Minimizing Expected Energy Consumption through Optimal Integration of DVS and DPM*

Baoxian Zhao Hakan Aydin
Department of Computer Science
George Mason University
Fairfax, VA 22030
bzhao@gmu.edu, aydin@cs.gmu.edu

ABSTRACT

While Dynamic Voltage Scaling (DVS) and Dynamic Power Management (DPM) techniques are widely used in real-time embedded applications, their complex interaction is not fully understood. In this research effort, we consider the problem of minimizing the expected energy consumption on settings where the workload is known only probabilistically. By adopting a system-level power model, we formally show how the optimal processing frequency can be computed efficiently for a real-time embedded application that can use multiple devices during its execution, while still meeting the timing constraints. Our evaluations indicate that the new technique provides clear (up to 35%) energy gains over the existing solutions that are proposed for deterministic workloads. Moreover, in a non-negligible part of the parameter spectrum, the algorithm's performance is shown to be close to that of a clairvoyant algorithm that can minimize the energy consumption with the advance knowledge about the exact workload.

Categories and Subject Descriptors

H.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

1. INTRODUCTION

Energy management has become one of the most important goals in modern embedded system design, especially for battery-operated systems. In this regard, *Dynamic voltage scaling* (DVS) is recognized as one of most effective and fundamental techniques. By considering the strictly convex relationship between the supply voltage and CPU power consumption, DVS attempts to save energy by scaling down the frequency along with the supply voltage. Early DVS studies can be classified in two categories: The *inter-task* DVS schemes [2, 10, 12] focus on allotting the CPU time to multiple tasks and perform frequency scaling only at task preemption/completion points. On the other hand, in *intra-task* DVS schemes [9, 17–19], the frequency changes can occur while a task is executing (in its allocated CPU time). In recent DVS research [1, 7], the concept of *energy-efficient frequency* is introduced after the researchers have observed that processing frequencies below a certain threshold can have negative effects on the system-wide energy consumption. This energy-efficient frequency is computed by balancing the off-chip device energy and CPU energy consumed during the task's execution.

Another widely-used energy management technique is *Dynamic Power Management* (DPM) that attempts to put the idle system components into low-power states whenever possible. In fact, off-chip devices (such as I/O devices and the main memory) have an *active* state and at least one low-power *sleep* state. However, significant transition energy/time overheads are involved in state transitions of the devices. In fact, a minimum idle interval length (called the *device break-even time*) is needed in DPM to justify the transition of the device to the *sleep* state. DPM reduces energy by putting the device into low power *sleep* state when the idle time is predicted to be no less than the device break-even time. DPM has been well-studied in the recently proposed power management schemes targeting different task/device settings [3, 6, 14, 15].

While the research efforts that focus on only on DVS or DPM are many, solutions that propose integrating both policies under a unified framework are relatively few [4, 5, 8, 13, 20]. Authors in [13] apply a stochastic DPM policy by using the different DVS voltage levels as multiple active power modes. The work in [20] proposes a DVS-DPM policy that maximizes the operational lifetime of an embedded system powered by a fuel cell based hybrid power source. The frequency scaling level is chosen in [8] by investigating the trade-offs between the DVS-enabled CPU and the DPM-enabled devices. The SYS-EDF algorithm is a heuristic-based energy management scheme for periodic real-time tasks [4].

The growing importance of system-wide energy management clearly mandates integration of both policies. Yet most of the existing solutions, while noteworthy, adopt heuristic-

*This work is supported in part by US National Science Foundation grants CNS-0720647, CNS-0720651 and CNS-546244 (CAREER Award).

ICCAD'09, November 2–5, 2009, San Jose, California, USA.

based solutions with no performance guarantees. In fact, it is somewhat surprising that the full solution for a single real-time embedded application using both DVS and DPM policies was published only recently [5]. That work captures the intriguing trade-off between DVS and DPM policies in a precise way.

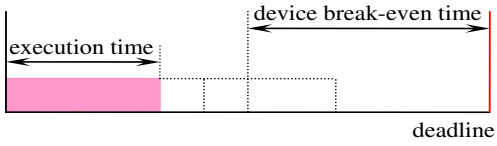

Figure 1: The interplay of DVS and DPM

DVS and DPM solutions often work against each other. As shown in Figure 1, if the processing frequency is lowered through DVS to save energy, the task execution time is extended. As a result, the idle time is shortened, which prevents DPM from putting the devices into the low-power sleep states. On the other hand, the device energy can be reduced by executing the task at higher frequency to obtain enough idle time for putting the devices to the low power sleep state while this results in additional transition energy overhead and CPU energy consumption. Moreover, the application may be using multiple devices with different power and break-even time characteristics, making an optimal solution non-trivial. An exact and formal characterization of the interplay between DVS and DPM for a real-time application using multiple devices was recently obtained in [5]. In the same work, an algorithm to compute the optimal processing frequency was also derived. However, the solution in [5] is derived assuming a known, worst-case workload. In other words, the solution is optimal only for *deterministic* workloads. As we show later in this paper, if the workload exhibits probabilistic behavior, minimizing *expected* energy is more important and the algorithm in [5] (called *DET* throughout this work) becomes sub-optimal, even overly pessimistic.

Although the task's actual workload cannot be predicted in advance with full accuracy, its probability distribution function can be obtained or estimated through *profiling* [9, 17–19]. Our primary goal in this paper is to determine the optimal processing frequency to minimize the expected energy at the *system-level*, when the cumulative distribution of the application's workload is known. We consider both DVS and DPM features and formally characterize the expected energy. We illustrate our technique by first focusing on an application using a single-device and show how the optimal frequency can be derived in linear-time. Then, we extend the solution to the case of multiple devices. Our solutions also ensure that the timing constraint of the application is met in every execution scenario. Our experimental evaluation shows that our solution yields energy savings of up to 35% over the *DET* algorithm [5] by exploiting the probabilistic information. Moreover, it approaches the performance of a *clairvoyant* algorithm that can compute the best solution using the workload information before-hand. To the best of our knowledge, this study is the first to exploit DVS and DPM properties *optimally* to minimize the *expected system-level energy consumption*, for workloads that are known only *probabilistically* before execution.

2. SYSTEM MODELS

2.1 Application Model

We consider a real-time embedded application that is invoked repetitively with the period P. Such applications are known as *frame-based* systems in the literature [5, 11, 17]. Whenever invoked, the application must complete its execution within the relative deadline d, which is assumed to be equal to P. The application's workload, characterized by the *number of cycles*, is assumed to be known only probabilistically: it can assume values between a lower bound and an upper bound that are given by *best-case execution cycles (bcc)* and *worst-case execution cycles (wcc)*, respectively. The cumulative distribution function for the application's workload is:

$$F(x) = p(X \leq x) \quad (1)$$

where X is the random variable for the application's CPU cycle demand and $p(X \leq x)$ represents the probability that the application will not require more than x cycles within a given frame. To approximate the cumulative distribution function $F(x)$, we use the effective *histogram* technique [17, 18]. Specifically, we assume that the application's workload function is partitioned into n *cycle groups*: $(b_0, b_1], (b_1, b_2], ..., (b_{n-1}, b_n]$, where $b_0 = bcc$, $b_n = wcc$ and $b_i < b_{i+1}$. As shown in Figure 2, the probability of executing the application with no more than x cycles is estimated conservatively as $F(b_{i+1})$, when x is in the interval $(b_i, b_{i+1}]$. Notice that $F(b_n) = 1$. Through profiling, the probability distributions for the cycle groups can be obtained [17, 18]. Also, we define the *size* of the cycle group i as $c_i = b_i - b_{i-1}$ for $i > 0$ and $c_i = b_i$ for $i = 0$.

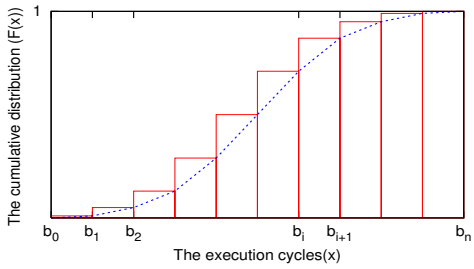

Figure 2: Histogram-based estimation of the application's probabilistic workload

We assume a DVS-enabled system where the processing frequency f can change from a minimum frequency f_{min} to a maximum frequency f_{max}. All frequency values are normalized with respect to f_{max}. Finally, the *utilization* of the application is defined as $U = \frac{wcc}{d} \leq 1$ and $WCET = \frac{wcc}{f_{max}}$ denotes its worst-case execution time when its executes its maximum possible workload wcc with f_{max}.

2.2 Device Model

The real-time embedded application uses a set of m devices $D = \{D_1, D_2, .., D_m\}$ during its execution. Each device D_i can be either in *active* or *sleep (low-power)* state, and is defined with the following parameters:

- P_a^i: The device power consumption in *active* state
- P_s^i: The device power consumption in *sleep* state

- E^i_{as} (E^i_{sa}): The energy overhead associated with active-to-sleep (sleep-to-active) state transition
- T^i_{as} (T^i_{sa}): The time overhead associated with active-to-sleep (sleep-to-active) state transition

Due to the periodic execution pattern, each active-to-sleep transition for a device will be eventually followed by a sleep-to-active transition. As a result, to conveniently represent the overheads, we define $E^i_{tr} = E^i_{as} + E^i_{sa}$ as the total device transition energy overhead and $T^i_{tr} = T^i_{as} + T^i_{sa}$ as the total transition delay. Following recent literature [3, 5, 6, 14–16], we assume *inter-task device scheduling*. In inter-task device scheduling, all devices used by the real-time application must be in *active* state while it executes. In fact, the absence of knowledge about the exact time instants when the application will need a specific device and non-trivial state transition costs easily justifies the inter-task device scheduling paradigm [3, 5, 6, 15].

However, the devices can be put to *sleep* state when application finishes its execution within a frame (until the beginning of the next frame). Considering the non-trivial energy and time overheads associated with state transitions, the device *break-even time* B_i is defined to represent the lower bound on the idle interval length so that putting the device to *sleep* state can be justified at run-time. In addition, no idle interval can be smaller than T^i_{tr}; hence, B_i is given as [3–6]:

$$B_i = \max(T^i_{tr}, \frac{E^i_{tr} - T^i_{tr} P^i_s}{P^i_a - P^i_s})$$

2.3 Energy Model

The total system energy consumption within a frame is the sum of static and dynamic energy consumptions. Following previous DPM work [3–6], we assume that shutting down the entire system within a frame is not an option (hence, static power is not manageable) and concentrate on the dynamic energy consumption E_d. E_d is a function of several factors, including the CPU frequency f and power characteristics and states of individual devices (such as main memory and I/O modules). For simplicity, we assume that all device active powers are given in excess of the device sleep powers, as in [4–6].

If the application executes c cycles at the frequency f within a given frame, it will complete its execution within $t = \frac{c}{f}$ time units. Within that frame, the interval $[0, \frac{c}{f}]$ is called the *execution period*. Similarly, the interval $[\frac{c}{f}, d]$ is called the *slack period*. During the slack period, the CPU is idle and the devices used by the application can be potentially put to *sleep* states by incurring transition energy overheads. However, such a transition for a device D_i is energy-efficient if only if $d - B_i > \frac{c}{f}$. For example, in Figure 3, the device D_1 can be put to sleep state during the slack period, while D_2 is forced to remain in active mode.

Let D_A denote the subset of devices that are forced to remain in *active* state during the slack period, due to short idle interval lengths. On the other hand, the devices in $D - D_A$ can be transitioned to *sleep* state during the slack period to save energy. The dynamic energy consumption $E_d(f)$ within a frame can be formally expressed as [5]:

$$E_d(f) = E_e(f) + E_s(f) + E_t(f) \quad (2)$$

where

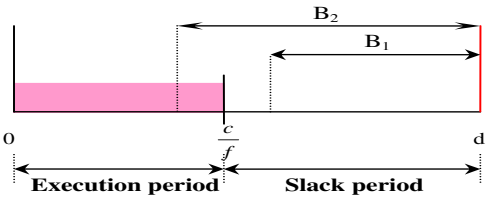

Figure 3: Execution and slack periods of an application

$$E_e(f) = (af^3 + \sum_{i=1}^{m} P^i_a)\frac{c}{f} \quad (3)$$

is the sum of the energy consumed by the CPU ($af^3 \cdot \frac{c}{f}$) and the devices ($\sum_{i=1}^{m} P^i_a \frac{c}{f}$) during the execution period;

$$E_s(f) = \sum_{i|D_i \in D_A} P^i_a (d - \frac{c}{f}) \quad (4)$$

is the energy consumed by the devices that remain in active state during the slack period, and,

$$E_t(f) = \sum_{i|D_i \in (D - D_A)} E^i_{tr} \quad (5)$$

is the transition energy incurred by the devices in $D - D_A$, which need to be activated at the beginning of the next frame.

3. SYSTEM-LEVEL ENERGY: STOCHASTIC CASE

3.1 Motivational Example

By using a system-level energy model very similar to that given in Section 2.3, the work in [5] derived a precise characterization of the interplay between DVS and DPM, and then showed how to obtain the optimal frequency to minimize system-level energy. A fundamental characteristic of that solution is that it assumes a deterministic workload equal to *wcc*. While provisioning for worst-case scenarios is important to guarantee the timing constraints, in practice, many real-time embedded applications complete early without consuming their worst-case workloads. Further, although the actual number of execution cycles cannot be known in advance exactly, its probability distribution can be obtained through profiling [17, 18]. This, in turn, provides new and significant opportunities to minimize the *expected* system-level energy consumption, while meeting the application's deadline, as the following example illustrates.

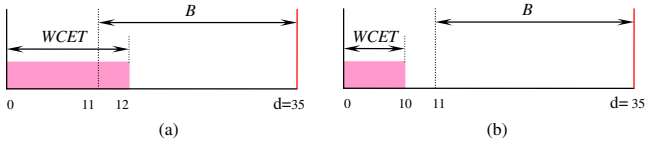

Figure 4: An application with $WCET = 12$ (a) and with $WCET = 10$ (b), using the IBM Microdrive device

Let us consider an application with deadline $d = 35ms$ running on a system with maximum frequency $1GHz$, using

a single IBM Microdrive device ($P_a = 1.3$ *Watt*, $E_{tr} = 12$ *Joules* and $B = 24ms$, as specified in [3, 5]). To start with, the analysis in [5] considers only the WCET information as the basis; hence, if $d - B < WCET$, the *DET* algorithm [5] would assume that the device will need to be kept in *active* mode during the slack period. For example, if $wcc = 12 \times 10^6$, *DET* would choose the optimal frequency as $f = 0.34Ghz$ (effectively planning to complete the application just at the deadline in the worst-case) since $WCET = 12ms > d - B = 11ms$ – as seen in Figure 4.a. Now assume that the application's workload is known probabilistically, and that it changes between $bcc = 2\times10^6$ and $wcc = 12\times10^6$ according to the normal distribution with the mean $\frac{bcc+wcc}{2}$ and the standard derivation $\frac{wcc-bcc}{12}$. While *DET* solution indeed minimizes energy for the worst-case workload, one can compute that the *expected* energy consumption with $f = 0.34Ghz$ is $E_{DET} = 47.71$ *Joules*. On the other hand, by executing the application at $f = 0.76GHz$, one can obtain an expected energy consumption of $E = 29.70$ *Joules*, a savings of 35% over that of *DET*, while still meeting the deadline. In fact, improvements due to a probabilistic analysis do also exist when $WCET \leq d - B$. For instance, if we change wcc as 10×10^6, as showed in Figure 4.b, then we find out that *DET* will yield the frequency $f = 0.91Ghz$ with the corresponding expected energy $E_{DET} = 27.79$ *Joules*. But if we use $f = 0.69Ghz$, the expected energy becomes $E = 26.45$ *Joules*, leading to approximately 5% additional energy savings. As this example suggests, minimizing the expected energy consumption while exploiting the subtle interaction between DVS and DPM warrants a full analysis.

3.2 Deriving Expected Energy Function

Given a processing frequency f and a cumulative distribution function for the workload, the expected energy consumption E_m within a frame can be written as the sum of three components: the expected energy consumption in the execution period $\overline{E}_e(f)$, that in the slack period $\overline{E}_s(f)$, and the expected transition energy overhead $\overline{E}_t(f)$:

$$E_m(f) = \overline{E}_e(f) + \overline{E}_s(f) + \overline{E}_t(f) \tag{6}$$

Let us elaborate on each of these components separately. **The execution period energy** $\overline{E}_e(f)$: During its execution, the application will execute a given cycle x in the range $[bcc, wcc]$ with a certain probability. In fact, this probability is equal to $(1-F(x))$, where $F(x)$ is the cumulative distribution function defined in Equation (1) [17,18]. Consequently, the expected value of overall (CPU + device) energy consumption during the execution period can be written as:

$$\begin{aligned}\overline{E}_e(f) = &\sum_{x=1}^{wcc}((1-F(x))af^3\tfrac{1}{f} + \sum_{i=1}^{m}P_a^i(1-F(x))\tfrac{1}{f}) \\ = &\sum_{x=1}^{wcc}(af^3 + \sum_{i=1}^{m}P_a^i)(1-F(x))\tfrac{1}{f}\end{aligned} \tag{7}$$

Further, by applying the histogram-based estimation technique, $\overline{E}_e(f)$ can be formally re-written as:

$$\overline{E}_e(f) = \sum_{j=0}^{n-1}(af^3 + \sum_{i=1}^{m}P_a^i)\frac{c_j}{f}(1-F(b_j)) \tag{8}$$

The slack period energy $\overline{E}_s(f)$: The energy consumption during the slack period is due to devices that are forced to remain in *active* state when their completion time within the frame does not allow an energy-efficient transition. Specifically, when the application completes at time $t > d - B_i$ for a given device D_i, that device will consume a total energy of $P_a^i(d-t)$ during the slack period by staying in *active* mode. Completion at each of such time instants occurs with a certain probability; hence, we find:

$$\overline{E}_s(f) = \sum_{i=1}^{m}\sum_{Z=d-B_i}^{d}p(t=Z)P_a^i(d-t) \tag{9}$$

where $p(t = Z)$ is the probability that the application will complete exactly at time Z.

The transition energy $\overline{E}_t(f)$: When the application completes at time $t < d-B_i$ the device D_i can and should be transitioned to *sleep* state during the slack period. However, each such transition will result in an energy overhead of E_{tr}^i. If t is the completion time, the expected value of E_{tr}^i is $p(t \leq d - B_i)\ E_{tr}^i$. Observe that:

$$p(t \leq d - B_i) = p(\frac{X}{f} \leq d - B_i) = p(X \leq f(d - B_i))$$

where X is the random variable for the cycle demand of the application and f is the processing frequency. Recalling that $p(X \leq x) = F(x)$, we get:

$$\overline{E}_t(f) = \sum_{i=1}^{m}F((d - B_i)f) \cdot E_{tr}^i \tag{10}$$

At this point, we are ready to develop our solution for the problem of minimizing the expected overall energy $E_m(f)$ while considering the probabilistic distribution of execution cycles while satisfying the deadline constraint.

4. SINGLE-DEVICE MODEL

In this section, we consider the case where the application uses only $m = 1$ device during its execution and derive the optimal frequency to minimize the expected energy by exploiting the interaction between DPM and DVS. This also allows us to lay the technical background for the more general case that will be addressed in Section 5. For simplicity, we use the notations $P_a^1 = P_a$, $B_1 = B$ and $E_{tr}^1 = E_{tr}$ throughout the section. Using the findings from Section 3.2, the single-device problem can be formally expressed as to *minimize*:

$$\begin{aligned}E(f) = \ &\textstyle\sum_{j=0}^{n-1}(af^3 + P_a)\frac{c_j}{f}(1-F(b_j)) \\ &+F((d-B)f)E_{tr} \\ &+\textstyle\sum_{Z=d-B}^{d}P(x=Z)P_a(d-x)\end{aligned} \tag{11}$$

Subject to:

$$\frac{wcc}{f} \leq d \tag{12}$$

$$f_{min} \leq f \leq f_{max} \tag{13}$$

Above, (12) encodes the deadline constraint while (13) indicates the feasible frequency ranges supported by the system. A non-trivial difficulty with the above optimization problem is that the unknown f appears as a parameter in the cumulative distribution function which may be of arbitrary form, making a closed form solution unlikely. However, notice that $F((d-B)f)$ can have only one of the $n+1$ distinct values that correspond to $F(b_i)$ $(0 \leq i \leq n)$ in the optimal solution. This property suggests an iterative approach: our original problem will be divided into to $n+1$ sub-problems by letting $F((d-B)f) = F(b_0), ..., F(b_n)$ successively. Each sub-problem will be attacked by assuming

that $F((d-B)f) = F(b_i)$ and the expected energy consumption corresponding to that sub-problem will be recorded. Finally, the frequency that leads to the minimal energy consumption in any of the sub-problems will be selected as the global optimal.

In the following, we focus on the solution of these subproblems. Notice that when $F((d-B)f) = F(b_i)$ $(0 \leq i \leq n)$, the expression (10) can be readily re-written as a function of $F(b_i)$. Further, the properties of the histogram-based approach and simple algebraic manipulation show that, when $F((d-B)f) = F(b_i)$, (9) is equivalent to:

$$\sum_{j=i}^{n-1}(F(b_{j+1}) - F(b_j))P_a(d - \frac{b_{j+1}}{f})$$

Hence, the i^{th} sub-problem can be formally defined as to *minimize*:

$$E^i(f) = \sum_{j=0}^{n-1}(af^3 + P_a)\frac{c_j}{f}(1 - F(b_j)) + F(b_i)E_{tr} + \sum_{j=i}^{n-1}(F(b_{j+1}) - F(b_j))P_a(d - \frac{b_{j+1}}{f}) \quad (14)$$

Subject to:

$$\frac{wcc}{f} \leq d \quad (15)$$

$$\frac{b_{i-1}}{f} < d - B \leq \frac{b_i}{f} \quad (16)$$

$$f_{min} \leq f \leq f_{max} \quad (17)$$

where[1] the new additional constraint (16) is the sufficient and necessary condition to enforce $F((d-B)f) = F(b_i)$. Let $f_{low,i} = max(f_{min}, \frac{wcc}{d}, \frac{b_{i-1}}{d-B}+\epsilon)$ and $f_{up,i} = min(f_{max}, \frac{b_i}{d-B})$. Notice that $f_{low,i}$ corresponds to the lower bound[2] on the feasible frequency range for the problem while $f_{up,i}$ is the upper bound to the frequency range. It is obvious that if $f_{up,i} < f_{low,i}$, the feasible region for this subproblem is empty. On the other hand, the case of $f_{up,i} \geq f_{low,i}$ can be solved precisely. Observe that $E^i(f)$ is a strictly convex function. Therefore, the frequency f_i that minimizes $E^i(f)$ without considering constraints (15), (16) and (17) can be found by setting its derivative to zero:

$$f_i = \left(\frac{P_a(\sum_{j=0}^{n-1} c_j(1 - F(b_j)) - \sum_{j=i}^{n-1}(F(b_{j+1}) - F(b_j))b_{j+1})}{2a\sum_{j=0}^{n-1} c_j(1 - F(b_j))}\right)^{\frac{1}{3}} \quad (18)$$

The convex nature of $E^i(f)$ justifies the following two basic properties for any $\Delta > 0$:

PROPERTY 1. $\forall f, f > f_i, E^i(f_i) \leq E^i(f) \leq E^i(f + \Delta)$

PROPERTY 2. $\forall f, f < f_i, E^i(f_i) \leq E^i(f) \leq E^i(f - \Delta)$

Based on these properties, one can obtain the following:

THEOREM 1. *The optimal frequency for the i^{th} subproblem is equal to $f_i^* = max\{f_{low,i}, min\{f_{up,i}, f_i\}\}$, whenever $f_{up,i} \geq f_{low,i}$.*

Once we evaluate the optimal solutions to all $n+1$ subproblems (each requiring $O(n)$ time), one can easily find the global optimal (in time $O(n^2)$). However, the procedure can be further speeded up by observing that once f_i is computed, f_{i+1} can be easily evaluated, since $f_{i+1} = f_i - \frac{(F(b_{i+2}) - F(b_{i+1}))b_{i+2}}{2a\sum_{j=0}^{n-1} c_j(1-F(b_j))}$. The denominator in this expression is indeed a constant (independent of i) and can be computed just once when evaluating f_0. Thus, one can easily design an algorithm that computes the f_i values *bottom-up*, by keeping track of the best candidate seen so far across the $n+1$ iterations. This optimization will help reduce the complexity to just $O(n \log n)$, where the dominant term will be due to the process of sorting the b_i and $F(b_i)$ values in increasing order. If they are already sorted in the input, the overall complexity is just $O(n)$.

5. MULTIPLE-DEVICE MODEL

In this section, we address the general problem where the application uses m devices $D_1, \ldots, D_m$ (with corresponding break-even times $B_1, \ldots, B_m$). We assume the device indices reflect the ordering of the break-even times, as shown in Figure 8. With multiple devices, the problem gains a new dimension. By adjusting the processing frequency f, the application's execution time within a frame can be controlled through DVS; but this has a direct impact on the applicability of DPM. Specifically, when the application completes at time t such that $(d - B_i) < t < (d - B_{i-1})$, the devices $D_1, \ldots, D_{i-1}$ can be transitioned to the *sleep* state during the slack period. However, the devices $D_i, \ldots, D_m$ should remain in *active* state throughout the frame. Obviously, the probabilistic behavior of the workload adds another non-trivial complexity layer to the problem.

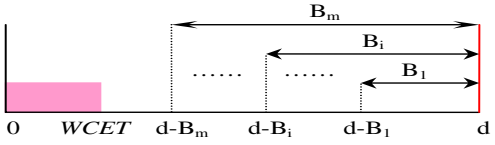

Figure 8: An application using m devices and break-even times

The problem can be formally stated as to minimize the expected energy consumption $E_m(f)$ given by (6), where the execution period, slack period, and transition energy figures are given by (8), (9), and (10), respectively. Again, the optimal frequency f should satisfy the deadline constraint $\frac{wcc}{f} \leq d$ and the feasible frequency range constraint $f_{min} \leq f \leq f_{max}$.

Similar to the case of single-device, the general problem can be seen to give rise to multiple sub-problems by assuming that the product $F((d - B_i)f)$ is equal to a specific $F(b_j)$ for each device D_i. Specifically, in the optimal solution, $F((d - B_i)f)$ $(1 \leq i \leq m)$ can be equal to $F(b_{a_i})$ $(0 \leq a_i \leq n)$. In other words, a_i will be equal to the *index* of the cycle group (given in the histogram) that the application is assumed to be executing at time $t = d - B_i$, in the subproblem that corresponds to each tentative optimal solution. Since the cycle group j (from b_j to b_{j+1}) should be executed *before* the cycle group $j+1$ (from b_{j+1} to b_{j+2}), one can infer that $a_i \geq a_{i+1}$ in a subproblem. Therefore, each sub-problem will be defined by a unique ordered sequence of $a_1, \ldots, a_m$.

Following a reasoning similar to the case of the single-device, one can obtain the corresponding formulation of the

[1]For mathematical convenience, we define $\sum_{j=n}^{n-1} x = 0$ for any value of x and we assume $\frac{b_{i-1}}{f} < d - B$ automatically holds when $i = 0$.

[2]In the definition of $f_{low,i}$, ϵ is a small number arbitrarily close to 0, which is introduced to enforce the constraint $\frac{b_{i-1}}{f} < d - B_i$ with the strict inequality requirement.

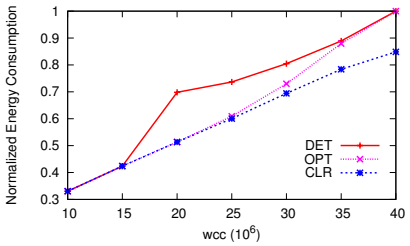

Figure 5: Normalized expected energy as a function of wcc ($P = 44ms$)

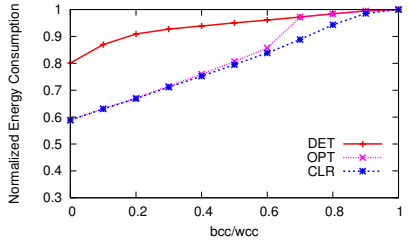

Figure 6: Normalized expected energy as a function of $\frac{bcc}{wcc}$ ($P = 44ms$ and $wcc = 20 \times 10^6$)

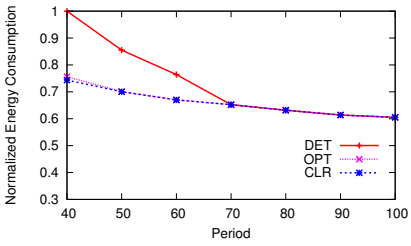

Figure 7: Normalized Expected Energy as a function of the application's period ($U = 0.5$)

subproblem for a given sequence $a_1, \ldots, a_m$ as to *minimize*:

$$E_{a_1,..,a_m}(f) = \sum_{i=1}^{m}\sum_{j=a_i}^{n-1}(F(b_{j+1}) - F(b_j))P_a^i(d - \tfrac{b_{j+1}}{f}) + \sum_{j=0}^{n-1}(af^3 + \sum_{i=1}^{m}P_a^i)\tfrac{c_j}{f}(1 - F(b_j)) + \sum_{i=1}^{m}F(b_{a_i})E_{tr}^i \quad (19)$$

Subject to:

$$\frac{wcc}{f} \leq d \quad (20)$$

$$\frac{b_{a_i-1}}{f} < d - B_i \leq \frac{b_{a_i}}{f} \quad (1 \leq i \leq m) \quad (21)$$

$$f_{min} \leq f \leq f_{max} \quad (22)$$

where $E_{a_1,..,a_m}(f)$ is the expected energy consumption with the specific values of $a_1, .., a_m$, and the new additional constraints (21) are the sufficient and necessary conditions to enforce $F((d - B_i)f) = F(b_{a_i})$. For a given sub-problem, we can define $f'_{low} = max(f_{min}, \frac{wcc}{d}, \frac{b_{a_i-1}}{d-B_i} + \epsilon(\forall i))$ and $f'_{up} = min(f_{max}, \frac{b_{a_i}}{d-B_i}(\forall i))$. By following steps similar to those in Section 4, we can conclude that if $f'_{up} < f'_{low}$, the corresponding subproblem does not have a solution. For the case where $f'_{up} \geq f'_{low}$, the solution f'_a is given by:

$$f'_a = (\frac{\alpha - \gamma}{\beta})^{\frac{1}{3}} \quad (23)$$

where $\alpha = (\sum_{j=0}^{n-1}\sum_{i=1}^{m}P_a^i c_j(1 - F(b_j))$,
$\beta = 2a\sum_{j=0}^{n-1}c_j(1 - F(b_j))$, and,
$\gamma = \sum_{i=1}^{m}\sum_{j=a_i}^{n-1}P_a^i(F(b_{j+1}) - F(b_j))b_{j+1})$.

THEOREM 2. *The subproblem of minimizing $E_{a_1,..,a_m}(f)$ admits an optimal solution $f_a^* = max\{f'_{low}, min\{f'_{up}, f'_a\}\}$, whenever $f'_{up} \geq f'_{low}$.*

As a quick inspection of the formula for f'_a reveals, it takes only $O(mn)$ steps to solve each sub-problem. However, the apparent computational difficulty comes from the large number of subproblems. In fact, the total number of sub-problems is given by the number of multisets of cardinality m, with elements taken from a finite set of cardinality n, which is equal to $\frac{(n+m-1)!}{m!(n-1)!}$. This gives $O(n^m)$ potential subproblems, which is prohibitively large.

Nevertheless, it is possible to develop a faster solution by observing that *most* of the subproblems have indeed empty feasible regions and that it is necessary and sufficient to consider only at most $m(n + 1) + 3$ subproblems, each of which is *uniquely defined by a separate combination of the cycle group index j and device index i*. The full details of this faster algorithm of overall complexity $O(mn \log mn)$ with the accompanying proofs are presented in the Appendix.

6. EXPERIMENTAL EVALUATION

To evaluate the performance gains yielded by our solution, we constructed a discrete-event simulator in C. In our simulator, we implemented the following three schemes:

- The optimal scheme, denoted by *OPT* and developed in this paper, which minimizes the *expected* system energy based on the application's probabilistic workload information, by integrating DVS and DPM in an optimal way.
- The scheme *DET* (proposed in [5]), which minimizes the system energy consumption again by considering both DVS and DPM features, but by assuming a deterministic workload (equal to wcc).
- The *clairvoyant scheme (CLR)*, that computes the optimal frequency by using the knowledge about the actual workload (number of cycles) of the application in advance. While it is not a practical scheme (since no algorithm can know the exact workload in advance), *CLR* is included in our comparison to assess the extent at which our algorithm's performance approaches absolute ideal bounds by exploiting the probabilistic information.

To be consistent with the experimental settings in [5], we performed our experiments by using the actual device specifications from [3] and the CPU power consumption is modeled after the Intel XScale specifications [17]. The application uses three devices during its execution: IBM Microdrive ($B = 24ms$) , Realtek Ethernet Chip ($B = 20ms$) and Simple Tech Flash Card ($B = 4ms$). The frame length P varied from $40ms$ to $100ms$; the application's execution cycles are generated using normal distribution with the mean $\frac{(bcc+wcc)}{2}$ and standard derivation $\frac{(wcc-bcc)}{12}$. This guarantees that 99.7% of the cycles fall in the range $[bcc, wcc]$ and cycles values outside this range are not considered. The $[bcc, wcc]$ range is divided into $n = 100$ cycle groups of equal size. The maximum CPU frequency is assumed to be $1GHz$

We first evaluate the effect of the worst-case execution time on the expected energy consumption with the frame length (period) $= 44ms$ as in [5], by changing wcc from 10×10^6 to 40×10^6 cycles with $bcc = 0$ (Figure 5). Notice that $wcc = 40 \times 10^6$ corresponds to an almost fully

utilized system. In Figure 5, all energy consumptions are normalized to that of OPT when $wcc = 40 \times 10^6$. As we can see, the performance of our solution OPT is very close to that of the clairvoyant scheme CLR, especially when $wcc \leq 30 \times 10^6$. When $wcc \leq 15 \times 10^6$, DET can achieve the same performance as CLR and OPT. The reason is that, in the case of very low workload, the application can be executed with the energy-efficient frequency (f_{ee}) [5] while still meeting the deadline and leaving enough idle time to turn off all the devices. However, when wcc increases beyond 20×10^6, DET chooses the frequency ($f = U$) to minimize the system energy under the wcc case, while OPT can still use f_{ee} to achieve better expected energy savings by considering the probabilistic workload information. But once $wcc > 25 \times 10^6$, OPT has to also increase the frequency to enforce longer idle intervals for device state transitions. In fact, when $wcc = 35 \times 10^6$, considering the interaction of DVS and DPM, OPT is forced to use the frequency $f = U$ just like DET. These results suggest that OPT is able to achieve performance levels that are practically indistinguishable from the clairvoyant algorithm, except when the worst-case workload is very high.

Figure 6 shows the relative performance of three schemes as a function of the actual workload variability. Specifically, we consider an application with period$= 44ms$, $wcc = 20 \times 10^6$, and we vary the ratio $\frac{bcc}{wcc}$. Clearly, the lower this ratio, the more the actual workload deviates from the worst-case. The energy values are normalized with respect to OPT when $bcc = wcc$. We observe that the performance of OPT is almost identical to that of CLR, when $\frac{bcc}{wcc} \leq 0.6$, exhibiting performance gains of around 35% over DET. However, when the ratio exceeds 0.7, OPT is forced to use the same processing frequency as that of DET. This is primarily due to the fact that large bcc values do not leave much opportunities to optimistically increase the frequency to create long device idle intervals. In fact, when $bcc = wcc$, all three schemes achieve exactly the same performance.

In Figure 7, we study the impact of varying the application period on the energy consumption by setting $U = \frac{wcc}{P} = 0.5$. The energy values are normalized with respect to that of DET when $P = 40ms$. As we expect, the energy consumption decreases as we increase the period, and increasing slack amounts enable more device state transitions. The performance of OPT is in fact indistinguishable from that of the clairvoyant scheme CLR. OPT provides energy savings of up to 35% over DET, though the savings tend to decrease with increasing period. This is because, increasing the period while keeping the device break-even times constant enables also DET to put the devices to sleep states, even when planning according to worst-case workload scenarios.

7. CONCLUSIONS

In this work, we considered the problem of optimally integrating DVS and DPM policies for real-time embedded applications characterized by probabilistic workload profiles and we presented algorithms to minimize the expected system-wide energy. Our solution is based on the precise characterization of the expected energy components, using DVS and DPM properties. First, we solved the problem for an application using a single device and then generalized it to multiple devices. By observing the special characteristics of the optimal solution for the multiple-device case, we also suggested a faster algorithm which significantly reduces the search space. Our experimental evaluation shows that our algorithm can achieve significant energy savings compared to the previous algorithm proposed in [5] for deterministic workloads and performs comparably even to a clairvoyant optimal scheduler that knows the exact workload in advance. To the best of our knowledge, this is the first solution to the system-wide energy management problem based on the optimal combination of DVS and DPM for probabilistic workloads.

8. REFERENCES

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

[2] H. Aydin, R. Melhem, D. Mossé, and P. Mejia-Alvarez. Power-aware scheduling for periodic real-time tasks. *IEEE Transactions on Computers*, 53(5):584 – 600, 2004.

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

[4] H. 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), 2007.

[5] V. Devadas and H. Aydin. On the interplay of dynamic voltage scaling and dynamic power management in real-time embedded applications. In *Proc. International Conference on Embedded Software (EMSOFT)*, 2008.

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

[7] R. Jejurikar and R. Gupta. Dynamic voltage scaling for systemwide energy minimization in real-time embedded systems. In *Proc. International Symposium on Low Power Electronics and Design (ISLPED)*, 2004.

[8] M. Kim and S. Ha. Hybrid run-time power management technique for realtime embedded system with voltage scalable processor. In *Proc. Conference on Languages, Compilers, and Tools for Embedded Systems (LCTES)*, 2001.

[9] J. Lorch and A. Smith. Improving dynamic voltage scaling algorithms with pace. In *Proc. SIGMETRICS*, 2001.

[10] P. Pillai and K. G. Shin. Real-time dynamic voltage scaling for lowpower embedded operating systems. In *Proc. ACM Symposium on Operating Systems Principles (SOSP)*, 2001.

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

[12] S. Saewong and R. Rajkumar. Practical voltage scaling for fixed priority rt-systems. In *Proc. IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2003.

[13] T. Simuinic, L. Benini, A. Acquaviva, P. Glynn, and G. D. Micheli. Dynamic voltage scaling and power management for portable systems. In *Proc. Design Automation Conference (DAC)*, 2001.

[14] V. Swaminathan and K. Chakrabarty. Energy-conscious, deterministic i/o device scheduling in hard real-time systems. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 22(7), 2003.

[15] V. Swaminathan and K. Chakrabarty. Pruning-based, energy-optimal, deterministic i/o device scheduling for hard real-time systems. *ACM Trans. on Embedded Computation Systems*, 4(1), 2005.

[16] V. Swaminathan, K. Chakrabarty, and S. S. Iyengar. Dynamic i/o power management for hard real-time systems. In *Proc. International Symposium on Hardware/software Codesign (CODES)*, 2001.

[17] R. Xu, D. Mosse, and R. Melhem. Minimizing expected

energy consumption in real-time systems through dynamic voltage scaling. *ACM Trans. for Embedded Computing Systems*, 25(4), 2007.

[18] W. Yuan and K. Nahrstedt. Energy-efficient soft real-time cpu scheduling for mobile multimedia systems. In *Proc. ACM Symposium on Operating Systems Principles (SOSP)*, 2003.

[19] Y. Zhang, Z. Lu, J. Lach, K. Skadron, and M. Stan. Optimal procrastinating voltage scheduling for hard real-time systems. In *Proc. Design Automation Conference (DAC)*, 2005.

[20] J. Zhou, C. Chakrabarti, and N. Chang. Energy management of dvs-dpm enabled embedded systems powered by fuel cellbattery hybrid source. In *Proc. International Symposium on Low Power Electronics and Design (ISLPED)*, 2007.

APPENDIX

Fast Algorithm for the Multiple-Device Model

In this appendix, we present the details of our fast solution to the expected energy minimization problem for multiple devices. First, we note that for each sub-problem that targets minimizing the expected energy consumption $E_h(f) = E_{a_1,..,a_m}(f)$ corresponding to a specific a_1,a_2,..,a_i,..a_m combination, the frequency range constraints (20)-(22) can be captured in a single constraint of the form:

$$x_h \leq f \leq y_h \tag{24}$$

where $x_h = max(f_{min}, U = \frac{wcc}{d}, \frac{b_{a_i-1}}{d-B_i} + \epsilon(\forall i))$, and $y_h = min(f_{max}, \frac{b_{a_i}}{d-B_i}(\forall i))$. Observe that the sub-problem has a non-empty feasible region if and only if $x_h \leq y_h$. We construct a sequence S, which contains the following $(n+1)m$ points: $\frac{b_0}{d-B_1}$, $\frac{b_1}{d-B_1}$,... $\frac{b_n}{d-B_1}$,... , $\frac{b_0}{d-B_m}$, $\frac{b_1}{d-B_m}$,... $\frac{b_n}{d-B_m}$. This sequence can be sorted in time $O(mn \log mn)$ in non-decreasing order. Observe that these points correspond to possible frequency ranges defined through the constraints (21) over all possible sub-problems. Then, the sequence S is updated by eliminating the points that correspond to the infeasible frequency ranges (i.e. those below $max(f_{min}, U)$ or above f_{max}), by removing the duplicate points, and finally by inserting $max(f_{min}, U)$ and f_{max} in proper position. After these operations, the points $z_0, z_1, \ldots$ in sequence S are in total order and their cardinality is at most $m\,(n+1)+2$. The following holds:

LEMMA 1. *For each sub-problem with non-empty feasible region, x_h and y_h are obtained from two consecutive points z_i and z_{i+1} in S, when $x_h < y_h$.*

PROOF. Based on its definition, x_h can be expressed as $x_h = z_i$ if $z_i = max\{f_{min}, U\}$, otherwise, $x_h = z_i + \epsilon$. Further, $y_h = z_j \; \exists j$. Assume that the lemma does not hold. In that case, j must be greater than $i+1$.

In this case, there exists a point $z \in S$ such that $z_i < z < z_j$. Hence, $x_h < z < y_h$ should also hold. Since $z \in S$, it can be expressed as $\frac{b_j}{d-B_k}$ for some existing j and k. Since $x_h = max(f_{min}, \frac{wcc}{d} = U, \frac{b_{a_i-1}}{d-B_i} + \epsilon(\forall i)) < z$, and then $\frac{b_{a_k-1}}{d-B_k} < \frac{b_j}{d-B_k}$ holds and we conclude that $b_{a_k-1} < b_j$. Because the value of b_i $(\forall i)$ is increasing with increasing index i, then $j > a_k - 1$ holds. Similarly, since $z < y_h = min(\; f_{max}, \; \frac{b_{a_i}}{d-B_i}(\forall i))$, then $\frac{b_j}{d-B_k} < \frac{b_{a_k}}{d-B_k}$ follows and we obtain $j < a_k$. But this implies $a_k - 1 < j < a_k$, contradicting with the fact that j is an integer. We reach a contradiction, showing the validity of the lemma. □

Lemma 1 implies that the frequency range $[x_h, y_h]$ is of the following form whenever $x_h < y_h$:

$$x_h = \begin{cases} z_k, & \text{if } z_k = max\{f_{min}, U\} \\ z_k + \epsilon, & \text{otherwise.} \end{cases} \tag{25}$$

$$y_h = z_{k+1} \tag{26}$$

where z_k is a point in the ordered sequence S.

LEMMA 2. *Each given frequency range $[x_h, y_h]$ $x_h \leq y_h$ uniquely determines an original subproblem that minimizes $E_{a_1,..,a_m}(f)$ where $a_i = j+1$ $(1 \leq i \leq m)$ and j satisfies both $b_j < (d - B_i)z_k$ and $b_{j+1} \geq (d - B_i)z_{k+1}$.*

PROOF. To prove this lemma, we will show that, given $x_h < y_h$, one can uniquely compute a_i $(i = 1, .., m)$.

Based on the construction of S, we know that there must exist w $(0 \leq w < n)$ such that $\frac{b_w}{d-B_i} < z_k$ (i.e. $b_w \leq (d - B_i)\; x_h$). In order to determine $a_i = j + 1$, we must find suitable j satisfying $b_j < (d - B_i)\; x_h$ and $(d - B_i)z_{k+1} \leq b_{j+1}$ at the same time. We claim that $j = max\{w|\frac{b_w}{d-B_i} < z_k\}$ is the unique solution. In fact, if $j < max\{w|\frac{b_w}{d-B_i} < z_k\}$, then $\frac{b_{j+1}}{d-B_i} \leq max\{w|\frac{b_w}{d-B_i} < z_k\} < z_k < z_{k+1}$, which is not possible. Similarly, if $j > max\{w|\frac{b_w}{d-B_i} < z_k\}$, $\frac{b_j}{d-B_i} > z_k$, which contradicts the assumption. Hence, j should be exactly $max\{w|\frac{b_w}{d-B_i} < z_k\}$. It is obvious that j exists and uniquely defined; consequently, $a_i = j + 1$ is uniquely determined. □

From the above discussion, we see that even in the worst case, we only need to consider $m(n+1)+1$ possible frequency ranges, which is constructed by consecutive points in S. Further, as Lemma 2 shows, each of these ranges uniquely determines a sub-problem with non-empty feasible region. As a result, this enables us to eliminate a large number of unnecessary subproblems.

Now we give a general solution for the lth problem $(1 \leq l \leq m(n+1)+1)$ with the frequency range $x_l < y_l$, where x_l and y_l are defined as in (25) and (26). The lth problem can be formally expressed as to *minimize*:

$$\begin{aligned} E_{m,l}(f) = \; & \textstyle\sum_{i=1}^{m} \sum_{j=a_{l,i}}^{n-1} (F(b_{j+1}) - F(b_j)) P_a^i (d - \frac{b_{j+1}}{f}) \\ & + \textstyle\sum_{j=0}^{n-1} (af^3 + \sum_{i=1}^{m} P_a^i) \frac{c_j}{f} (1 - F(b_j)) \\ & + \textstyle\sum_{i=1}^{m} F(b_{a_{l,i}}) E_{tr}^{a_{l,i}} \end{aligned} \tag{27}$$

Subject to:

$$x_l \leq f \leq y_l \tag{28}$$

where $E_{m,l}(f)$ is the expected energy with m devices for the lth problem and each $a_{l,i}$ is determined by the frequency range $[x_l, y_l]$ (as stated in Lemma 2).

Using a method similar to that in Section 4 and 5, we can state:

THEOREM 3. *If there exists a feasible solution to the subproblem of minimizing $E_{m,l}(f)$, that feasible solution is equal to $f^*_{m,l} = max\{x_l, \; min\{y_l, f_{m,l}\}\}$, where $f_{m,l} = (\frac{\alpha - \sigma}{\beta})^{\frac{1}{3}}$,*
$\alpha = \sum_{j=0}^{n-1} \sum_{i=1}^{m} P_a^i c_j (1 - F(b_j))$,
$\beta = 2a \sum_{j=0}^{n-1} c_j (1 - F(b_j))$, and,
$\sigma = \sum_{i=1}^{m} \sum_{j=a_{l,i}}^{n-1} P_a^i (F(b_{j+1}) - F(b_j)) b_{j+1}$.

Also observe that the relationship between $f_{m,l}$ and $f_{m,l+1}$ is similar to that between f_i and f_{i+1} in Section 4. Therefore, if we store the constant values of α and β, then based on the result of $f_{m,l}$, we just need at most m steps to compute $f_{m,l+1}$ (since $a_{l+1,i}$ is increased by at most 1 compared to $a_{l,i}$). Hence, starting at $f_{m,1}$, we need at most $O(mn)$ steps to directly compute all l subproblems. Note that, for completeness, one also needs to compute the expected energy numbers for two cases where $x_l = y_l$ and compare against the best energy figure obtained through the above process. In fact, this is the case only for two boundary values $max(f_{min}, \frac{wcc}{d} = U)$ and f_{max}. The complexity of the entire procedure is $O(mn)$. However, one needs to construct the set S and order the points therein, implying an overall complexity of $O(mn \log(mn))$.