Profile-based Optimal Intra-task Voltage Scheduling for Hard Real-Time Applications

Jaewon Seo
CS, KAIST
jwseo@jupiter.kaist.ac.kr

Taewhan Kim
EE, Seoul National Univ.
tkim@ssl.snu.ac.kr

Ki-Seok Chung
CIC, Hanyang University
kchung@hanyang.ac.kr

ABSTRACT

This paper presents a set of comprehensive techniques for the intra-task voltage scheduling problem to reduce energy consumption in hard real-time tasks of embedded systems. Based on the execution profile of the task, a voltage scheduling technique that optimally determines the operating voltages to individual basic blocks in the task is proposed. The obtained voltage schedule guarantees minimum average energy consumption. The proposed technique is then extended to solve practical issues regarding transition overheads, which are totally or partially ignored in the existing approaches. Finally, a technique involving a novel extension of our optimal scheduler is proposed to solve the scheduling problem in a discretely variable voltage environment. In summary, it is confirmed from experiments that the proposed optimal scheduling technique reduces energy consumption by 20.2% over that of one of the state-of-the-art schedulers [11] and, further, the extended technique in a discrete voltage environment reduces energy consumption by 45.3% on average.

Categories and Subject Descriptors: C.3 [Special-purpose and application-based systems]: Real-time and embedded systems

General Terms: Algorithms, Design

Keywords: low energy design, DVS, intra-task voltage scheduling

1. INTRODUCTION

Over the past several years, minimizing energy consumption has become an important concern in system design. Since the energy consumption in CMOS circuits has a quadratic dependency on the supply voltage, lowering the supply voltage is one of the most effective techniques for reducing the energy consumption. Recently, a dynamic voltage scaling (DVS) framework, which involves a dynamic adjustment of the supply voltage (and corresponding operating speed), has become a major research area. The supply voltage scaling usually results in considerable energy savings, but increases execution time. Thus, minimizing efforts should be made with care under timing constraints.

Several DVS techniques for both soft and hard real-time tasks running on DVS processors have been proposed. Since many of the applications running on energy-budgeted systems have tight temporal constraints, in this paper, we focus on the voltage scheduling problem for hard real-time tasks.

Many previous works (e.g., [1–5]) have focused on real-time systems with multiple tasks where their primary concern was to assign a proper voltage to each task dynamically while satisfying the task's deadline. In these techniques, the determination of supply voltage is carried out on a task-by-task basis (i.e., *inter-task voltage scheduling*), and the supply voltage assigned to a task is unchanged during the execution of the task.

Recently, several studies (e.g., [6–12]) have proposed an *intra-task voltage scheduling* technique, which adjusts the supply voltage in a task to further reduce the energy consumption. Lee *et al.* [6] proposed a scheme which partitions a task into several time slots and performs run-time supply voltage control on a time slot-by-time slot basis. Mossé *et al.* [7] described techniques that change the processor speed based on the natural slack of the system and time reclaimed from code sections that finish before their deadlines. Hsu *et al.* [8] suggested a compilation strategy that identifies scaling opportunities without significant overall performance penalty. Gruian [9] proposed a stochastic voltage scheduler using the probability distribution of the execution pattern for a task. However, it is essentially based on the idea of scheduling (i.e., scaling) voltage at each cycle, which is practically impossible due to excessive voltage transition overhead. Shin *et al.* [10] proposed a *reference path-based intra-task scheduling*, in which the worst-case execution path is selected as a reference. One limitation of this approach is that the voltage adjustment is performed purely based on the worst-case execution, and never takes into account the degree of the likelihood of the possible execution paths, consequently, missing an opportunity for further reducing the average energy consumption. To overcome this limitation, an average-case execution path based scheduling has been proposed [11]; however, it does not always achieve minimum average energy consumption. Azevedo *et al.* [12] proposed heuristic techniques using program checkpoints at which the processor speed and voltage are re-calculated. A major limitation of this method is that it does not take into account the power and time overheads for the extra run-time voltage scheduling operations as well as the voltage transition overheads.

2. ENERGY MODEL

It is known [10] that the amount of energy consumption E during the execution of a task is expressed as $E \propto {V_{\mathrm{DD}}}^2 \times n_{\mathrm{total}}$ where V_{DD} is the supply voltage and n_{total} is the total number of instruction cycles executed. The relationship between clock frequency and voltage in CMOS circuits is $f_{\mathrm{CLK}} \propto (V_{\mathrm{DD}} - V_{\mathrm{T}})^{\alpha}/V_{\mathrm{DD}} \approx V_{\mathrm{DD}}$ where V_{T} is the threshold voltage and α $(1.4 \leq \alpha \leq 2)$ is the velocity saturation index. Because of the one-to-one correspondence between clock frequency f_{CLK} and supply voltage V_{DD}, we use processor voltage and speed (i.e., clock frequency) interchangeably throughout the paper. In addition, we assume that the speed is linearly proportional to the voltage as in [1, 5, 8, 9, 17].

Since many tasks exhibit different behaviors depending on input data, thus generating different numbers of execution cycles, the simple energy equation may not be applied directly. Thus, we extend the energy model so that it can handle general cases. One way to deal

DAC'04, June 7–11, 2004, San Diego, California, USA.

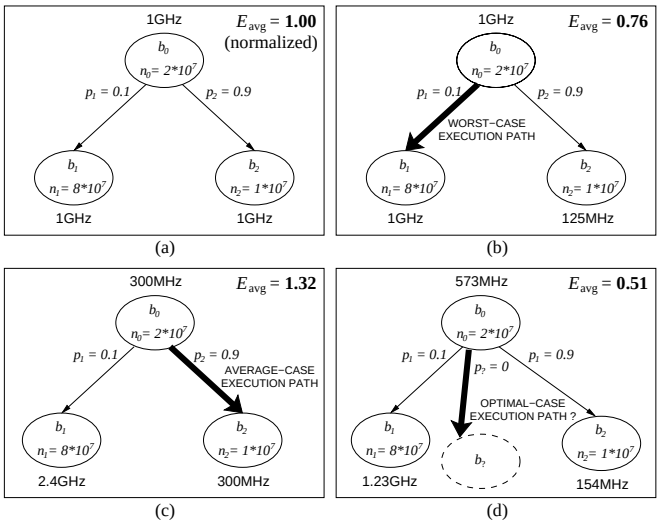

Figure 1: (a) Schedule with a single high speed; (b) Schedule by the RWEP-based technique [10]; (c) Schedule by the RAEP-based technique [11]; (d) Schedule by our ROEP-based technique.

with the non-uniform execution path lengths is to consider an average (i.e., expected) energy consumption, which can be expressed as

$$E_{\text{avg}} = \sum_{\forall \text{execution path } \pi} \big(P(\pi) \cdot E(\pi)\big) \quad (1)$$

where $P(\pi)$ is the probability that the execution of the task follows path π and $E(\pi)$ is the energy consumption for π. The fact that many tasks in real-time environments are executed repeatedly provides the rationale for the use of the model in *Eq.* (1).

3. OPTIMAL VOLTAGE SCHEDULING

3.1 Motivating Example

As an example illustrating how the conventional intra-task voltage scheduling techniques work and what their limitations are, consider a simple hard real-time task τ_{simple} with a deadline of 100 ms, and three basic blocks b_0, b_1, and b_2. The control flow graph (CFG) of τ_{simple} is shown in Figure 1(a), in which each node represents a basic block of τ_{simple}, and each directed edge represents the control dependency between basic blocks. The number inside each node indicates the number of execution cycles of the corresponding basic block and the number assigned to each edge indicates the probability that the control flow follows the edge. The problem then is to determine at what speed each basic block should be executed in order to minimize energy consumption. Note that the granularity is confined to basic blocks, because by definition basic blocks have single execution paths, and thus a single speed can be used for each basic block without losing optimality [2].

The first technique, which is the most commonly used, is to execute the entire task at a single high speed. In this strategy, if the task is completed prior to the deadline, which means there remains *slack time*, the system enters a power-down mode. To minimize the energy consumption while meeting the deadline, the speed for τ_{simple} should be set tightly to $\frac{\text{length of the critical path}}{\text{remaining time to deadline}} = \frac{(2+8)10^7 \text{ cycles}}{100 \cdot 10^{-3} \text{ sec}}$ = 1 GHz. Figure 1(a) shows the speed schedule produced by this technique. Assuming that the energy consumption in the power-down mode is 0, the average energy consumption E_{avg} of *Eq.* (1) is $0.1 \cdot E(b_0b_1) + 0.9 \cdot E(b_0b_2) = 0.1 \cdot (1\,\text{GHz})^2((2+8) \cdot 10^7 \text{ cycles}) + 0.9 \cdot (1\,\text{GHz})^2((2+1) \cdot 10^7 \text{ cycles}) = 3.7$ units of energy[1]. Note that for execution paths other than $\pi_{\text{worst}} = b_0b_1$, the task always finishes before the deadline.

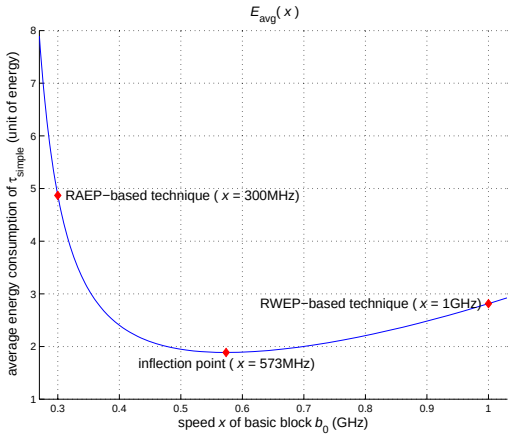

Figure 2: A graph showing the relation between the energy consumption and initial speed of b_0.

Based on the fact that slack time is undesirable in terms of energy efficiency, the second technique [10] sets the speed as low as possible, while still meeting the deadline, whenever the execution deviates from a *remaining worst-case execution path* (RWEP). As a result, it finishes the task exactly at the deadline, thereby allowing no slack time. For example, in Figure 1(b), when the execution follows edge (b_0, b_2) other than (b_0, b_1), it sets the speed to $\frac{\text{length of the RWEP (from } b_2)}{\text{remaining time to deadline}} = \frac{1 \cdot 10^7 \text{ cycles}}{80 \cdot 10^{-3} \text{ sec}} = 125$ MHz. Consequently, the average energy consumption using this technique is $0.1 \cdot (1\,\text{GHz})^2((2+8) \cdot 10^7 \text{ cycles}) + 0.9 \cdot \{(1\,\text{GHz})^2(2 \cdot 10^7 \text{ cycles}) + (125\,\text{MHz})^2(1 \cdot 10^7 \text{ cycles})\} = 2.81$, which is 24% less energy than that of the first technique.

A generalized version of the second technique, *reference path based intra-task scheduling*, sets the initial speed assuming that the task execution follows a reference path such as a worst-case execution path. When the current execution takes a path other than the reference path, a new reference path, such as RWEP, is constructed, starting from the current basic block. The speed is then adjusted based on the new path.

Another variation of reference path based intra-task scheduling involves use of the *remaining average-case execution path* (RAEP) as a reference path [11]. An example of which is shown in Figure 1(c), where the speed of the basic block b_0 is set to $\frac{\text{length of the RAEP (from } b_0)}{\text{remaining time to deadline}} = \frac{(2+1) \cdot 10^7 \text{ cycles}}{100 \cdot 10^{-3} \text{ sec}} = 300$ MHz, and the speeds of b_1 and b_2 are computed similarly. The main motivation for using the RAEP rather than the RWEP is to make the common case more energy-efficient. The average energy consumption by this technique is $0.1 \cdot \{(300\,\text{MHz})^2(2 \cdot 10^7 \text{ cycles}) + (2.4\,\text{GHz})^2(8 \cdot 10^7 \text{ cycles})\} + 0.9 \cdot (300\,\text{MHz})^2(10 \cdot 10^7 \text{ cycles}) = 4.87$, which is even larger than that of the RWEP-based technique in Figure 1(b).

Thus far, we have introduced major approaches in the literature on intra-task voltage scheduling, and have shown that the RWEP-based technique is the most energy efficient, at least for task τ_{simple}. However, RWEP-based scheduling does not always guarantee minimum average energy consumption. Let x be the speed of the basic block b_0. Then, the average energy consumption of τ_{simple}, $E_{\text{avg}}(x)$, for the given initial speed x and deadline of 100 ms, is expressed as

[1]For clarity, we ignore any constant terms in the energy equation.

$$E_{\text{avg}}(x) = x^2(2\cdot 10^7) + 0.1\cdot\left(\frac{8\cdot 10^7}{100\cdot 10^{-3} - 2\cdot 10^7/x}\right)^2(8\cdot 10^7) + 0.9\cdot\left(\frac{1\cdot 10^7}{100\cdot 10^{-3} - 2\cdot 10^7/x}\right)^2(1\cdot 10^7). \quad (2)$$

Figure 2 is a graph showing the variation of the quantity of $E_{\text{avg}}(x)$ with the initial speed x of τ_{simple}. From the graph, we can verify that none of the execution paths can be used as a reference path leading to an optimal energy schedule. Note that the optimal value of x is 0.573, as indicated in Figure 2. Figure 1(d) shows an optimal speed schedule of τ_{simple} produced by our optimal scheduling technique, which uses the *optimal-case execution path*[2], $\pi_{\text{optimal}} = b_0 b_?$, as the reference path, resulting in a 33% reduction in energy compared to the RWEP-based scheduling technique. This example clearly shows that the determination of the speed of each basic block according to a particular execution path among the existing paths in the task may not produce an optimal solution. Instead, it strongly suggests that it is necessary to calculate the speed of each basic block in a different manner in order to minimize the average energy consumption.

3.2 The Proposed Technique

The intra-task scheduling problem can be stated more formally as: *Given a task's CFG and the execution profile which offers the probabilities of execution paths, determine the operating speed (or voltage) of each basic block so that the average energy consumption of Eq. (1) for the task is minimized while satisfying the deadline constraint of the task.*

In the following, we describe, during the execution of a task, the procedure of determining optimal speeds of basic blocks on the execution path.

Case 1 - Task with a single branch: For the case where the task includes only a single branch such as τ_{simple}, the problem can be solved easily by finding the inflection point of the energy equation. (See Figure 2 for an example.)

Let n_0, n_1, and n_2 be the numbers of execution cycles of basic blocks b_0, b_1, and b_2 in the CFG structure of Figure 1, respectively, and T be the remaining time to deadline. Then, by generalizing the formulation of *Eq.* (2), we obtain the minimum average energy consumption when we set the speed of b_0 to

$$\frac{n_0 + \sqrt[3]{p_1 {n_1}^3 + p_2 {n_2}^3}}{T} \quad (3)$$

where p_1 and p_2 are the probabilities corresponding to edges (b_0, b_1) and (b_0, b_2), respectively. Note that the quantity of *Eq.* (3) corresponds to the inflection point of the energy equation, and we call the numerator of *Eq.* (3) the length of the *remaining optimal-case execution path* (ROEP) starting from b_0. Although the ROEP does not belong to any of the possible execution paths in the CFG, since our technique, as will be described, works just like the reference path based scheduling technique and the results are optimal, naming of an 'optimal path' can be justified.

Case 2 - Task with more than one branch: For an arbitrary structure of CFG other than that of Case 1, we define the length of the ROEP, δ_i, starting from basic block b_i as

$$\delta_i = \begin{cases} n_i, & \text{if } b_i \text{ has no successor in CFG} \\ n_i + \sqrt[3]{p_j {\delta_j}^3 + p_k {\delta_k}^3}, & \text{otherwise} \end{cases} \quad (4)$$

where n_i is the number of execution cycles of b_i, basic blocks b_j and b_k are immediate successors of b_i, and p_j and p_k are the probabilities that b_j and b_k are conditionally executed after the execution of b_i, respectively.

Our optimal intra-task voltage scheduling technique, which we call an ROEP-based technique, is described as follows: *Before executing the task, we compute the length of the ROEP (i.e., δ's) for each basic block and insert the following instruction code at the beginning of each basic block b_i*

```
change_f_V(δi/remaining_time());
```

where the instruction `change_f_V(`f_{CLK}`)` *changes the current clock frequency (i.e., speed) to f_{CLK} and adjusts the supply voltage accordingly and* `remaining_time()` *returns the remaining time to deadline.*

Note that during the execution of the task, whenever the control flow of execution reaches a basic block b_i, it deviates from the previous ROEP, causing an adjustment of the speed based on the new reference path that is the ROEP starting from b_i.

LEMMA 3.1. Let b_0 be the top (i.e., the beginning) basic block. If we set the speed of each basic block b_i to δ_i/T_i, the average energy consumption becomes ${\delta_0}^3/{T_0}^2$, where T_i is the remaining time to deadline from b_i (i.e., the time returned by the instruction `remaining_time()` in b_i).

PROOF. We use mathematical induction on the number of branches m. For a task that has no branch (m = 0), it is trivial that the above property holds. Assume that the property holds for the task that has $k < m$ branches ($m > 0$). For a task that has m branches, the average energy consumption is expressed as

$$x^2 n_0 + p_1 E_{\text{avg}_1} + p_2 E_{\text{avg}_2} \quad (5)$$

where x is the speed of the top basic block b_0, and E_{avg_1} and E_{avg_2} are the average energies consumed in the execution path from b_1 and b_2, which are two immediate successors of b_0, to the end of the task, respectively.

Since E_{avg_1} and E_{avg_2} can be considered as the energies consumed by tasks which have less than m branches, by the induction hypothesis, *Eq.* (5) becomes

$$x^2 n_0 + p_1 \frac{{\delta_1}^3}{(T_0 - n_0/x)^2} + p_2 \frac{{\delta_2}^3}{(T_0 - n_0/x)^2}. \quad (6)$$

Now, we set the speed x of the basic block b_0 to δ_0/T_0. By definition of the length of the ROEP in *Eq.* (4), we obtain

$$x = \frac{n_0 + \sqrt[3]{p_1 {\delta_1}^3 + p_2 {\delta_2}^3}}{T_0}. \quad (7)$$

Substituting *Eq.* (7) into *Eq.* (6), we have

$$\left(n_0 + \sqrt[3]{p_1 {\delta_1}^3 + p_2 {\delta_2}^3}\right)^3 / {T_0}^2, \quad (8)$$

which is ${\delta_0}^3/{T_0}^2$. □

THEOREM 3.1. The ROEP-based technique produces an optimal intra-task voltage schedule that minimizes the quantity of *Eq.* (1).

PROOF. We use mathematical induction on the number of branches m. For a task that has no branch (m = 0), the proof is trivial. Assume that the theorem holds for the task that has $k < m$ branches ($m > 0$). For a task that has m branches, by the induction hypothesis and Lemma 3.1, a minimum average energy consumption for a given initial speed x is equal to the quantity of *Eq.* (6). To obtain the value of x that minimizes the equation, we differentiate it with respect to x and set the derivative to zero, from which we derive *Eq.* (7), which states $x = \delta_0/T_0$. Therefore, the theorem holds. □

The values of δ_i, $i = 0, 1, \cdots$ can be calculated from the task's CFG in a bottom-up fashion (for example, using a recursive function).

[2] An optimal-case execution path is essentially a 'virtual' path, which will be discussed in the next subsection.

4. PRACTICAL VOLTAGE SCHEDULING

4.1 Deadline Guarantee

In general, the available speeds/voltages are limited to a specific range. Suppose that we are required to use speeds only in the range of $[f_{\min}, f_{\max}]$ and we set the speed of a basic block b_i to f_i. If $f_i < f_{\min}$, we simply use $f_{\min}$ instead of f_i without violating the deadline constraint. However, if $f_i > f_{\max}$, using the speed $f_{\max}$ (or any other speed in the range $[f_{\min}, f_{\max}]$) instead of f_i may lead to an infeasible schedule, thus missing the deadline for an execution path. To tackle this problem, when we set the speed of b_i, we check whether it is lower than the lower bound $f_{\mathrm{LB}}(b_i)$, which is expressed as

$$f_{\mathrm{LB}}(b_i) = \frac{n_i}{T_i - (r_i - n_i)/f_{\max}} \tag{9}$$

where r_i is the length of the RWEP starting from b_i. Note that $f_{\mathrm{LB}}(b_i)$ is the lowest speed for b_i at which the task barely finishes within the deadline under the condition that all succeeding basic blocks are executed at the highest speed. It is obvious that if we set the speed of b_i to $f_i < f_{\mathrm{LB}}(b_i)$, the deadline constraint may not be satisfied. Therefore, in this case we use $f_{\mathrm{LB}}(b_i)$ instead of f_i to guarantee the feasibility of the schedule.

4.2 Supporting Loop-constructs

A *while* statement can be treated as a collection of (possibly an infinite number of) *if* statements. For a basic block b_i in a loop, its δ_i value must be computed for each iteration instance of the basic block. Let I be the maximum number of iterations for a loop; then b_i has I different δ_i values, say δ_{ij} for the j^{th} iteration. This may be a significant overhead since storing many δ_{ij} values may cause additional memory reference operations and also lead to an increased code size. Thus, rather than considering the speed/voltage transitions for *all* iterations, we divide I by a user-defined constant k and allow the transition to occur at *every* k^{th} iteration. Note that in this case b_i keeps only $\lfloor I/k \rfloor$ δ_{ij} values $(\delta_{ik}, \delta_{i2k}, \cdots, \delta_{i\lfloor I/k \rfloor k})$.

4.3 Transition Overheads

In section 3, we have ignored the voltage transition overhead in our voltage scheduling technique. During the execution of a task, three types of transition overhead are encountered for a voltage transition: *transition cycle*, *transition interval*, and *transition energy*.

A *transition cycle* (denoted as Δn) is the number of execution cycles of the transition code itself, which is a constant. For each basic block b_i, the total number of instruction cycles required to execute b_i including the voltage transition becomes $n_i + \Delta n$.

A *transition interval* (denoted as Δt) is the time taken during the voltage transition, which is also a constant [17]. A possible misleading assumption for the variable voltage processors is that the transition interval is proportional to the difference between the starting and ending transition voltages [17]. In many variable voltage processors, the Phase-Locked Loop that is used to set the clock frequency requires a fixed amount of time to lock on a new frequency [18]. This locking time is known to be independent of the source and target frequencies, and is typically larger than the time it takes for the voltage to change. Therefore, it is reasonable to assume that the transition interval is a constant Δt. In addition, we assume that no instructions in the task are executed during the transition interval.

If every execution path of a task contains the same number of branches, say m branches, then every execution encounters the same number of transitions. Thus, integrating the transition interval into our ROEP-based scheduling technique is straightforward, since the problem can be solved by replacing the remaining time to deadline, T, by $(T - m\Delta t)$. Although this case rarely happens, it suggests an idea to approximate an optimal voltage schedule for the general case.

Algorithm 1 (**Step2**) Removal of unnecessary transition codes

```
 1: for all basic block b_i, i > 0 do
 2:    removed[i] ⇐ false;
 3: end for
 4: removed[0] ⇐ true;    /* b_0 is the top basic block */
 5: E_min ⇐ +∞;
 6: repeat
 7:    for all basic block b_i do
 8:       if removed[i] == false then
 9:          gain[i] ⇐ G(b_i) of Eq. (11);
10:       else
11:          gain[i] ⇐ +∞;
12:       end if
13:    end for
14:    Select b_i s.t. gain[i] is minimum;
15:    Remove the transition code of b_i;
16:    removed[i] ⇐ true;
17:    j ⇐ index of immediate predecessor of b_i;
18:    if δ_i − n_i < δ_j then
19:       δ_i ⇐ n_i + δ_j;    /* case1: f_i ⇐ f_j */
20:    else
21:       δ_j ⇐ δ_i − n_i;    /* case2: f_j ⇐ f_i */
22:    end if
23:    E_cur ⇐ E_avg of Eq. (1) for current configuration;
24:    if E_cur < E_min then
25:       E_min ⇐ E_cur;
26:       result ⇐ current configuration;
27:    end if
28: until ∀i, removed[i] == true
29: return result;
```

The following theorem gives the range where the value of the optimal speed of a basic block should lie within.

THEOREM 4.1. Let $m_i^{\min}$ and $m_i^{\max}$ be, respectively, the minimum and maximum numbers of branches in the execution path starting from the basic block b_i. Then, an energy-optimal speed f_{opt} for b_i lies between

$$\frac{\delta_i}{T - m_i^{\min}\Delta t} \le f_{\text{opt}} \le \frac{\delta_i}{T - m_i^{\max}\Delta t}. \tag{10}$$

(We omit the proof due to a lack of space.)

Based on the above theorem, we roughly estimate f_{opt} to be $\delta_i/(T - m_i^{\text{avg}})$, where m_i^{avg} is the average (i.e., expected) number of branches in the execution path starting from b_i.

Finally, a *transition energy* (denoted as ΔE) is the amount of energy consumed during the transition interval by the system. The value of ΔE may vary depending on the starting and ending voltage levels in transition. The problem induced by the transition energy is much more difficult to solve because the generalized model of the transition energy for various systems/processors is hard to obtain and even a simplified version of the problem with the assumption that the transition energy is a constant is non-trivial.

For a voltage transition between two basic blocks and the corresponding transition energy, we can determine if the voltage transition will indeed lead to energy savings, or whether we should simply ignore it. This issue is discussed in detail in the next subsection.

4.4 Discretely Variable Voltage Processors

Current commercial variable voltage processors (e.g., [14–16]) support a limited number of supply voltages. We extend the ROEP-based scheduler to solve the problem of voltage scheduling with a discretely variable voltage processor.

Our extended voltage scheduling technique, which we refer to as ROEP_DISC, is performed in three steps:

Step1 Compute the length of the ROEP, δ_i, for each b_i.

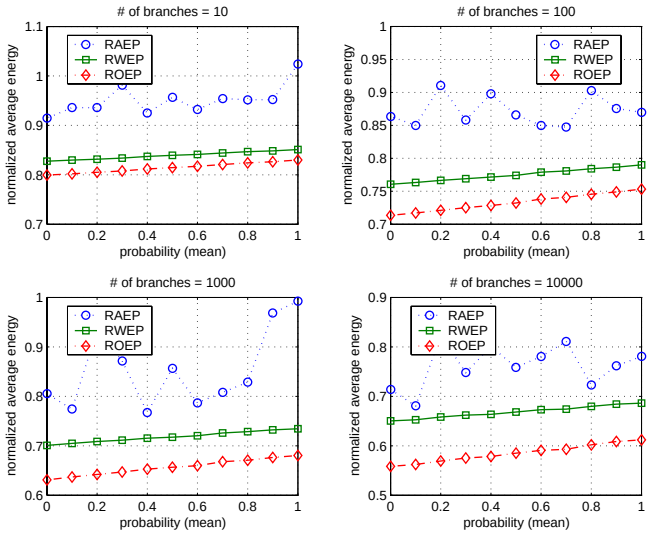

Figure 3: Simulation results for randomly generated tasks while varying the number of branches and probability distribution.

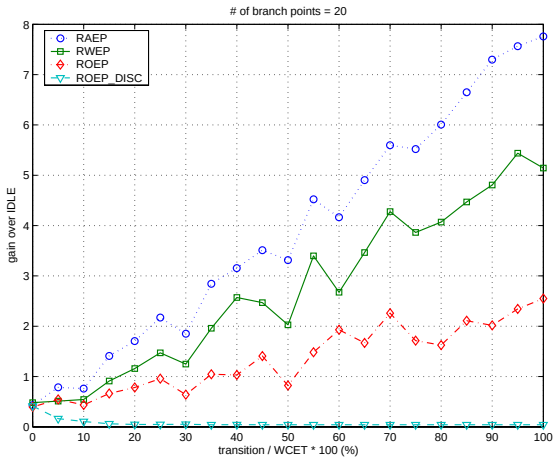

Figure 4: Simulation results for randomly generated tasks with 20 branches varying the transition interval.

Step2 Disable the voltage transitions that cause an increase of energy consumption by deleting the corresponding transition codes.

Step3 (immediately before executing b_i) Choose the nearest (discrete) speed f_i such that $f_i \geq f_{\mathrm{LB}}(b_i)$.

Note that **Step1** and **Step2** are carried out before the execution of the task, while **Step3** is conducted during the task execution. **Step2** is a core engine of ROEP_DISC. In this step, we disable the voltage transitions that yield no gain in energy consumption.

For a given configuration[3] and a specific scheduling technique, let $E^{-}(\pi, b_i)$ be the amount of energy consumption in the execution path π when the voltage transition at b_i is disabled. Then, we define the *transition gain* $G(b_i)$ for b_i as follows

$$G(b_i) = \sum_{\forall \text{execution path } \pi} \{P(\pi) \cdot (E^{-}(\pi, b_i) - E(\pi))\}. \qquad (11)$$

It is obvious that the transition with a negative gain should be removed in order to save energy.

Algorithm 1 describes the details of **Step2**. First, we compute the transition gain for each basic block (Lines 7-13). Then, the basic block with minimum transition gain is selected and the corresponding transition code is removed (Lines 14-16). This process is repeated until there is no transition code to remove. Finally, the best configuration that produces the minimum average energy consumption is returned as output. Note that after the removal of a transition code, the length of ROEP for the corresponding basic block or its immediate predecessor should be adjusted to reflect the change (Lines 18-22).

Contrary to the strategy which makes a decision on allowing or disallowing the transition during the execution of task, our technique, which disables the transitions *before the execution*, has two benefits; (1) it avoids an excessive runtime in calculating transition gains during execution, and (2) it can disable transitions that increase the speed. Note that there are two types of transitions, one that increases the speed and another that decreases it. For the online strategy, the latter transitions should not be disabled even if they incur unnecessary energy consumption, because otherwise, the deadline constraint may not be satisfied.

Since the number of execution paths increases nearly exponentially as the number of branches increases, $G(b_i)$ and E_{avg} may be *estimated* rather than calculated. For $G(b_i)$, summing each sub-gain without the probability term was verified to be a good estimation. To approximate E_{avg}, the calculation can be done over a limited number of execution paths.

[3] A configuration refers to a source code of the task augmented with transition codes.

5. EXPERIMENTAL RESULTS

We tested the proposed approach on a set of benchmark tasks as well as randomly generated tasks to demonstrate the effectiveness of the proposed voltage scheduling techniques. We assumed that the target system enters a power-down mode when the system is idle and the energy consumption of the power-down mode is 5% of that of the normal mode running at maximum speed [13].

• **Assessing the effectiveness of the ROEP-based scheduling technique:** We experimented on a set of randomly generated tasks while varying the number of branches and the probability at each branch. We compared our ROEP (ROEP-based technique) with two representative intra-task voltage scheduling techniques, RWEP [10] and RAEP [11], which are, respectively, a remaining worst and average case execution path based techniques. Since none of the previous techniques on intra-task voltage scheduling, including RWEP and RAEP, take into account the transition overheads completely, we simply assumed the voltage transition interval is 0 and no additional energy is consumed during the transition. The deadline of each task was set to the value of the worst-case execution time (WCET) of the task using the highest speed.

Figure 3 shows comparisons of the results. The x-axis of each graph represents the probability (mean) that the longer execution path is taken. The probability at each branch of the tasks was drawn from a random normal distribution with a standard deviation of 1.0. The y-axis indicates the amount of energy consumed by the techniques that are normalized with respect to that of a naive approach, IDLE, in which no DVS is applied, i.e., tasks are executed at the highest speed/voltage from the beginning to the end and at the termination of the execution the system becomes idle using a power-down mode. To obtain the average characteristic of each technique, the experiment was repeated 10000 times. From the graphs, we can easily see that the energy consumed by ROEP is always lower than the others and it decreases as the number of branches increases. It also decreases as the probability of taking a longer execution path becomes lower. RWEP shows this tendency clearly while RAEP does not, mainly because the speed in RAEP changes rather arbitrarily.

• **Assessing the effectiveness of ROEP with the consideration of transition overheads:** Since there is no accurate model for the tran-

Benchmark Task [19]	Energy Consumption (*normalized to* RWEP)									
	IDLE	RWEP [10]	RAEP [11]	ROEP	RWEP [10]	RAEP [11]	ROEP	ROEP* (*minimized*)	ROEP†	ROEP_DISC
discrete voltages‡	x	x	x	x	x	x	x	x	o	o
transition overheads§	x	x	x	x	o	o	o	o	o	o
amotsa	1.5379	1	1.0311	0.9398	1.2301	1.2651	1.2380	1.2286	1.3012	1.1089
dawson	1.8745	1	1.0017	0.9954	1.2947	1.2641	1.2794	1.2812	1.4128	1.2912
gcf	2.2983	1	1.1126	0.8939	2.0664	2.0538	2.0346	2.0690	2.0078	1.3069
gser	2.8891	1	0.9872	0.9576	3.2355	3.2676	3.3835	3.1273	3.2061	1.0948
gsimp	4.6923	1	1.2405	0.3316	1.2531	2.1875	2.1875	2.1875	1.6676	0.7903
hypser	2.6390	1	1.0000	0.9781	2.9733	2.9733	3.0775	2.8503	2.9529	1.1766
igray	1.9714	1	1.6348	0.9722	1.8208	2.2606	1.8539	1.7552	1.9465	1.3281
realft	1.8046	1	1.0000	0.9641	1.2726	1.2365	1.2623	1.2405	1.3052	1.1521
rtnewt	4.4799	1	1.0000	0.9085	6.0877	6.0877	6.6428	6.0945	5.6929	1.5648
trsec	1.8409	1	0.9999	0.9748	1.7488	1.7487	1.7659	1.7549	1.8657	1.2402
sncndn	2.4788	1	1.5553	0.9140	1.8962	2.8064	1.8193	1.8125	1.8638	1.1456
trapzd	1.2940	1	1.0000	1.0000	1.1386	1.1386	1.1362	1.1392	1.4125	1.3621
Average	2.4834	1	1.1303	**0.9025**	2.1682	2.3575	2.3067	**2.2117**	2.2196	**1.2135**

* Energy consumption is minimized by using Theorem 4.1.
† This strategy refers to ROEP_DISC without the selective removal step of unnecessary transition codes.
‡ o : Transmeta Crusoe 1000 MHz TM5800 Processor [16] (7 voltage levels) is used, x : A continuously variable voltage processor is used.
§ o : Transition overheads are taken into account, x : Transition overheads are ignored.

Table 1: Comparisons of results produced by our ROEP-based technique and existing techniques for tasks in [19]

sition energy, we simply assumed that energy is consumed during the transition interval at the voltage that varies linearly with time. The transition cycle Δn, which is a machine-dependent constant, was assumed to be 0. We generated a random task with 20 branches and set the slack factor, defined as $\frac{\text{deadline}-\text{WCET}}{\text{deadline}}$, to 0.1, giving a tight bound. We then obtained the energy consumption of IDLE, RAEP, RWEP, and ROEP, respectively, for the task with a processor to which continuous voltage is allowed as well as with a Transmeta Crusoe 1000 MHz TM5800 Processor [16], which has 7 discrete voltage levels. We repeated the experiment 100 times, varying the ratio $\frac{\text{transition interval}}{\text{deadline}}$ from 0 to 1. The results are shown in Figure 4. From the graph, one can immediately notice that the transition overheads (both interval and energy) can cause a dramatic increase in energy consumption. In the discrete voltage environment, our extended technique ROEP_DISC clearly outperformed the others, as indicated in Figure 4.

• **Assessing the effectiveness of our techniques on a set of real tasks [19]:** We assumed that the transition interval is 5% of WCET of each task. A probability of 0.5 was assigned to each of the branches for which we could not collect the execution profiles with sample input data. The amount of consumed energy, measured in each experiment was normalized to that of RWEP. The results are shown in Table 1. In short, ROEP reduces the energy consumption by 20.15% and 9.75% on average and by up to 66.84% and 73.27% over that of RWEP and RAEP, respectively. When the transition overheads are included, RWEP, RAEP, and ROEP use 116.82%, 108.57%, and 155.59% additional energy, respectively. In the discrete voltage environment, ROEP_DISC reduces the energy by 45.33% on average over that of the strategy corresponding to column 10, which is a simple extension of ROEP to handle discretely variable voltages.

6. CONCLUSIONS

We presented a set of comprehensive and important techniques for the intra-task voltage scheduling problem in hard real-time applications. The key contributions of our work are: (1) we solved the intra-task voltage scheduling problem optimally, proposing an energy optimal scheduler ROEP; (2) we then extended ROEP to solve a set of practical issues regarding the transition overheads; and (3) proposed a novel technique, ROEP_DISC, to solve the intra-task voltage scheduling problem in a discretely variable voltage environment. From experiments, we verified the optimality of ROEP and confirmed that ROEP_DISC reduces energy consumption effectively. We are currently investigating the impact of our intra-task scheduling techniques on the inter-task scheduling problem.

ACKNOWLEDGEMENT : This work was supported by the Korea Science and Engineering Foundation (KOSEF) through the Advanced Information Technology Research Center (AITrc).

7. REFERENCES

[1] F. Yao, A. Demers and S. Shenker, "A Scheduling Model for Reduced CPU Energy", *IEEE Symposium on Foundations of Computer Science*, 1995
[2] T. Ishihara and H. Yasuura, "Voltage Scheduling Problem for Dynamically Variable Voltage Processors", *ISLPED*, 1998
[3] I. Hong, M. Potkonjak and M.B. Srivastava, "On-line Scheduling of Hard Real-Time Tasks on Variable Voltage Processor", *ICCAD*, 1998
[4] Y. Shin and K. Choi, "Power Conscious Fixed Priority Scheduling for Hard Real-Time Systems", *DAC*, 1999
[5] W. Kwon and T. Kim, "Optimal Voltage Allocation Techniques for Dynamically Variable Voltage Processors" *DAC*, 2003
[6] S. Lee and T. Sakurai, "Run-time Voltage Hopping for Low-power Real-time Systems" *DAC*, 2000
[7] D. Mossé *et al.*, "Compiler-Assisted Dynamic Power-Aware Scheduling for Real-Time Applications", *COLP*, 2000
[8] C. Hsu, U. Kremer and M. Hsiao, "Compiler-Directed Dynamic Voltage/Frequency Scheduling for Energy Reduction in Microprocessors", *ISLPED*, 2001
[9] F. Gruian "Hard Real-Time Scheduling for Low-Energy Using Stochastic Data and DVS Processors" *ISLPED*, 2001
[10] D. Shin, J. Kim and S. Lee, "Intra-Task Voltage Scheduling for Low-Energy Hard Real-Time Applications", *IEEE Design & Test of Computers*, 2001
[11] D. Shin and J. Kim, "A profile-based energy-efficient intra-task voltage scheduling algorithm for hard real-time applications", *ISLPED*, 2001
[12] A. Azevedo *et al.*, "Profile-based Dynamic Voltage Scheduling using Program Checkpoints" *DATE*, 2002
[13] T.D. Burd and R.W. Broderson, "Processor Design for Portable Systems", *J. VLSI Signal Processing Systems*, 1996
[14] Intel Corporation, Enhanced Intel SpeedStep technology, 2003 http://www.intel.com
[15] Advanced Micro Devices, Inc. AMD PowerNow! technology, 2003 http://www.amd.com
[16] Transmeta Corporation, Crusoe Processor Specification, 2003 http://www.transmeta.com
[17] B. Mochocki, X. S. Hu and G. Quan, "A Realistic Variable Voltage Scheduling Model for Real-Time Applications", *ICCAD*, 2002
[18] J. Pouwelse, K. Langendoen and H. Sips, "Dynamic Voltage Scaling on a Low-Power Microprocessor", *MOBICOM*, 2001
[19] W. H. Press, B. P. Flannery, S. A. Teukolsky and W. T. Vetterling, *Numerical Recipes in C*, Cambridge University Press, 1988.