A Profile-Based Energy-Efficient Intra-Task Voltage Scheduling Algorithm for Hard Real-Time Applications*

Dongkun Shin
School of Computer Science and Engineering
Seoul National University
sdk@davinci.snu.ac.kr

Jihong Kim
School of Computer Science and Engineering
Seoul National University
jihong@davinci.snu.ac.kr

ABSTRACT

Intra-task voltage scheduling (IntraVS), which adjusts the supply voltage within an individual task boundary, is an effective technique for developing low-power applications. In this paper, we propose a novel intra-task voltage scheduling algorithm for hard real-time applications based on average-case execution information. Unlike the original IntraVS algorithm where voltage scaling decisions are based on the worst-case execution cycles, the proposed algorithm improves the energy efficiency by controlling the execution speed based on average-case execution cycles while still meeting the real-time constraints. The experimental results using an MPEG-4 decoder program show that the proposed algorithm reduces the energy consumption by up to 34% over the original IntraVS algorithm.

1. INTRODUCTION

Since energy consumption E of CMOS circuits has a quadratic dependency on the supply voltage V_{DD}, lowering the supply voltage V_{DD} is the most effective way of reducing energy consumption. However, lowering the supply voltage also decreases the clock speed, since the CMOS circuit delay T_D is given by $T_D \propto V_{DD}/(V_{DD} - V_T)^{\alpha}$ [6], where V_T is a threshold voltage, and α is a velocity saturation index. This trade-off introduced various dynamic voltage scaling (DVS) techniques. DVS techniques change the clock speed and its corresponding supply voltage dynamically to the lowest possible level while meeting the task's performance constraint.

1.1 Dynamic Voltage Scaling

For hard real-time systems, there exist two DVS approaches depending on the scaling granularity. *Inter-task voltage scheduling* (InterVS) [9, 2, 8, 5] determines the supply voltage on task-by-task basis, while *intra-task voltage scheduling* (IntraVS) [4, 7] adjusts the supply voltage within an individual task boundary. Both approaches can guarantee the required performance constraints of real-time systems.

*This work is supported by the Ministry of Information & Communication of Korea (Support Project of University foundation research<'00> supervised by IITA).

ISLPED'01, August 6-7, 2001, Huntington Beach, California, USA.

Intra-task voltage scheduling [4, 7] has been proposed as a solution to overcome the limitations of inter-task voltage scheduling. IntraVS algorithms exploit all the slack time from run-time variations of different execution paths; there is no slack time when the scheduled program completes its execution, thus significantly improving energy efficiency. Furthermore, since IntraVS does not involve OS in adjusting the clock speed, it can be used with an existing OS without any modifications on a variable voltage processor.

We propose an energy-efficient IntraVS algorithm for hard real-time applications based on *average-case* execution information. Unlike the original IntraVS algorithm [7] where voltage scaling decisions are based on *worst-case execution cycles*, the proposed algorithm controls the execution speed based on the *average-case* execution paths (ACEPs), which are the most frequently executed paths. Since the proposed algorithm is optimized for the energy reduction in the ACEP(s), which are the most likely path(s) that will be executed at run time, the proposed algorithm is more effective than the original intraVS algorithm [7] in reducing the energy consumption. The novel aspect of the proposed algorithm is that the timing constraints of a hard real-time program is still satisfied, even if the ACEP(s) are used for voltage scaling decisions.

2. ORIGINAL INTRA-TASK VOLTAGE SCHEDULING ALGORITHM

For a hard real-time task, the goal of an intra-task voltage scheduling algorithm is to assign a proper clock speed to each basic block so that energy consumption is minimized while satisfying timing requirements. In this section, we briefly describe the original intra-task voltage scheduling algorithm [7] as a short introduction to intra-task voltage scheduling.

Throughout this paper, we assume the following about the target variable voltage processor: The processor provides a special instruction, `change_f_V(`f_{CLK}`)`, which changes the current clock frequency to f_{CLK} and adjusts the supply voltage to the corresponding voltage V_{DD}. f_{CLK} and V_{DD} can be set continuously within the operational range of the processor. When the processor changes the clock/voltage, there is a clock/voltage transition overhead. During clock/voltage transition, the processor stops running.

Consider a hard real-time program P with the deadline of 2 μsec shown in Fig. 1(a). The CFG G_P of the program P is shown in Fig. 1(b). In G_P, each node represents a basic block of P and each edge indicates the control dependency between basic blocks. The number within each node indicates $C_{EC}(b_i)$ which is the number of execution cycles of the corresponding basic block. The back edge from b_5 to b_{wh} models the **while** loop of the program P.

Using a WCET analysis tool, we can find the path p_{worst} = (b_1,b_{wh}, b_3,b_4,b_5,b_{wh},b_3,b_4,b_5,b_{wh},b_3,b_4,b_5,b_{wh},b_{if},b_6,b_7) as the worst case

execution path (WCEP) for the example program P, assuming that the maximum number of **while** loop iterations is set to 3 by user. The predicted execution cycles of p_{worst} is, therefore, 160 cycles, which is the worst case execution cycles (WCEC) of program P. If a target processor operates at the maximal 80-MHz clock frequency, the program P completes its execution in 2 μsec, resulting in no slack time.

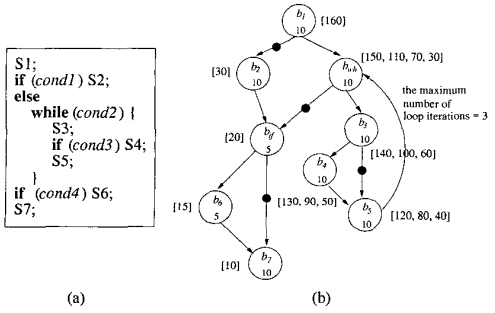

Figure 1: An example program; (a) an example real-time program P and (b) its CFG G_P.

The key observation behind the IntraVS approach is that there are large execution time variations among different execution paths. For short execution paths, if we were able to identify them in the early phase of its execution, we can lower the clock speed substantially, thus saving a significant amount of energy consumption.

For the speed adjustment, intra-task voltage scheduling technique uses an adaptive approach with the help of a static program analysis technique on worst case execution times. Assume that $C_{RWEC}(b_i)$ represents the remaining worst case execution cycles (RWEC) among all the execution paths that start from b_i. Using a modified WCET analysis tool, for each basic block b_i, we can compute $C_{RWEC}(b_i)$ in *compile time*. In Fig. 1(b), the symbol [ ] contains the $C_{RWEC}(b_i)$ values of each basic block. For the basic blocks related to the **while** loop (i.e., b_{wh}, b_3, b_4, b_5), the corresponding nodes are associated with multiple $C_{RWEC}(b_i)$ values, reflecting the maximum three iterations of the **while** loop.

With the $C_{RWEC}(b_i)$ values computed, we can statically identify an edge (b_i, b_j) (of a CFG G) where $[C_{RWEC}(b_i) - C_{EC}(b_i)] \neq C_{RWEC}(b_j)$. For example, in Fig. 1(b), we can identify four such edges, i.e., (b_1, b_2), (b_{wh}, b_{if}), (b_{if}, b_7) and (b_3, b_5), which are marked by the symbol •. These marked edges form a set of candidate Voltage Scaling Edges (VSEs). If an edge (b_i, b_j) is selected as a VSE, it means that the clock speed will change when the thread of execution control branches to b_j from b_i. For example, the clock speed will be lowered when the basic block b_2 is executed after b_1 because the remaining work is reduced by 1/5 (i.e., the ratio of $C_{RWEC}(b_2)$ to $[C_{RWEC}(b_1) - C_{EC}(b_1)]$).

At the selected VSEs, the new clock speed is determined by how much the remaining work is reduced. For example, when the thread of execution control meets a VSE (b_i, b_j), the clock speed can be lowered because the remaining work is reduced by $[R - C_{RWEC}(b_j)]$ where $R = C_{RWEC}(b_i) - C_{EC}(b_i)$. After b_i is executed at the clock speed S, the clock speed can be changed to reflect the reduction in the remaining work. The new clock speed for b_j is set to $S \times \frac{C_{RWEC}(b_j)}{R}$. We call $\frac{C_{RWEC}(b_j)}{R}$ as the speed update ratio (SUR) for the edge $b_i \rightarrow b_j$, denoted by $SUR(b_i \rightarrow b_j)$.

By the original IntraVS algorithm, the clock speed is changed from 80 MHz to 16 MHz (= 80 MHz $\times \frac{30}{160-10}$) at the edge (b_1, b_2). Assuming that no energy is consumed in an idle state and $E \propto C_L \cdot N_{cycle} \cdot {V_{DD}}^2$, when the execution follows the path $p_1 = (b_1, b_2, b_{if}, b_6, b_7)$, the original IntraVS algorithm reduces the energy consumption by 69%.

Since there exists the transition overhead during speed changes, not all the candidate VSEs are selected as VSEs. A candidate VSE is selected as a VSE when the number of saved cycles at the candidate VSE is larger than a given threshold value. The threshold value is determined by a VSE selection policy, which is a function of the transition time overhead, the transition power overhead, and the code size increase (by the added scaling code).

3. PROFILE-BASED INTRA-TASK VOLTAGE SCHEDULING

3.1 Motivation

Before the profile-based IntraVS algorithm is presented, we first generalize the original IntraVS algorithm described in Section 2. In order to adjust the clock speed at VSEs, IntraVS first selects a (predicted) reference execution path such as the WCEP. Once the reference execution path is decided, IntraVS sets the initial operating voltage and its corresponding clock frequency assuming that the task execution will follow the predicted reference execution path.

When the actual execution deviates from the (predicted) reference execution path (say, by a branch instruction), the clock speed can be adjusted depending on the difference between the number of remaining execution cycles of the reference execution path and the number of remaining execution cycles of the newly deviated execution path. If the new execution path takes significantly longer to complete its execution than the reference execution path, the clock speed should be *raised* to meet the deadline constraint. On the other hand, if the new execution path can finish its execution earlier than the reference execution path, the clock speed can be *lowered* to save the energy consumption. Once the actual execution takes a different path from the reference path, a new reference path is constructed starting from the deviated basic block.

Using a static program-analysis technique, IntraVS identifies the appropriate program locations where the clock speed should be raised or lowered relative to the current clock speed. For the clock speed adjustment at run time, IntraVS algorithm inserts voltage scaling code to the selected program positions. The candidate positions for inserting voltage scaling code are the branching edges of the CFG, which correspond to the branch or loop statements.

We call the original IntraVS as the remaining worst-case execution path (RWEP)-based IntraVS, because the remaining worst-case execution path (RWEP) is used as the reference path. In the RWEP-based IntraVS, the clock speed is monotonically decreasing at all the VSEs. Depending on how the reference path is selected, however, the clock speed may be increased as well at some VSEs. Therefore, we divide VSEs into Up-VSEs and Down-VSEs. The clock speed is increased at an Up-VSE while the clock speed is decreased at a Down-VSE.

Although the RWEP-based IntraVS reduces the energy consumption significantly while guaranteeing the deadline, this is a pessimistic approach because it always predicts that the longest path will be executed. A more optimistic approach is to use the average case execution path (ACEP) as a reference path. The ACEP is defined to be an execution path that is most likely to be executed. The ACEP can be decided by the execution profile information.

The main motive of using the ACEP instead of the WCEP is to make the common case more energy-efficient. For a typical program, about 80 percent of the program's execution occurs in only 20 percent of its code, which is called the hot paths [1]. For an In-

traVS algorithm to be energy-efficient, it should be energy-efficient when the hot paths are executed. If we use one of hot paths as a reference path for intra-task voltage scheduling, the speed change graph for the hot paths will be a near flat curve with little changes in the clock speed, which gives the best energy efficiency under a given amount of work [3]. Even for the paths that are not the hot paths, if we take one of hot paths as a reference paths, they are more energy-efficient because they can start with a lower clock speed than when the WCEP is used as a reference path.

In the profile-based IntraVS, we take the ACEP, which is the best representative of the hot paths, as the reference path. We call such an IntraVS algorithm as the remaining average-case execution path (RAEP)-based IntraVS because the remaining average-case execution path (RAEP) is used as the reference path.

Figure 2 shows an RAEP-based CFG G_P^{RAEP} with $C_{RAEC}(b_i)$ values that represent the remaining average-case execution cycles among all the paths that start from b_i. The bold edges in G_P^{RAEP} means that it has a higher probability to be followed at run time between two branching edges. In Fig. 2, the initial reference path is $(b_1, b_{wh}, b_3, b_5, b_{wh}, b_3, b_5, b_{wh}, b_{if}, b_7)$. With the reference path, $C_{RAEC}(b_i)$ is computed. For example, $C_{RAEC}(b_{if}) = C_{EC}(b_{if}) + C_{RAEC}(b_7)$. At the RAEP-based IntraVS, there are Up-VSEs (marked by ○ in Fig. 2) as well as Down-VSEs (marked by • in Fig. 2). Figure 3 shows how the speed and voltage change by the RAEP-based scheduling. The speed is changed from 14 MHz to 21 MHz at the edge (b_{if}, b_6) because this is an Up-VSE with the SUR value of 1.5 ($= \frac{15}{15-5}$). Compared to the energy consumption of the RWEP-based IntraVS algorithm, the RAEP-based IntraVS algorithm achieves 55% more energy reduction.

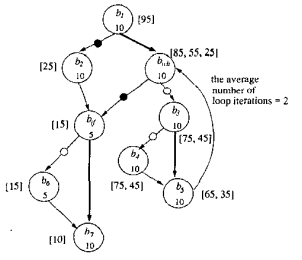

Figure 2: A RAEP-based CFG G_P^{RAEP}.

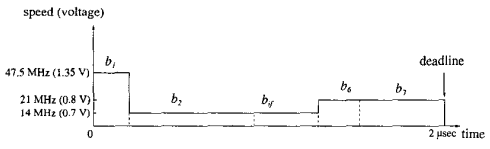

Figure 3: Speed and voltage changes by the RAEP-based IntraVS.

Though the RAEP-based scheduling is more effective in reducing the energy consumption than the RWEP-based scheduling, the pure RAEP-based approach cannot meet the timing requirements of hard real-time applications. This is because it dose not satisfy the timing constraint for all the execution paths. For example, consider the case when the WCEP and ACEP take significantly different number of execution cycles. When the execution takes the WCEP at the middle of program execution, it is possible that the program cannot meet its deadline even if the remaining path executes with the maximum clock speed. The next section describes a novel approach that is still based on the RAEC but can guarantee the timing constraint for all the execution paths.

3.2 Reference Path Modification

To overcome the deadline miss problem of the pure RAEP-based IntraVS algorithm, we modify the reference path whenever the deadline miss situations are identified. Assume that the reference path is $p_{ref} = (b_1, \cdots, b_i, b_{i+1}, \cdots, b_N)$, b_i is a branching node whose children basic blocks are b_{i+1} and b_{miss}, and the current clock speed at b_i is S. If the clock speed at b_{miss}, given by $S \times SUR(b_i \rightarrow b_{miss})$, is larger than the maximal clock speed ($MaxS$) of the processor, it indicates that if the current execution branches to b_{miss}, the deadline will be missed. This is because the remaining time T_R to the deadline is $T_R = \frac{C_{RAEC}(b_{i+1})}{S}$ and $MaxS \times T_R < C_{RAEC}(b_{miss})$. There are $M = \lceil C_{RAEC}(b_{miss}) - MaxS \times T_R \rceil$ cycles that miss the deadline. In order to avoid the deadline miss, we increment $C_{RAEC}(b_k)$ by M for all $k \leq i$. That is, we modify the reference path by adding a new virtual basic block b_v between b_i and b_{i+1}. $C_{EC}(b_v)$ is set to M. The virtual basic block is used only to prevent the deadline miss during the speed assignment and not executed at run time.

Figure 4 illustrates how the reference path modification works. Given an original G_P^{RAEP}, the ACEP, (b_1, b_3, b_4), is used as the reference path. (The bold edges indicate higher probability edges to be selected at run time.) With the 100-MHz maximal clock frequency, the path (b_1, b_3, b_5) misses the 0.5-μsec deadline, because the speed at (b_3, b_5) should be raised to 120 MHz (i.e., 60 MHz×2). Because $\frac{10}{3}$ cycles[1] are missed from the deadline, we add a virtual block b_v between b_3 and b_4, as shown in Fig. 4(b). $C_{EC}(b_v)$ is set to 4 ($= \lceil \frac{10}{3} \rceil$).

With the added b_v, $C_{RAEC}(b_1)$ and $C_{RAEC}(b_3)$ are modified to 34 and 24, respectively, and the speed update ratios are recalculated. For example, the SUR at (b_3, b_5) is modified to 1.43 ($= \frac{20}{14}$) from 2.

Figures 4(c) and 4(d) compare the speed changes for the RWEP-based IntraVS, the RAEP-based IntraVS and the modified RAEP-based IntraVS for the paths (b_1, b_3, b_4) and (b_1, b_3, b_5), respectively. The modified RAEP-based scheduling is more energy-efficient than the RWEP-based scheduling for the hot paths (Fig. 4(c)), which affect most on the overall energy efficiency. It also satisfies the deadline requirement (Fig. 4(d)), unlike the pure RAEP-based scheduling algorithm.

4. EXPERIMENTAL RESULTS

We have extended the existing voltage scaling tool, the Automatic Voltage Scaler (AVS) [7], to evaluate the energy efficiency of the proposed IntraVS over the original IntraVS. AVS takes as inputs an original *DVS-unaware* program P and its timing requirements, and produces a low-energy *DVS-aware* program P_{DVS} that satisfies the same timing requirements of P. The converted program P_{DVS} contains voltage scaling code that handles all the idiosyncrasy of scaling speed/voltage on a variable voltage processor. The extended AVS can convert a program using either the RWEP-based IntraVS or the RAEP-based IntraVS.

To evaluate the power reduction effect of the proposed extensions to the original IntraVS algorithm, we have experimented with an MPEG-4 video decoder using an energy simulator [7]. We assume that both DVS-aware and DVS-unaware systems enter into a power-down mode when the system is idle. The energy consumption of a power-down mode is assumed to be 0. The supply voltage for a given clock frequency is obtained from $f_{CLK} = 1/T_D \propto (V_{DD} - V_T)^{\alpha}/V_{DD}$ [6] where V_{DD}, V_T, and α are assumed to be 2.5V, 0.5V, and 1.3, respectively. For the RAEP-based IntraVS, the

[1] 20 cycles - 100 MHz $\times \frac{10 \text{ cycles}}{60 \text{ MHz}} = \frac{10}{3}$ cycles

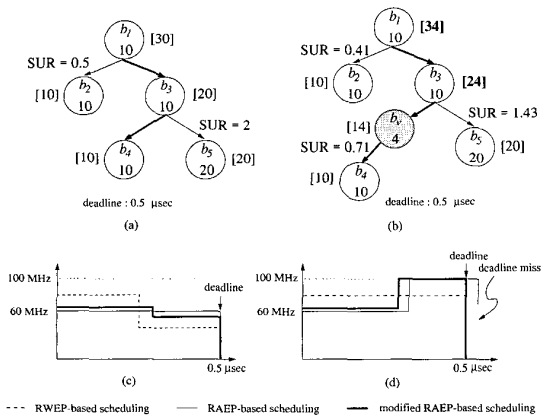

Figure 4: Modified RAEP-based IntraVS: (a) an original G_P^{RAEP}, (b) a modified G_P^{RAEP}, and (c)-(d) the speed change graphs of three IntraVS algorithms for the paths (b_1, b_3, b_4) and (b_1, b_3, b_5), respectively.

probability of branch edges and the average number of loop iterations in a CFG of the MPEG-4 video decoder are estimated using the profiled information. A probability of 0.5 is assigned to the branch edges for which we cannot collect the execution profiles with sample test bitstreams. For the experiments, the slew rate of the clock/voltage transition is assumed to be 1.0V/200μsec, which is typical for state-of-the-art DC-DC converters.

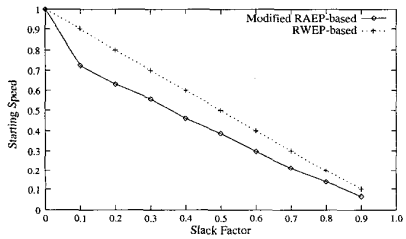

Figure 5: Normalized starting speed changes of the RWEP-based IntraVS and the RAEP-based IntraVS (varying the slack factor).

Figure 5 shows how the normalized starting speeds change over various slack factor values. The slack factor, defined by $\frac{deadline - WCET}{deadline}$, represents the fraction of time that a processor becomes idle after WCET. The execution times of modified ACEPs (by the procedure described in Section 3.2) for the MPEG-4 decoder is up to 35% smaller than the WCET. This means that the processor can start initially 35% more slowly than the speed required by the RWEP-based IntraVS algorithm.

Figure 6 compares the energy consumption of two IntraVS scheduling algorithms, varying the slack factor. (All the results were normalized over the energy consumption of the original program running on a DVS-unaware system.) For the MPEG-4 decoder, the modified RAEP-based IntraVS algorithm reduces the energy consumption up to 34% over the RWEP-based IntraVS algorithm.

Note that there is a large gap between energy consumption of RWEP-based and RAEP-based IntraVS algorithms, even when the slack factor is 0 (i.e. deadline = WCET). This is because, although the starting speed is set to the same speed as in the RWEP-based IntraVS, there are many execution paths that still can take advantage of the RAEP-based speed settings. That is, in order to meet the timing constraint, virtual blocks are added so that the initial speed is set to the same speed as in the RWEP-based IntraVS algorithm. However, the (partial) paths following the virtual blocks can take advantage of the ACEP-based speed settings. As the slack factor increases, the energy consumption gap decreases because supply voltages of both IntraVS algorithms get lower. Since the energy consumption is proportional to $V_{DD}{}^2$, the lower voltage values result in a smaller difference in the energy consumption.

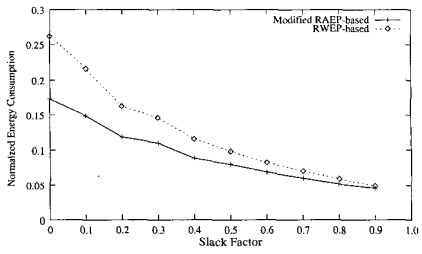

Figure 6: Normalized energy consumption of the RWEP-based IntraVS and RAEP-based IntraVS (varying the slack factor).

5. CONCLUSION

We have presented a novel IntraVS algorithm based on the RAEP information. The proposed algorithm exploits the fact that the average-case execution paths are more likely to be followed at run time than the WCEP, and optimize the energy consumption for such hot paths. The main contribution of the proposed algorithm is that it enhances the original IntraVS algorithm by exploiting the probability of each execution path, while guaranteeing the worst-case timing constraints. The experimental results using an MPEG-4 video decoder show that the RAEP-based IntraVS improves the energy efficiency up to 34% over the RWEP-based IntraVS.

6. REFERENCES

[1] T. Ball and J. R. Larus. Using paths to measure, explain, and enhance program behavior. *IEEE Computer*, 33(7):57-65, 2000.

[2] I. Hong, G. Qu, M. Potkonjak, and M. B. Srivastava. Synthesis techniques for low-power hard real-time systems on variable voltage processor. In *Proc. of the 19th IEEE Real-Time Systems Symposium*, pages 178-187, 1998.

[3] T. Ishihara and H. Yasuura. Voltage scheduling problem for dynamically variable voltage processors. In *Proc. of International Symposium On Low Power Electronics and Design*, pages 197-202, 1998.

[4] S. Lee and T. Sakurai. Run-time voltage hopping for low-power real-time systems. In *Proc. of the 37th Design Automation Conference*, pages 806-809, 2000.

[5] Y. Lee and C. M. Krishna. Voltage-clock scaling for low energy consumption in real-time embedded systems. In *Proc. of the 6th International Conference on Real-Time Computing Systems and Applications*, pages 272-279, 1999.

[6] T. Sakurai and A. Newton. Alpha-power law MOSFET model and its application to CMOS inverter delay and other formulas. *IEEE Journal of Solid State Circuits*, 25(2):584-594, 1990.

[7] D. Shin, J. Kim, and S. Lee. Intra-task voltage scheduling for low-energy hard real-time applications. *IEEE Design and Test of Computers*, 18(2):20-30, 2001.

[8] Y. Shin and K. Choi. Power conscious fixed priority scheduling for hard real-time systems. In *Proc. of the 36th Design Automation Conference*, pages 134-139, 1999.

[9] F. Yao, A. Demers, and S. Shenker. A scheduling model for reduced CPU energy. In *Proc. of the 36th Annual Symposium on Foundations of Computer Science*, pages 374-382, 1995.