Then all jobs in $\mathcal{T}$ can meet their deadlines if the starting execution time of $hp(J_i)$ is delayed to $\widetilde{t}_{LS}$, where

$$\widetilde{t}_{LS} = \min_{J_l \in hp(J_i)} (r_l + \Delta_l) \tag{3}$$

Algorithm 1 The online phase of scheduling algorithm.

1: **Input:** The current job J_i and the current time t_{cur};
2: **if** J_i is the only job in the ready queue **then**
3: Let $ed_i = \min\{d_i, t_{LS}\}$; // t_{LS} is computed with equation (2)
4: Compute I^i_{eb} based on J_i's current feasible interval $[t_{cur}, ed_i]$;
5: let f_i be the expected completion time of J_i under s_{crit};
6: **if** $(ed_i - f_i) \ge I^i_{eb}$ **then**
7: Execute J_i with s_{crit} and shut down the processor and devices in Φ_i at f_i and set up the wake up timer to be $(t_{LS} - f_i)$;
8: // critical speed strategy with shut-down
9: **else**
10: Let t_{na} be the earliest arrival time for the upcoming jobs;
11: **if** $f_i \ge t_{na}$ and $f_i \le d_i$ **then**
12: Execute J_i with s_{crit}; // critical speed strategy without shutting down shared devices in Φ^i_{share}
13: **else**
14: Execute J_i with $s_i' = \frac{c_i \times s_i}{ed_i - t_{cur}}$ non-preemptively within $[t_{cur}, ed_i]$; // DVS strategy stretching to ed_i;
15: **end if**
16: **end if**
17: **else**
18: Compute $\widetilde{t}_{LS}$ based on equation (3);
19: Execute J_i with $\max\{s_i, s_{crit}\}$ non-preemptively within $[t_{cur}, \widetilde{t}_{LS}]$ and shut down its non-shared devices $\Phi^i_{nonshare}$ upon its completion;
20: **end if**

With Corollary 1 and 2, we are now ready to formulate our dynamic scheduling algorithm. As shown in Algorithm 1, whenever the current job J_i is the only job in the ready queue, according to Corollary 1, the upcoming jobs can be procrastinated up to t_{LS}. So the speed of J_i can be reduced safely so long as it finishes no later than the $ed_i (= \min\{d_i, t_{LS}\})$. Moreover, it is possible that the system will become idle upon completion of J_i. In this scenario, whether we should adopt the *critical speed* strategy or the DVS strategy will depend on the expected completion time f_i of J_i under the critical speed s_{crit} (f_i can be computed by $f_i = \frac{c_i \times s_i}{s_{crit}} + t_{cur}$). If $(ed_i - f_i) \ge I^i_{eb}$, it is energy beneficial to shut down the processor and devices between $[f_i, t_{LS}]$ if we procrastinate the upcoming jobs to t_{LS} (line 7). Otherwise we should further check if f_i is less than d_i

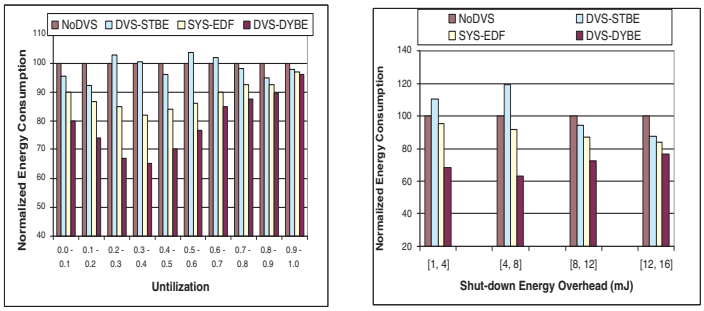

Fig. 2. (a) The energy comparison for randomly generated task sets; (b) The energy comparison for different shut-down overheads.

and larger than t_{na}, the earliest arrival time for the upcoming jobs. If it is, since the processor can continue to execute other jobs at t_{na} and there is no overhead for shutting down the processor and shared devices, the *critical speed* is still a good choice (line 12). Otherwise, there might be idle interval generated between $[f_i, t_{na}]$ and it is not beneficial to shut down the processor and devices between $[f_i, ed_i]$. In this scenario, the DVS strategy should be adopted between $[t_{cur}, ed_i]$ (line 14).

When there are more than one jobs in the ready queue, we adopt the preemption control technique based on Corollary 2 to execute J_i non-preemptively with the maximum between its prescaled speed s_i and s_{crit} and shut down its non-shared devices in $\Phi^i_{nonshare}$ upon its completion (line 17-19).

IV. Experimental Results

Two groups of experiments were conducted. In the first group of experiments, we randomly generated periodic task sets with five tasks. The periods were randomly chosen in the range of $[10,\ 100]ms$. The worst case execution time (WCET) of a task was set to be uniformly distributed from $1ms$ to its deadline and the actual execution time of a job was randomly picked from [0.2WCET, WCET]. For the processor model we will adopt the Intel XScale processor [8]. The devices were randomly chosen from five types of devices [3]: $M^1 = (0.187, 1.25)$, $M^2 = (0.75, 4)$, $M^3 = (1.3, 6)$, $M^4 = (0.225, 0.2)$, and $M^5 = (0.075, 0)$, where each device type is characterized by two parameters (P^i_{dact}, E^i_o), representing its active power (in Watt) and shut-down/start-up energy overhead (in mJoule).

Four different approaches were compared. The first approach *NoDVS* executed all tasks with the highest speed and we used its results as the reference results. The second approach $DVS-STBE$ is the approach in [6] which adopts static shut-down threshold idle time for each device and uses task level critical speed as the threshold for scaling. The third approach $SYS-EDF$ is the approach in [3] which adopts static break even time for each device and system energy-efficient processor speed as the threshold for scaling. The fourth approach $DVS-DYBE$ is our approach illustrated in Section III. The results are shown in Figure 2(a). We can see from the results that both $DVS-STBE$ and $SYS-EDF$ have much higher energy consumption than $DVS-DYBE$ and in some utilization intervals $DVS-STBE$ can have even higher energy consumption than *NoDVS*. This is because, since both $DVS-STBE$ and $SYS-EDF$ adopt static break even time and consider device shut-down and speed reduction separately, after speed reduction, some devices that can originally be shut-down will have to stay in active mode due to shortened idle interval length. At the same time, both of them adopt the critical speed or the energy-efficient speed as the lower bound for scaling, which can only reduce the active energy and might not minimize the system energy. Compared with the other approaches, our algorithm $DVS-DYBE$ adopting dynamic break even time and determining job speed by considering device shut-down and task procrastination simultaneously can reduce the overall energy significantly in most utilization intervals. The maximal reduction by $DVS-DYBE$ over $DVS-STBE$ and $SYS-EDF$ can be nearly 35% and 20% respectively.

In the second group of experiments, we varied the device shut-down/wake-up overheads by randomly selecting E^i_o from one of four ranges [1,4]mJ, [4,8]mJ, [8,12]mJ, and [12,16]mJ, respectively. The results for task sets within representative intervals, *i.e.*, [0.2, 0.5] are shown in Figure 2(b).

As shown in Figure 2(b), when the shut-down energy overheads are chosen from relatively smaller ranges, the energy consumption of $DVS-STBE$ and $SYS-EDF$ can be higher than or close to that by *NoDVS* for the same reasons as stated above. When energy overheads become larger, the energy performance of $DVS-STBE$ and $SYS-EDF$ become better than *NoDVS* because all of them adopt static break even time and in this situation all of them cannot shut down the devices in most idle intervals and the preemption control and device scheduling in $DVS-STBE$ and $SYS-EDF$ begin to take effect. Again, our approach $DVS-DYBE$ outperformed the other approaches significantly for all overhead ranges. The maximal energy reduction by $DVS-DYBE$ over $DVS-STBE$ and $SYS-EDF$ can be nearly 38% and 23%, respectively.

V. Summary

In this paper, we present a dynamic scheduling algorithm to minimize the system level energy consumption for hard real-time systems. Different from previous work that adopted the statically defined break even idle time, we propose an approach to enhance the computation of the energy beneficial break even idle time dynamically. Then based on it a dynamic scheduling approach is proposed in the management of speed determination and task preemption to reduce the energy consumption. Through extensive experiments, our approach can reduce the energy consumption for the processor/devices significantly when compared with the existing research.

References

[1] R. Jejurikar and R. Gupta, "Dynamic voltage scaling for system-wide energy minimization in real-time embedded systems," *ISLPED*, 2004.

[2] J. Zhuo and C. Chakrabarti, "System level energy efficient dynamic task scheduling," *DAC*, 2005.

[3] H. Cheng and S. Goddard, "Sys-edf: A system-wide energy-efficient scheduling algorithm for hard real-time systems," *International Journal of Embedded System: Special Issue on Low Power Embedded Computing*, vol. 4, no. 2, pp. 141–151, 2009.

[4] B. Zhao and H. Aydin, "Minimizing expected energy consumption through optimal integration of dvs and dpm," in *ICCAD*, 2009.

[5] L. Niu, "Rate-monotonic scheduling for reducing system-wide energy consumption for hard real-time systems," in *ICCD*, 2010.

[6] L. Niu and G. Quan, "Peripheral-conscious scheduling on energy minimization for weakly hard real-time systems," *DATE*, 2007.

[7] H. Aydin, V. Devadas, and D. Zhu, "System-level energy management for periodic real-time tasks," in *RTSS*, 2006.

[8] INTEL-XSCALE. (2003). [Online]. Available: http://developer.intel.com/design/xscale/.