On the Interplay of Dynamic Voltage Scaling and Dynamic Power Management in Real-Time Embedded Applications*

Vinay Devadas, Hakan Aydin
Dept. of Computer Science, George Mason University
Fairfax, VA, USA
{vdevadas,aydin}@cs.gmu.edu

ABSTRACT

Dynamic Voltage Scaling (DVS) and Dynamic Power Management (DPM) are two popular techniques commonly employed to save energy in real-time embedded systems. DVS policies aim at reducing the CPU energy, while DPM-based solutions involve putting the system components (e.g. memory or I/O devices) to low-power/sleep states at run-time, when sufficiently long idle intervals can be predicted. Despite numerous research papers that tackled the energy minimization problem using DVS or DPM separately, the interactions of these two popular techniques are not yet well understood. In this paper, we undertake an exact analysis of the problem for a real-time embedded application running on a DVS-enabled CPU and using potentially multiple devices. Specifically, by adopting a generalized system-level energy model and taking into account the non-trivial time/energy overheads involved in device transitions, we characterize the variations in different components of the system energy as a function of the CPU processing speed. Then, we propose a provably optimal algorithm to determine the optimal CPU speed as well as device state transition decisions to minimize the system-level energy. Our algorithm runs in $O(m \log m)$ time, where m is the number of devices used by the application. The evaluations with realistic system parameters indicate that our solution, which combines DVS and DPM optimally, can lead to substantial energy savings when compared to previous solutions.

Categories and Subject Descriptors

D.4.1 [**Operating Systems**]: Process Management—*Scheduling*; D.4.7 [**Operating Systems**]: Organization and Design—*Real time systems and embedded systems*

General Terms

Algorithms, Performance

Keywords

Real-Time Systems, Energy Management, Dynamic Voltage Scaling, Dynamic Power Management

*This work is supported by US National Science Foundation grants CNS-072047 and CNS-546244 (CAREER Award)

EMSOFT'08, October 19–24, 2008, Atlanta, Georgia, USA.

1. INTRODUCTION

Many embedded devices are usually battery-operated and hence, have limited energy supply. Due to the growing demand for smaller devices with longer battery life, energy management has become one of the major goals in embedded systems research. Over the past decade, the research community has made significant progress in the area of low-power system design [7, 10]. On the industry side, the Advanced Configuration and Power Interface (ACPI) standard, which was introduced in 1997, has moved power management to the operating system level by providing system calls for predictive shutdown of system components. Many applications running on power-limited systems (such as embedded controllers) are subject to timing constraints. As a result, the real-time and energy-aware operation is a highly desirable and sometimes critical feature of an embedded system.

Dynamic Voltage Scaling (DVS) [19] is a popular and widely-used technique for power management in real-time embedded systems. With DVS, the processor can operate at different frequency (speed) levels. Since the CPU power consumption increases in convex fashion with the frequency, DVS helps to significantly reduce the CPU dynamic energy consumption. In real-time systems, preserving the temporal correctness (the system *feasibility*) is of paramount importance. Hence, in DVS settings, utmost care must be exercised to avoid deadline misses. The problem of minimizing the energy consumption while satisfying the timing constraints is known as the *Real-Time DVS (RT-DVS)* problem. The RT-DVS problem has been extensively studied in recent past for various task/system models [2, 11, 12, 14].

Dynamic Power Management (DPM) is another commonly used energy management technique, aiming at reducing off-chip device energy consumption. Typical off-chip devices have an active state in which they process requests and at least one low-power sleep state. DPM deals with transitioning devices to low-power states when not in use so as to reduce the device energy consumption. Memory modules and I/O devices which consume significant energy have been the primary targets of DPM. However, non-trivial time and energy overheads are associated with each device state transition. As a consequence, transitioning devices to low-power states is energy-efficient only when the device idle interval is guaranteed to be greater than a certain threshold (typically called the *device break-even time*).

One of the primary difficulties associated with the use of DPM is to decide when to switch a device to a low-power state. DPM techniques can be classified as stochastic, predictive and timeout-based [3]. In real-time systems, the pre-

dictive DPM techniques are commonly used. The predictive techniques involve making accurate predictions about the *next usage time* of idle devices. As such, predicting the next device usage time is of critical importance in real-time systems. Under-estimations may lead to inefficient energy management (as devices would not be put to low-power states) and over-predictions may lead to potential deadline misses (due to the non-trivial transition delays). Several offline and online solutions have been proposed for real-time DPM under different task/device settings [5, 6, 18, 16, 17].

While DVS and DPM are popular techniques targeting energy minimization in CPU and external devices respectively, a comprehensive system-level energy management policy is likely to use *both* DVS and DPM. However, integrating DVS and DPM in a single framework poses several challenges. With DVS, low processor frequencies lead to low CPU dynamic energy consumption figures. However, this also results in elongated task execution times and shortened device idle intervals. This not only forces devices to remain in active state for longer periods but also limits the possibility of transitioning devices to sleep states (as shortened idle intervals will tend to be smaller than the device break-even times). On the other hand, running the processor at higher frequencies reduces the device energy and creates more opportunities for transitioning devices to sleep states, *at the cost of increased CPU energy and transition energy*. Thus, there is an intriguing trade-off spectrum covering the benefit/cost spaces of DVS and DPM techniques.

Recently, there have been a few research efforts addressing system-wide energy consumption issues for real-time embedded systems. In [8], the concept of *critical speed* (or, *energy-efficient speed*) was introduced. This stems from the observation that lowering the processor speed below a certain threshold can have negative effects on the system-wide energy consumption. The energy-efficient speed is calculated by considering both the device energy and CPU energy consumed during task executions. Each task can potentially have a unique energy-efficient speed, depending on the devices it uses during its execution. In [8], the authors provide a single policy to manage both processor leakage energy and device energy. In [24], the authors propose a dynamic task scheduling algorithm using the concept of critical speed which minimizes the system-wide energy consumption.

In [1], the authors consider a generalized power model taking into account several factors such as on-chip/off-chip workload ratios, effective switching capacitance and frequency-dependent and frequency-independent power components. For this generalized power model, the authors show how to derive the task-level energy-efficient speeds and propose an $O(n^3)$ algorithm to optimally solve the system-wide energy minimization problem for periodic hard real-time tasks. In [22], the authors address the energy minimization problem assuming a DVS processor with limited number of speed values. Recently, other research groups addressed the CPU and memory energy minimization problems [15, 23].

While the concept of critical speed helps mitigate the negative impacts of DVS on the system-wide energy, one major drawback of most system-level real-time energy management research efforts is that they either assume negligible device transition overheads or provide no DPM policies. In fact, most of these studies often make the assumption that the device will be transitioned to the low-power state whenever it is not in use. However, this is not the case due to the non-trivial time/energy device transition overheads. A recent research effort that combines both DVS and DPM in the same framework is given in [4], where the authors propose a practical system-level energy management heuristic called SYS-EDF for periodic real-time tasks and discrete model DVS capable processor. SYS-EDF is a combination of a DPM policy Conservative Energy-EfficientDevice Scheduling (CEEDS) and a DVS scheme based on energy-efficientscaling.

Despite all these efforts, **an extremely important but mostly unexplored issue is the analysis of exact interplay of DVS and DPM in real-time embedded systems**. Obviously, a straightforward integration of a DVS scheme using critical speed and a DPM heuristic (such as SYS-EDF) does not exploit this interplay fully. We contend that it is imperative to formally characterize this interplay to devise optimal energy management systems in the presence of both DVS and DPM features.

Contributions of this research effort:

- In this paper, we address the problem of exact characterization of *system-level* energy consumption on a real-time embedded environment with *both DVS and DPM* features. This characterization allows us to perform a *formal* analysis of the interplay between DVS and DPM for a real-time application that uses several devices. By using the results from this analysis, we propose a $O(m \log m)$-time algorithm (where m is the number of devices used by the application) to determine the optimal processor speed and device transitioning decisions to minimize the system-wide energy consumption. To the best of our knowledge, this is the first research effort to not only investigate the exact interplay between DVS and DPM, but to also provide a *provably optimal* solution to the system-level energy management problem *by taking into account DVS/DPM-related issues and device transition overheads*.
- We show that our optimal scheme, designed by considering the DVS/DPM interplay and device transition overheads, yields significant energy gains when compared to the existing sub-optimal approaches. Our experimental results are obtained using real device and CPU specifications, as well as real energy consumption figures.

The remaining of the paper is organized as follows. In Section 2, we give the system model and assumptions. In Section 3, we illustrate the non-trivial challenges in the analysis of the interplay between DVS and DPM, by focusing on the simple case where the real-time application uses only one device. In Section 4, we solve the general case of the problem for multiple devices. We conclude in Section 5.

2. SYSTEM MODEL AND ASSUMPTIONS

2.1 Application Model

We consider a real-time embedded application that is invoked periodically with a period of d, at time instants $k \cdot d$ ($k \in Z$). At each invocation, the application must complete its execution within the time interval $[k \cdot d, (k+1) \cdot d]$, which is referred to as a *frame*. This embedded application model is also known as a *frame-based system* in the literature [13, 20].

The system is equipped with a DVS-enabled CPU where the processing frequency f can be adjusted up to a maximum frequency f_{max}. We normalize the frequency values with respect to f_{max}. The worst-case execution time (WCET) of the application under maximum frequency f_{max} is denoted by c. We assume that the application's execution time scales linearly with processor frequency; that is, at frequency f the WCET of the application is $WCET(f) = \frac{c}{f}$.

The base *utilization* of the real-time application is denoted by $U = \frac{c}{d} \leq 1$. The *slack* refers to the unused CPU time between the completion time of the application and the beginning of the next frame, at each invocation. Formally, the slack of the application at frequency f is given by $\delta(f) = d - \frac{c}{f}$.

2.2 Device Model

The real-time embedded application is assumed to use a set of m devices denoted by $\mathcal{D} = \{D_1 \ldots D_m\}$ during its execution. Each device is assumed to have (at least) two states: an active state and a sleep (low-power) state. Following [5, 6, 18, 16, 17] we assume *inter-task device scheduling*. Under *inter-task device scheduling* approach, all devices needed by the real-time application must be in active state at the beginning of each frame and they should remain in active state until the application completes its execution in that frame. A device can be put to sleep state when the application completes its execution (i.e. during the *slack* period). These assumptions are realistic given that the device state transitions typically involve non-trivial costs and it is fairly difficult to predict when a running application will re-request a specific device during *execution* [5, 6, 18, 16, 17].

The following parameters are associated with each device D_i:

- P_a^i: The device power consumption in active state.
- P_s^i: The device power consumption in sleep state.
- T_{sd}^i and T_{wu}^i: The device state transition times (from active to sleep and from sleep to active, respectively).
- E_{sd}^i and E_{wu}^i: The corresponding device transition energy overheads.

Given that devices are associated with non-zero transition costs, the device *break-even time* B_i denotes the minimum length of idle period which justifies a device transition from active to sleep state. Let $T_{sw}^i = T_{sd}^i + T_{wu}^i$. We denote by B_{actual}^i the minimum idle interval length during which keeping D_i in active state consumes the same amount of energy as transitioning D_i from active to sleep and back from sleep to active. Thus, $B_{actual}^i = \frac{E_{sd}^i + E_{wu}^i - T_{sw}^i \cdot P_s^i}{P_a^i - P_s^i}$. In other words, B_{actual}^i characterizes the minimum idle interval length for *energy-efficient device state transitions*. Further, the device idle interval should be long enough to allow at least two device transitions: one from active to sleep and another from sleep to active, implying that device break-even times cannot be shorter than T_{sw}^i. Hence, the device break-even time B_i is given as [4, 5]:

$$B_i = max(B_{actual}^i, T_{sw}^i)$$
$$= max(\frac{E_{sd}^i + E_{wu}^i - T_{sw}^i \cdot P_s^i}{P_a^i - P_s^i}, T_{sd}^i + T_{wu}^i)$$

Observe that the devices become idle at the end of task execution and remain so until the beginning of the next frame. In each frame, a device D_i can be transitioned to sleep state at the end of task execution, only if $\delta(f) \geq B_i$, where f is the processor frequency at which the application is executed. Further, if the slack at the maximum frequency $\delta(f_{max})$ is smaller than B_i, then D_i will be forced to remain in active state throughout the frame (since lower frequencies can only reduce its slack time). Since this work explores the combined effects of DPM and DVS, we will assume that $\delta(f_{max}) > B_i$, for all devices[1].

2.3 Energy Model

Since the embedded application is invoked in periodic fashion, we concentrate on the energy consumption over a single frame. The system energy E can be divided into static energy (E_s) and dynamic energy ($E(f)$):

$$E = E_s + E(f)$$

The static energy, E_s, is due to the static power which is required for purposes such as keeping the system clock running, maintaining the basic circuits and keeping the devices in sleep states. Since the static power can only be eliminated by completely turning off the entire system, we assume that the static energy is not manageable. As such, we focus on minimizing the dynamic energy consumption $E(f)$ which is a function of the processor frequency and includes the system components such as CPU, the main memory and I/O devices.

At the end of task execution in each frame, depending on the system slack, the devices can either be transitioned to sleep state or kept in active state. Let $\mathcal{D}_A$ denote the set of devices kept in active state throughout the frame. Devices in $\mathcal{D} - \mathcal{D}_A$ are transitioned to sleep state. It is assumed that each device $D_j \in (\mathcal{D} - \mathcal{D}_A)$ is re-activated T_{wu}^j time units before the start of the next frame, to allow timely execution. Also, let $P_{ind} = \sum_{i|D_i \in \mathcal{D}} P_a^i$ denote the total active power of the application's devices. Given this notation, the dynamic system energy consumption at frequency f over the duration of a frame is given as:

$$E(f) = (af^3 + P_{ind})\frac{c}{f} + \sum_{i|D_i \in \mathcal{D}_A} P_a^i \cdot \delta(f) + \sum_{i|D_i \in (\mathcal{D} - \mathcal{D}_A)} (E_{sd}^i + E_{wu}^i)$$

The processor power consumption is modeled as a convex function af^3, where 'a' is the switching capacitance. At frequency f, the application executes for $\frac{c}{f}$ units, during which the processor consumes $af^3 . \frac{c}{f}$ energy. $P_{ind} \frac{c}{f}$ represents the total device energy consumption during the execution of the application. $\sum_{i|D_i \in \mathcal{D}_A} P_a^i \cdot \delta(f)$ represents the total energy consumed over the slack period by devices remaining in active states. $\sum_{i|D_i \in \mathcal{D} - \mathcal{D}A} (E_{sd}^i + E_{wu}^i)$ represents the transition energy overhead for devices that are transitioned to sleep state during the slack period. We note that this component of the system energy representing device transition overheads was not considered in previous system-level energy management papers [1, 8, 24].

[1]If this condition is not satisfied for a given device D_i, then D_i cannot be managed and its energy consumption can be considered as part of the static energy. The framework of the paper is still applicable to the remaining devices.

3. SINGLE-DEVICE MODEL

In this section, we consider a simplified model where the real-time application uses a single device. Using this simple model, we illustrate several non-trivial observations that lead us to the characterization of the exact interplay between DVS and DPM. We also provide an $O(1)$ algorithm to calculate the speed that optimizes system-wide energy while taking into account the DVS/DPM interplay and device transition overheads. In Section 4, we will extend these results to the general case with multiple devices.

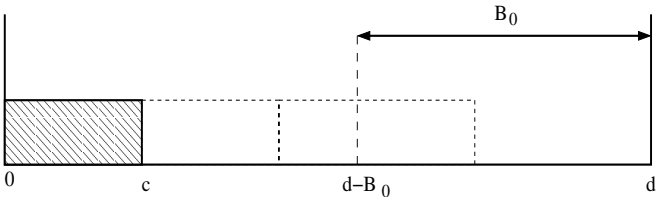


Figure 1: The break-even time and the impact of DVS

The exact characterization of the trade-offs between DVS and DPM is critical for system-wide energy minimization. Consider a real-time application with WCET of c units and frame length of d units, using a device D_0 with break-even time B_0. By adjusting the processor frequency, the completion time of the task can be anywhere from c to d (Figure 1). This frequency assignment has obviously serious consequences for the applicability of DPM, and hence for overall system energy. Let us denote the frequency which produces a slack of exactly B_0 units by $f^* = \frac{c}{d-B_0}$. Note that to transition D_0, the processor has to run the real-time frame-based application at a frequency no less than f^*.

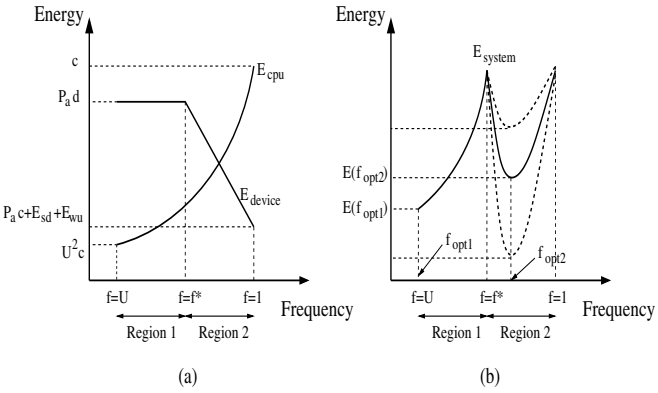


Figure 2: System Energy Consumption as a function of Processor Frequency

Figure 2(a) shows the variations in device energy consumption (E_{device}) and CPU energy consumption (E_{cpu}) as a function of the processor frequency.[2]Note that E_{device} also includes device transition costs, when applicable. To start with, E_{cpu} increases with increasing frequency in a quadratic manner. However, E_{device} follows different patterns in two different regions. In Region 1 where $U \leq f < f^*$, the device D_0 cannot be transitioned (because the slack is smaller than B_0) and it is forced to remain in active state throughout the frame. As such, $E_{device} = P_a\, d$ is constant in Region 1. In Region 2 where $f^* \leq f \leq f_{max}$, the device can be transitioned to sleep state. Further, as the frequency increases beyond f^* in Region 2, the slack and hence, the length of the device sleep interval, increases. Thus, in Region 2, E_{device} decreases with increasing frequency.

Figure 2(b) shows the variation of the system energy consumption, $E_{system} = E_{device} + E_{cpu}$, as a function of the frequency. We can see that E_{system} exhibits varying trends in Regions 1 and 2. While E_{system} increases in Region 1 with increasing frequency, the local minimal of E_{system} in Region 2 can lie anywhere in the range $[f^*, f_{max}]$.

It is worthwhile to compare these trends to the results of prior energy management studies. Region 1 is the spectrum where the dynamic CPU power dominates. In fact, this was precisely the assumption of the early *RT-DVS* papers [2, 11], which effectively ignored Region 2. Consequently, in Region 1, the minimum frequency that guarantees system feasibility ($f = U$) is optimal. Region 2 is somewhat similar to the spectrum assumed by the recent system-level energy management papers [1, 8, 24], which considered the CPU and device energy figures at the same time. But, these papers neither accounted for energy transition overheads nor addressed the question of whether DPM is justifiable at runtime, given the length of idle intervals. As a result, these approaches ignored Region 1. We can see that one really needs to consider *both* regions to analyze (and get full benefits of) DVS and DPM, simultaneously.

The local optimal frequencies that minimizes E_{system} in Regions 1 and 2 are well defined. However, *there is no a priori reason why global optimal frequency that minimizes E_{system} should lie in Region 1 or Region 2.* Depending on the relative power consumption rates of the device and CPU, the execution time of the application and the relative positions of U and f^*, the global optimal may be in either Region 1 or Region 2. In fact, f_{opt2}, which optimizes $E_{cpu} + E_{device}$ *and* transitions the device to sleep state *(by incurring the transition energy)* may possibly consume more system-wide energy compared to the frequency f_{opt1}, which minimizes E_{cpu} and avoids device transition costs, without paying special attention to E_{device}. As such, *exact evaluation and comparison of the local optimal values in Regions 1 and 2 is necessary in order to determine the global optimal.*

3.1 System Energy Minimization in Region 2

While the local optimal in Region 1 is straightforward to find, the one in Region 2 requires some elaboration. In Region 2, all frequency values support energy-efficient device transitions and the device *will* be transitioned at the end of task execution. Thus, in Region 2 the system energy consumption can be expressed as:

$$\bar{E}(f) = (af^3 + P_a^0)\frac{c}{f} + (E_{sd}^0 + E_{wu}^0)$$

[2]For the purpose of presentation, Figure 2 is drawn assuming device break-even time $B = B_{actual}$. The formal analysis does not make such an assumption. Further, the device sleep power consumption P_s^i can only be eliminated by completely turning off the device. Consequently, we assume that the transition costs associated with completely turning off devices are significant and devices cannot be completely turned off. As a result, a device D_i will always consume power at the rate of at least P_s^i. In this and upcoming figures/discussions, without loss of generality, we assume $P_s^i = 0$, $P_a^i = P_a^i - P_s^i$, $E_{sd}^i = E_{sd}^i - (P_s^i \cdot T_{sd}^i)$ and $E_{wu}^i = E_{wu}^i - (P_s^i \cdot T_{wu}^i)$, for all devices. Thus, all power consumption rates for a particular device, D_i, are given in excess of P_s^i. Notice that such a transformation changes neither the original value of B_i nor the analysis in the upcoming sections.

Observe that $\bar{E}(f)$ is a strictly convex function. $(E^0_{sd} + E^0_{wu})$ appears as a constant in $\bar{E}(f)$ and hence, the frequency that minimizes $\bar{E}(f)$ can be found by setting its derivative to zero. This gives:

$$f_{ee} = (\frac{P^0_a}{2a})^{1/3}$$

This is, as expected, numerically equal to the *energy-efficient speed* value given in [1], that did not consider neither the transition energy, nor the DPM issues.

REMARK 1. *An energy-efficient device state transition as assumed by the operation in Region 2 may not be possible by using the frequency f_{ee}, if f_{ee} lies outside the range $[f^*, f_{max}]$.*

REMARK 2. *Even when f_{ee} is in the range $[f^*, f_{max}]$, in order to find the global optimal frequency, one still needs to compare the minimum energy consumption in Region 2 which* incurs a transition overhead *against the minimum energy consumption in Region 1 which* does not incur a transition overhead. *This will be fully analyzed in Section 3.2.*

The convex nature of $\bar{E}(f)$ justifies the following two basic properties for any $\epsilon > 0$.

PROPERTY 1. $\forall f,\ f > f_{ee},\ \bar{E}(f_{ee}) \leq \bar{E}(f) \leq \bar{E}(f + \epsilon)$

PROPERTY 2. $\forall f,\ f < f_{ee},\ \bar{E}(f_{ee}) \leq \bar{E}(f) \leq \bar{E}(f - \epsilon)$

Let $\bar{f}$ denote the frequency that minimizes system energy in Region 2. We determine $\bar{f}$ by considering 3 possible cases.

- **Case 1:** $f^* \leq f_{ee} \leq f_{max}$
  In this case, D_0 can be transitioned to sleep state at $f = f_{ee}$ as $\delta(f_{ee}) \geq B_0$. Also there is no other frequency which can transition D_0 *and* yield better system energy consumption in Region 2. Thus, $\bar{f} = f_{ee}$.

- **Case 2:** $f_{ee} > f_{max}$
  From Property 2, the system energy in Region 2 is minimized when $f = f_{max}$. Thus, $\bar{f} = f_{max}$.

- **Case 3:** $f_{ee} < f^*$
  This implies $\delta(f_{ee}) < B_0$ and D_0 cannot be transitioned in energy-efficient fashion at $f = f_{ee}$. Thus, in an effort to transition the device we have to increase frequency beyond f_{ee} and towards f^*, which represents the first instance when the device can be transitioned. From Property 1, we can find that the system energy in Region 2 is minimized when $f = f^*$. Hence, $\bar{f} = f^*$.

Based on the analysis above, we can write:

$$\bar{f} = max(f^*, min(f_{ee}, f_{max}))$$

Note that, this formulation covers the three cases examined for energy minimization in Region 2 and also restricts $\bar{f}$ to the range $[f^*, f_{max}]$. Since $B_0 < d$, $f^* > U$ and it follows that $\bar{f} > U$. Thus, $\bar{f}$ preserves the system feasibility as well.

3.2 Finding the Global Optimal

In the previous discussions, we showed that $f = U$ and $f = \bar{f}$ are the local optimal values in Regions 1 and 2, respectively. However, there is no a priori reason for global optimal frequency that minimizes system energy across Regions 1 and 2 to lie in one region as opposed to the other. To determine the global optimal, it is essential to consider the local optimal values in both Regions 1 and 2 and compare them.

Recall that $\bar{f} \geq U$. Let $E_t = E^0_{sd} + E^0_{wu}$. By running the application at $f = \bar{f}$ as opposed to $f = U$, the CPU energy consumption increases by $\Delta E_{cpu} = ac\,(\bar{f}^2 - U^2)$ and the device energy consumption decreases by $\Delta E_{device} = cP^0_a(\frac{1}{U} - \frac{1}{\bar{f}}) - E_t$. Running the task at $f = \bar{f}$ and transitioning the device is better compared to running the task at $f = U$ and keeping the device active throughout the frame, *only if* the increase in CPU energy, ΔE_{cpu}, is overshadowed by the decrease in device energy, ΔE_{device}.

Given frequencies $\bar{f}$ and U where $0 \leq U \leq \bar{f} \leq f_{max}$, let $\phi(\bar{f}, U)$ denote the net change in system energy when the application is run at $f = \bar{f}$ as opposed to $f = U$. Hence:

$$\phi(\bar{f}, U) = \Delta E_{cpu} - \Delta E_{device} = ac(\bar{f}^2 - U^2) - P^0_a c(\frac{1}{U} - \frac{1}{\bar{f}}) + E_t$$

If $\phi(\bar{f}, U) < 0$ then $\Delta E_{cpu} < \Delta E_{device}$ and the energy is minimized at $f_{opt} = \bar{f}$. On the other hand, if $\phi(\bar{f}, U) > 0$ then $\Delta E_{cpu} > \Delta E_{device}$ and the energy is minimized at $f_{opt} = U$.

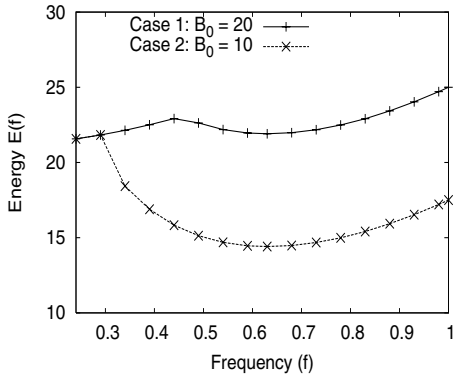


Figure 3: Example illustrating the position of global optimal.

Now, we illustrate this analysis through an example, which also shows that that the global optimal can lie in either Region 1 or Region 2 depending on the power characteristics of the devices and the CPU. Consider a real-time application with $c = 10$ and $d = 42$. Let D_0 have the parameters $P^0_a = 0.5$, $E^0_{sd} = E^0_{wu} = 5$ and $T^0_{sd} = T^0_{wu} = 10$. Recall that all power figures are given in excess of device sleep power. Assume switching capacitance $a = 1$. From the data, we can find that $B_0 = 20$ and $f_{ee} = 0.63$. Observe that $\delta(f_{ee}) > B_0$. Hence, it turns out that the device can be effectively put to sleep state and run at $f = f_{ee}$. However, we obtain $E(U = \frac{10}{42}) = 21.57$ and $E(f_{ee}) = 21.91$. *This shows that, for the given settings, despite the fact that the device can be transitioned at f_{ee}, from the system energy point of view it is better to keep D_0 in* active *state throughout the frame and run the application at $f = U$.* This is shown through Case 1 of Figure 3. In the same example, by changing $E^0_{sd} = E^0_{wu} = 1.25$ and $T^0_{sd} = T^0_{wu} = 5$, we get $B_0 = 10$. With these new parameters, one can verify that $E(f_{ee}) < E(U)$ as shown in Case 2 of Figure 3.

The $O(1)$ algorithm for computing the processor frequency that minimizes the system-wide energy for the single-device model is given below.

- Set $f_{ee} = (\frac{P_a^0}{2a})^{1/3}$
- Set $f^* = \frac{c}{d - B_0}$
- $\bar{f} = max(f^*, min(f_{ee}, f_{max}))$
- if $\phi(\bar{f}, U) < 0$ then $f_{opt} = \bar{f}$ else $f_{opt} = U$

Figure 4: Algorithm for Single-Device System.

A final observation is in order about the relative ordering of B^0_{actual} and T^0_{sw}, whose maximum was defined as the break-even time B_0. If $B_0 = B^0_{actual}$, since $\delta(f^*) = B_0 = B^0_{actual}$, the following inequality holds:

$$E(U) \leq E(f^*) \leq E(f^* + \epsilon)$$

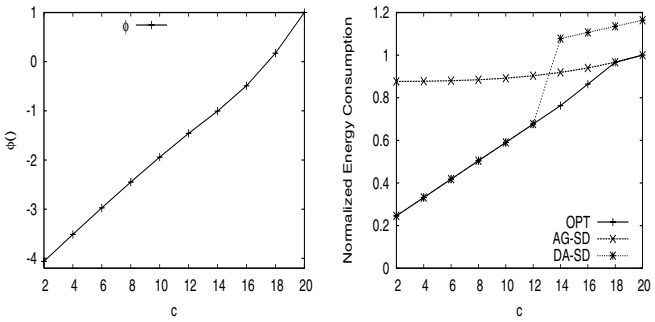
This implies that, if $\bar{f} = f^*$, we know for sure $f = U$ is the global optimal and a comparison between $E(U)$ and $E(\bar{f} = f^*)$ is not required. However, the above inequality may not hold when $B_0 = T^0_{sw} > B^0_{actual}$. In this case, nothing can be said about the relative ordering of $E(U)$ and $E(f^*)$, as illustrated by the following example.

Let $c = 5$, $d = 19$, $P^0_a = 0.25$, $T^0_{sd} = T^0_{wu} = 5$, $E^0_{sd} = E^0_{wu} = 0.625$. For the given data, $B_0 = T^0_{sw} = 10$ and $f^* = \frac{5}{9}$. Assume $a = 1$. We can verify $E(f^*) = 5.04 < E(U) = 5.096$. By changing $E^0_{sd} = E^0_{wu} = 1$ we can verify that B_0 and f^* still remain the same. However, in these new settings, $E(f^*) > E(U)$.

3.3 Experimental Evaluation

In this section, we perform an experimental evaluation using the actual device specifications taken from [5]. The CPU power consumption rate at the maximum processing frequency is modeled after Intel XScale [21]. We consider a real-time application with a frame length of $44ms$. The application is assumed to use IBM Microdrive during its execution. Based on the device characteristics of IBM Microdrive [5], its break-even time can be computed as $24ms$. Observe that if the worst-case execution time c of the real-time application at the maximum speed is greater than $20ms$, then the device can never be transitioned to sleep state as there is not enough slack to justify it. Thus, when $c > 20$, the problem of system-wide energy minimization reduces to minimizing CPU energy only and running the CPU at $f = U$ is optimal. Hence, we only vary c from 2-20ms in steps of $2ms$. For each distinct utilization value, we compare three schemes.

- *OPT*: Optimal scheme from Section 3.2.
- *Aggressive Slow-Down (AG-SD)*: Run the processor at the lowest frequency $f = \frac{c}{d} = U$ that can still meet the deadline. In this scheme, the devices are never transitioned to sleep state. The frequency $f = U$ minimizes the *CPU dynamic energy consumption only* [2, 11].
- *Device-Aware Slow-Down (DA-SD)*: This scheme is based on the concept of energy-efficient speed [1, 8, 24] and is adopted from [1], where the authors propose an optimal solution to the system-wide energy minimization problem *ignoring DPM issues and device transition overheads*. The energy-efficient speed (denoted by f_{ee}) is computed as the speed that minimizes the system energy, by considering only CPU energy and device active energy consumption. Since f_{ee} can be less than the system utilization, to preserve the feasibility, we execute the task at $f = max(U, f_{ee})$. If the device can be transitioned at $f = f_{ee}$ we do so; else the device remains in active state throughout the frame.



(a) $\phi()$ as a function of c (b) Relative performance of *OPT*, *AG-SD* and *DA-SD* as a function of c

Figure 5: Effect of the worst-case execution time, c.

We consider the effect of varying worst-case execution time. Figure 5(a) shows the variation of $\phi(\bar{f}, U)$ with increasing c. The results are normalized with respect to $\phi(\bar{f}, U)$ for $c = 20$. Recall from Section 3.2 that $\bar{f}$ represents the candidate local optimal frequency in Region 2 to be compared with U, which is the local optimal in Region 1. $\phi(\bar{f}, U)$ represents the net change in system energy. Figure 5(b) shows the relative ordering of the three schemes as a function of c. The values are normalized with respect to *AG-SD* when $c = 20$. Observe that for $c \leq 18$, $\phi(\bar{f}, U)$ is negative, implying that the gain in device energy consumption, ΔE_{device}, overshadows the loss in CPU energy, ΔE_{cpu}. Hence, the system energy benefits from running the processor at speeds higher than U and $f = \bar{f}$ is optimal in this region. On the other hand, for $c > 18$, ΔE_{cpu} starts to overshadow ΔE_{device}. Consequently, running at speeds higher than U starts to hurt system energy resulting in positive $\phi(\bar{f}, U)$ values. Hence, $f = U$ is the optimal in this spectrum.

It can be seen that in the interval where $c \leq 12$, *OPT* follows *DA-SD* as $\phi(\bar{f} = f_{ee}, U)$ is negative. Similarly, for $c \geq 18$, *OPT* follows *AG-SD* as $\phi(\bar{f}, U)$ is positive. For $12 < c < 18$ it can be seen that *OPT* follows neither *AG-SD* nor *DA-SD*. During this period $\bar{f} = f^*$ and $\phi(\bar{f} = f^*, U)$ is negative. Thus, *OPT* represents $E(f^*)$ which is optimal in this spectrum. Notice that for this example, the optimal frequency that minimizes the system-wide energy, changes from $f_{opt} = f_{ee}$ to $f_{opt} = f^*$ and finally to $f_{opt} = U$ with increasing utilization values. As f_{opt} transitions from f_{ee} to f^*, the device can no longer be put to sleep state af $f = f_{ee}$. Thus, running at $f = f_{ee}$ consumes more system energy compared to $f = U$, explaining the sharp increase in *DA-SD* at $c = 14$. The optimal frequency f^* when $12 < c < 18$ is found by neither *AG-SD* nor *DA-SD*. The energy optimal scheme, *OPT*, along with finding f^* also overcomes the bad performances of *AG-SD* and *DA-SD* at low and high utilization values respectively.

We emphasize that there is an interval where f_{opt} is neither U nor f_{ee}, but f^*, in the above results. Thus, the optimal scheme (*OPT*) is more than just determining at every point the better frequency in the set $\{U, f_{ee}\}$. In fact, there are regions where both of these well-known frequencies fail to minimize the system-wide energy consumption.

4. MULTIPLE-DEVICE MODEL

In this section, we generalize our solution to the case of multiple devices. A real-time application characterized by a worst-case execution time of c (under f_{max}) and a frame length of d is assumed to use m different devices $\{D_1 \dots D_m\}$ during the span of its execution. Each device D_i, has its own parameters (P_a^i, P_s^i, E_{sd}^i, E_{wu}^i, T_{sd}^i, T_{wu}^i) and is associated with a break-even time denoted by B_i. First, we formally define the problem.

Problem Statement: *Given a frame-based real-time application using m different devices, determine the CPU speed and device transitioning decisions so as to minimize the system-wide energy consumption.*

Since each device can be put to sleep state at the end of the execution, or remain in active state until the end of the frame, at first, it seems that there are 2^m possibilities that need to be examined. If true, this would imply an exponential-time algorithm. By careful analysis, we establish some important properties of the optimal solution, which enables us to develop an $O(m \log m)$-time algorithm.

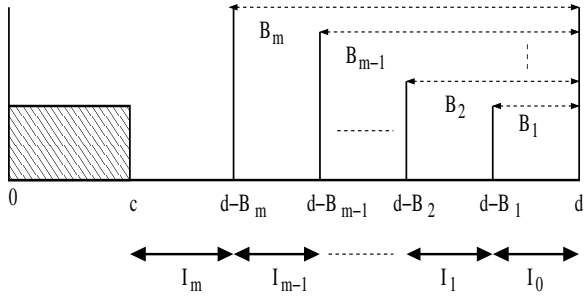


Figure 6: The ordering of the break-even times

Let R_{opt} and DS_{opt} denote the response time of the real-time application and the set of devices that will be transitioned to sleep state at the end of task execution in the optimal solution, respectively. We know that $R_{opt} \in [c, d]$. Without loss of generality, the break-even times are arranged in non-decreasing order, i.e. $0 < B_1 < \ldots < B_m < d - c$. With this ordering, we can divide $[c, d]$ into $m + 1$ intervals $\{[c, (d - B_m)] \ldots [(d - B_{i+1}), (d - B_i)] \ldots [(d - B_1), d]\}$ denoted by $\{I_m \ldots I_0\}$, respectively. The intervals I_0 to I_m are shown in Figure 6. R_{opt} belongs to exactly one of these $m + 1$ intervals[3].

For convenience, we divide our analysis into two steps that eventually will lead to an $O(m \log m)$-time algorithm.

- *Step*1: For each of these intervals, *assuming that R_{opt} lies in that interval*, we determine DS_{opt} which helps to evaluate the exact system-wide energy consumption function. Once we have the exact form of system-wide energy in an interval, we show how it can be minimized.

- *Step*2: Based on the analysis in *Step*1, we narrow down our analysis to at most $m + 2$ cases that have to be examined before we determine the optimal solution.

We now elaborate on these two steps and present our full analysis. By ordering the devices in non-decreasing order of break-even times, *Step*1 can be addressed as follows. If R_{opt} belongs to interval I_i, then devices $\{D_{i+1} \ldots D_m\}$ cannot be transitioned as the idle time is smaller than their break-even times. However, all devices $\{D_1 \ldots D_i\}$ *can and should be transitioned*. This is because if any device in the set $\{D_1 \ldots D_i\}$ is not transitioned to sleep state at the end of task execution, then by transitioning that device, we would effectively reduce device energy consumption and hence obtain a schedule with reduced system energy consumption. Based on this, one can quickly infer that if $R_{opt} \in I_m$, all m devices will be transitioned. Similarly, if $R_{opt} \in I_0$, then none of the devices will be transitioned.

Based on the interval to which R_{opt} belongs, we can exactly characterize the devices that should be transitioned to sleep states at the end of task execution. With this information, we can exactly formulate the system energy consumption function when $R_{opt} \in I_i$. *The number of devices transitioned to* sleep *states and hence the exact form of the system energy consumption remains the same as R_{opt} varies within a given interval and changes only when R_{opt} transitions between intervals.* Let $E_i(f)$ represent the system energy consumption when $R_{opt} \in I_i$. Specifically,

$$E_i(f) = (af^3 + \sum_{j=1}^{i} P_a^j)\frac{c}{f} + \sum_{j=i+1}^{m} P_a^j d + \sum_{j=1}^{i}(E_{sd}^j + E_{wu}^j)$$

Notice that in the formulation of $E_i(f)$, devices $\{D_1 \ldots D_i\}$ are transitioned while devices $\{D_{i+1} \ldots D_m\}$ are kept in active state throughout the frame. For uniformity, let us denote the lower and upper limits of interval I_i by LL_i and UL_i respectively. That is, $LL_0 = d - B_1$, $UL_0 = d$, $LL_m = c$, $UL_m = d - B_m$, $LL_i = d - B_{i+1}$ and $UL_i = d - B_i$, $\forall\, i = 1 \ldots m-1$. We can formalize the problem of minimization of E_i by enforcing that the response time of the task falls in interval I_i. This leads to the following constrained convex optimization problem for I_i denoted by OPT_i.

$$\text{minimize} \quad E_i(f) \tag{1}$$

$$\text{subject to} \quad -\frac{c}{f} + LL_i \leq 0 \tag{2}$$

$$\frac{c}{f} - UL_i \leq 0 \tag{3}$$

Constraints (2) and (3) make sure the response time of the application does not fall outside the range of interval I_i to which R_{opt} is assumed to belong.

PROPOSITION 1. *If the frequency f is the solution to the optimization problem OPT_i, then, $U \leq f \leq f_{max}$*

PROOF. If $f > f_{max}$, from both (2) and (3) it follows that the response time at $f > f_{max}$ is in the range $[LL_i, UL_i]$. Since $c = LL_m$, this implies $\frac{c}{f_{max}+\epsilon} \geq c$, which is a contradiction. Similarly. if $f < U$, from both (2) and (3) it follows that the response time at $f < U$ is in the range $[LL_i, UL_i]$. Now, since $UL_0 = d$, this implies $\frac{c}{U-\epsilon} \leq d$, which is again a contradiction. □

Let $f_i = \sqrt[3]{\frac{\sum_{j=1}^{i} P_a^j}{2a}}$. By observing that f_i is the value that sets the derivative of $E_i(f)$ to zero, we get the following.

PROPOSITION 2. *If f_i satisfies the conditions (2) and (3) then it is the solution to the optimization problem OPT_i.*

Observe that when $I_i = I_m$, we get $f_m = \sqrt[3]{\frac{P_{ind}}{2a}} = f_{ee}$, the traditional energy-efficient speed for a task using

[3]If two devices have the same break-even time or if $B_m = d - c$ then we will have less than $m + 1$ intervals. The same analysis can then be performed on this reduced interval set.

m devices *while ignoring DPM issues* [1]. Assuming that f_{ee} satisfies the response time constraints for interval I_m, an interesting observation at this point is that f_{ee} is *only* the local optimal solution for the interval I_m.

If f_i does not satisfy the constraints of the optimization problem then we proceed to find the optimal solution that satisfies the constraints.

LEMMA 1. *If f_i does not satisfy conditions (2) and (3) then, the optimal solution to OPT_i is either $f = \frac{c}{LL_i}$ or $f = \frac{c}{UL_i}$.*

The proof of Lemma 1 is presented in Appendix.

While Lemma 1 solves *Step*1 of the analysis, the following Corollary connects *Step*1 and *Step*2.

COROLLARY 1. *If $\forall i,\ i = 0 \ldots m,\ f_i \notin I_i$ then in the optimal solution for m devices, $R_{opt} \in \{c, (d-B_m), \ldots, (d-B_1), d\}$.*

Corollary 1 states that if for all the $(m+1)$ intervals, I_i, $i = 0 \ldots m$, f_i does not satisfy the conditions (2) and (3) of the optimization problem OPT_i, then in the optimal solution the response time of the application is limited to the set $\{c, (d-B_m), \ldots, (d-B_1), d\}$. *That is, if the given conditions hold, in the optimal solution, the slack of the application should be exactly equal to* 0, $d - c$ *or one of the break-even times* $\{B_i\}$.

Observe that the upper limit of interval I_i becomes the lower limit for interval I_{i-1}. Say $UL_i = LL_{i-1} = \beta$. We evaluate the energy consumption when $R_{opt} = \beta$ twice, once in interval I_i and again in interval I_{i-1}. The difference is that in interval I_i, we assume device D_i is transitioned while in interval I_{i-1} we assume D_i is not transitioned. This totally covers at most $2m$ cases were $R_{opt} = d - B_i,\ i = 1 \ldots m$. Further, we also evaluate the energy consumption when $R_{opt} = c$ and $R_{opt} = d$. Thus, an implication of Corollary 1 is that there are at most $2m + 2$ cases that need to be examined to determine the optimal solution.

Observe that $f = U$ is a natural candidate for the solution based on Corollary 1. In fact, for the single-device model where $m = 1$, it was shown that $f = U$, is *always* a legitimate candidate. Lemma 2 below, reduces the $2m + 2$ cases to be examined as given by Corollary 1 to at most $m + 2$ cases.

LEMMA 2. $E_i(f = \frac{c}{UL_i}) \leq E_{i-1}(f = \frac{c}{LL_{i-1}})$

PROOF. Assume $E_i(f = \frac{c}{UL_i}) > E_{i-1}(f = \frac{c}{LL_{i-1}})$. Since $UL_i = LL_{i-1} = \beta$, running the processor at $f = \frac{c}{UL_i}$ gives the same response time as running the processor at $f = \frac{c}{LL_{i-1}}$. Consequently, the CPU energy consumption in both cases is the same. Note that in both cases devices $\{D_{i+1} \ldots D_m\}$ are not transitioned while devices $\{D_1 \ldots D_{i-1}\}$ are transitioned. This being the case the difference in system energy consumption in the two cases, if any, must come from the energy consumption of device D_i. Hence we can say that if $E_i(f = \frac{c}{UL_i}) > E_{i-1}(f = \frac{c}{LL_{i-1}})$, then $E^i_{sd} + E^i_{wu} > (d - \beta)P^i_a$. Since $d - \beta \geq B_i$, this is a contradiction. □

An interesting question is whether there exists a pattern among these final $m + 2$ cases that can be further exploited by optimization techniques. Let EC_{opt} denote the set of energy consumption values obtained by evaluating the final $m + 2$ cases. Unfortunately, the following observation shows that the relative ordering of the values in the set EC_{opt} can be arbitrary.

OBSERVATION 1. *The relative ordering of the $(m+2)$ values in the set EC_{opt} does not exhibit a special pattern.*

We give an example to justify the above observation. Consider a real time application with $c = 10$ and $d = 30$. It uses four devices $D_1(P^1_a = 0.2, B_1 = 5)$, $D_2(P^2_a = 0.15, B_2 = 10)$, $D_3(P^3_a = 0.5, B_3 = 15)$ and $D_4(P^4_a = 0.4, B_4 = 17)$. Let us assume $a = 1$ and $T_{sw} \leq B^i_{actual}$ for all devices. $B^i_{actual} = \frac{E^i_{sd} + E^i_{wu}}{P^i_a},\ i = 1 \ldots 4$. In these settings, we can verify that $f_4 = 0.855,\ f_3 = 0.752,\ f_2 = 0.559$ and $f_1 = 0.464$. Notice that $\forall i,\ \frac{c}{f_i} \in I_i$. In interval I_0, $f = U$ is the best as no devices are transitioned to sleep states. With the above data we can verify $E(f = U) = 38.611$, $E(f = f_1) = 38.963$, $E(f = f_2) = 38.886$, $E(f = f_3) = 38.958$ and $E(f = f_4) = 38.730$.

Notice how the interval-optimal energy consumption $E_i(f)$ first increases, next decreases, then increases before decreasing once again, as we move from the first candidate frequency f_1 to f_2, f_3 and f_4. This shows that the energy consumption values of the final $m + 2$ cases, need not to have a well-defined relationship which can be exploited by optimization techniques. As an implication, it turns out that it is indeed necessary to evaluate and compare the $m+2$ candidate cases for the optimal solution.

4.1 Computing the Optimal Speed Efficiently

Based on the above characterizations we formulate an $O(m \log m)$ algorithm, given in Figure 7, to find the optimal speed for the multiple-device model. As an implication of Observation 1, it is necessary to compare the best energy consumptions obtained by assuming $R_{opt} \in I_i,\ i = 0 \ldots m$ to obtain the global optimal. From Lemma 1 and 2, in every interval $I_i,\ i \neq m$, if $f = f_i$ does not satisfy the response time constraints then it is sufficient to evaluate and compare energy consumption at $f = UL_i$. In I_0, $f_0 = 0$. Hence, we start out with the assumption that in the optimal solution $f = U$ (lines $4 - 5$). The E_{best} variable holds the minimum system energy consumption value encountered so far and the f_{best} holds the corresponding frequency. In lines $6 - 17$, we consider cases where the optimal response time of the application is assumed to belong to each of the remaining m intervals $I_1 \ldots I_m$. For each such interval we compute f_i. Based on whether or not f_i satisfies the response time constraints we compare energy consumption at either $f = f_i$ or $f = UL_i$ with E_{best}. For interval I_m if f_m does not satisfy the response time constraints then it is necessary to evaluate and compare energy consumption at $LL_m = c$, as c does not act as an upper limit to any interval. In lines $18 - 21$, we perform this final comparison. At the end, f_{best} holds the optimal value of f that minimizes system energy consumption.

Time Complexity: Sorting the devices based on break-even times requires $O(m \log m)$ time. The algorithm performs a constant time comparison in every interval and there are at most $m + 1$ intervals. Thus, the complexity of the algorithm is $O(m \log m)$.

Function *Optimal Speed:*

$P_{ON} = \sum_{i=1}^{m} P_a^i$
$P_{OFF} = 0$
$E_T = 0$
$E_{best} = aU^2c + P_{ON}d$
$f_{best} = U$
for $i = 1$ *to* m
  $P_{ON} = P_{ON} - P_a^i$
  $P_{OFF} = P_{OFF} + P_a^i$
  $E_T = E_T + E_{sd}^i + E_{wu}^i$
  $f_i = \sqrt[3]{\frac{P_{OFF}}{2a}}$
  if $(\frac{c}{f_i} \in I_i)$ then $f = f_i$
  else $f = \frac{c}{UL_i} = \frac{c}{d-B_i}$
  if $(E(f) < E_{best})$
    Set $E_{best} = E(f)$
    Set $f_{best} = f$
  endif
endfor
if $(\frac{c}{f_m} \notin I_m \quad AND \quad E(f_{max}) < E_{best})$
  Set $E_{best} = E(f_{max})$
  Set $f_{best} = f_{max}$
endif
return f_{best}

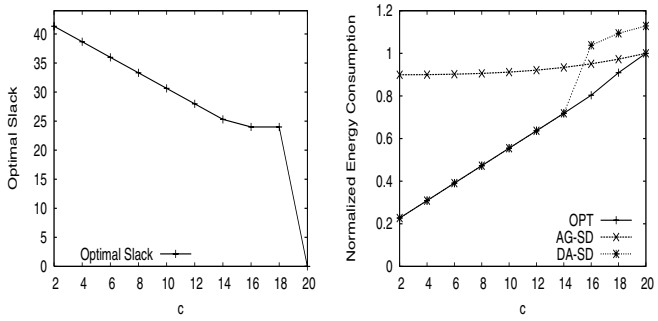
Function *System Energy:*

$E(f) = af^2c + P_{ON}d + P_{OFF}\frac{c}{f} + E_T$

Figure 7: Algorithm to Compute Optimal Speed (Multiple-Device Case)

4.2 Experimental Evaluation

The experimental methodology we follow in this section is the same as the one described in Section 3.3. Again, the real-time application has a frame length of $44ms$. It uses three devices during its execution: IBM Microdrive ($B = 24ms$), Realtek Ethernet Chip ($B = 20ms$) and Simple Tech Flash Card ($B = 4ms$). The specifications of these devices are from [5]. Among the devices under consideration, IBM Microdrive has the largest break-even time equal to $24ms$.



(a) Optimal Slack as a function of c

(b) Relative performance of *OPT*, *AG-SD* and *DA-SD* as a function of c

Figure 8: Multiple Device Model, effect of worst-case execution time, c

We consider the effect of worst-case execution time on both optimal system slack and optimal system energy consumption. Figure 8(a) shows the optimal slack which minimizes system-wide energy as a function of c. Figure 8(b) shows the relative performance of the three schemes. The energy values are normalized with respect to *AG-SD* when $c = 20ms$. The optimal slack decreases uniformly with increasing utilization in the range $2 \leq c \leq 14$. In this interval, $f = f_{ee}$ is optimal, and the *OPT* scheme follows *DA-SD*. For values in the range $14 < c < 20$, *OPT* is significantly different from both *DA-SD* and *AG-SD*. During this period one or more devices cannot be transitioned at $f = f_{ee}$, which explains the sharp increase in *DA-SD* at $c = 16$. The step like behavior of the optimal slack is also a consequence of the optimal frequency shifting from $f = f_{ee}$ to an intermediate value between f_{ee} and U. Note that depending on the power characteristics of devices and frame length, the sharp increase in *DA-SD* scheme may also occur at an earlier stage than the one shown in figure. In such cases, the advantages of our optimal scheme is even more pronounced.

In models minimizing the system-wide energy *while ignoring device transition overheads and DPM related issues*, *DA-SD* scheme was shown to be optimal assuming it satisfies feasibility constraints [1]. Observe that *AG-SD* outperforms *DA-SD* in the spectrum $c \geq 16$ in Figure 8(b). Due to device transition overheads, as mentioned before, transitioning devices at $f = f_{ee}$ is not always possible. When $c = 16$, IBM Microdrive cannot be transitioned at f_{ee} and remains active throughout the frame significantly increasing device energy consumption. The CPU power consumption rate is significantly high compared to that of Flash Card and Ethernet Chip. Thus, with IBM Microdrive in active state throughout the frame, the CPU energy savings in scheme *AG-SD* dominates the device energy saving obtained by transitioning Flash Card and Ethernet Chip in *DA-SD*. This explains the reason why *AG-SD* outperforms *DA-SD*.

Finally, at $c = 20$, *OPT* follows *AG-SD*. Observe that for the devices considered $T_{sw}^i > B_{actual}^i$. As a result, the device break-even time, defined as $max(T_{sw}^i, B_{actual}^i)$, is dominated by the device transition times. Thus, even at $c = 20$, all the three devices can be efficiently transitioned yielding lower device energy consumption. However, when $c > 20$, the slack is not large enough to produce a device transitioning decision, involving transitioning at least one device to sleep state, which can reduce device energy consumption to an extent that it overshadows the increase in CPU energy by running the processor at speeds higher than $f = U$. Thus, *AG-SD* is optimal in that region.

5. CONCLUSIONS

In this work, we addressed the problem of system-wide energy minimization through a novel approach. Unlike prior studies, our system-level energy model considered both DVS- and DPM-related issues and accounted for device transition overheads. With this general model, we were able to characterize the exact interplay between DVS and DPM formally. By deriving useful properties from this characterization, we formulated a $O(m \log m)$-time algorithm (where m is the number of devices) to determine the CPU speed and device transitioning decisions to minimize the system-wide energy. Experimental evaluations using real device parameters demonstrated the potential benefits of our optimal scheme. To the best of our knowledge, this is the first work formally investigating the interplay between two well-known energy management techniques for real-time embedded systems: DVS and DPM, by also considering the device transition overheads at the same time. We believe that the

non-trivial the interplay of DVS and DPM as portrayed in this paper provides valuable insights to system-level energy management in real-time embedded systems.

6. REFERENCES

[1] H. Aydin, V. Devadas, and D. Zhu. System-level energy management for periodic real-time tasks. In *Proceedings of the 27th IEEE Real-Time Systems Symposium (RTSS'06)*, 2006.

[2] H. Aydin, R. Melhem, D. Mosse, and P. Mejia-Alvarez. Power-aware scheduling for periodic real-time tasks. *IEEE Transactions on Computers*, 53(10):584–600, May 2004.

[3] L. Benini, A. Bogliolo, and G. D. Micheli. A survey of design techniques for system-level dynamic power management. *IEEE Transactions on VLSI Systems*, 8(3):299–316, 2000.

[4] Cheng and S. Goddard. Sys-edf: A system-wide energy efficient scheduling algorithm for hard real-time systems. *International Journal of Embedded Systems on Low Power Real-Time Embedded Computing*, 4(4):45–56, 2007.

[5] H. Cheng and S. Goddard. Online energy-aware i/o device scheduling for hard real-time systems. In *Proceedings of Design Automation and Test in Europe (DATE'06)*, 2006.

[6] V. Devadas and H. Aydin. Real-time dynamic power management through device forbidden regions. In *Proceedings of the IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'08)*, 2008.

[7] P. J. M. Havinga and G. J. M. Smith. Design techniques for low-power systems. *Journal of Systems Architecture*, 46(1), 2000.

[8] R. Jejurikar and R. Gupta. Dynamic voltage scaling for system-wide energy minimization in real-time embedded systems. In *Proceedings of the 2004 International Symposium on Low Power Electronics and Design (ISLPED'04)*, 2004.

[9] D. Luenberger. *Linear and Nonlinear Programming*. Addison-Wesley, Reading Massachusetts, 1984.

[10] M. Pedram. Power minimization in ic design: Principles and applications. *ACM Transactions on Design Automation of Electronics Systems*, 1(1):3–56, 1996.

[11] P. Pillai and K. G. Shin. Real-time dynamic voltage scaling for low-power embedded operating systems. In *Proceedings of the ACM Symposium on Operating Systems Principles (SOSP'01)*, 2001.

[12] A. Qadi, S. Goddard, and S. Farritor. A dynamic voltage scaling algorithm for sporadic tasks. In *Proceedings of the IEEE Real-Time Systems Symposium (RTSS'03)*, 2003.

[13] C. Rusu, R. Melhem, and D. Mosse. Maximizing rewards for real-time applications with energy constraints. *ACM Transactions for Embedded Computing Systems*, 2(4), 2003.

[14] S. Saewong and R. Rajkumar. Practical voltage-scaling for fixed-priority real-time systems. In *Proceedings of the IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'03)*, 2003.

[15] D. Snowdon, S. Petters, and G. Heiser. Accurate online prediction of processor and memory energy usage under voltage scaling. In *Proceedings of the International Conference On Embedded Software*, 2007.

[16] V. Swaminathan and K. Chakrabarty. Energy conscious deterministic i/o device scheduling in hard real-time systems. In *Proceedings of the International Conference in Computer Aided Design (ICCAD'03)*, 2003.

[17] V. Swaminathan and K. Chakrabarty. Pruning-based, energy-optimal deterministic i/o scheduling for hard real-time systems. *ACM Transactions on Embedded Computing Systems*, 4(1):141–167, 2005.

[18] V. Swaminathan, K. Chakrabarty, and S. S. Iyengar. Dynamic i/o power management for hard real-time systems. In *Proceeding of the International Conference on Hardware-Software Co-design and System Synthesis (CODES'01)*, 2001.

[19] M. Weiser, B. Welch, A. Demers, and S. Shenker. Scheduling for reduced cpu energy. In *USENIX Symposium on Operating Systems Design and Implementation*, 1994.

[20] R. Xu, D. Mosse, and R. Melhem. Minimizing expected energy consumption in real-time systems through dynamic voltage scaling. *ACM Transactions on Computer Systems*, 25(4), 2007.

[21] R. Xu, C. Xi, R. Melhem, and D. Mosse. Practical pace for embedded systems. In *Proceedings of the ACM International Conference on Embedded Software (EMSOFT'04)*, 2004.

[22] X. Zhong and C.-Z. Xu. System-wide energy minimization for real-time tasks: Lower bound and approximation. In *Proceedings of the Int'l Conference on Computer-Aided Design (ICCAD'06)*, 2006.

[23] X. Zhong and C.-Z. Xu. Frequency-aware energy optimization for real-time periodic and aperiodic tasks. In *Proceedings of the Conference on Languages, Compilers and Tools for Embedded Systems (LCTES'07)*, 2007.

[24] J. Zhuo and C. Chakrabarti. System-level energy-efficient dynamic task scheduling. In *Proceedings of Conference on Design Automation (DAC)*, 2005.

APPENDIX: Proof of Lemma 1

The solution f to the optimization problem OPT_i must also satisfy the following Kuhn-Tucker conditions for convex programs [9].

$$2af^3 - \sum_{j=1}^{i} P_a^j + (\mu_1 - \mu_2) = 0 \tag{4}$$

$$\mu_1(-\frac{c}{f} + LL_i) = 0 \tag{5}$$

$$\mu_2(\frac{c}{f} - UL_i) = 0 \tag{6}$$

$$\forall i,\ \mu_i \geq 0 \tag{7}$$

PROPOSITION 3. *In the solution to OPT_i both μ_1 and μ_2 cannot be strictly positive.*

PROOF. Assume both μ_1 and μ_2 are strictly positive, i.e $\mu_i > 0,\ i = 1, 2$. Under this assumption there must exist an f that satisfies the following two conditions.

$$-\frac{c}{f} + LL_i = 0 \tag{8}$$

$$\frac{c}{f} - UL_i = 0 \tag{9}$$

But, since $LL_i \neq UL_i$ this is impossible. □

From Proposition 3 we know that both μ_1 and μ_2 cannot be strictly positive. Observe that by setting $\mu_1 = \mu_2 = 0$ we get $f = f_i$, the frequency which minimizes $E_i(f)$. Considering other possibilities:

Case1: Assume $\mu_1 > 0$ and $\mu_2 = 0$. Then from Equation (5), $-\frac{c}{f} + LL_i = 0$. This implies $f = \frac{c}{LL_i}$ is a candidate optimal solution.

Case2: Assume $\mu_1 = 0$ and $\mu_2 > 0$. Then from Equation (6), $\frac{c}{f} - UL_i = 0$. This implies $f = \frac{c}{UL_i}$ is a candidate optimal solution.

Thus, either $f = \frac{c}{LL_i}$ or $f = \frac{c}{UL_i}$ is the solution to OPT_i.