Intra-Task Voltage Scheduling on DVS-Enabled Hard Real-Time Systems

Dongkun Shin, *Student Member, IEEE*, and Jihong Kim, *Member, IEEE*

***Abstract*—This paper proposes a novel intra-task dynamic voltage scheduling (IntraDVS) framework for low-energy hard real-time applications. Based on a static timing analysis technique, the proposed approach controls the supply voltage within an individual task boundary. By fully exploiting all the slack times, a scheduled program by the proposed technique always completes its execution near the deadline, thus achieving a high energy reduction ratio. The problem formulation of IntraDVS is first presented and two heuristics are proposed: one based on worst-case execution information and the other on average-case execution information. In order to validate the effectiveness of the proposed heuristics, a software tool that automatically converts a DVS-unaware program into an equivalent low-energy program was built. In an experiment on a DVS-enabled system, the low-energy version of a Moving Pictures Expert Group (MPEG)-4 encoder/decoder consumed only 35%–51% of the energy consumption of the original program running on a fixed-voltage system with a power-down mode. The energy efficiency of the IntraDVS algorithms was also compared with that of task-level voltage scheduling algorithms. The experimental results show that the IntraDVS algorithm can be useful in multitask environments as well.**

***Index Terms*—Dynamic voltage scaling, low-power design, power management, real-time systems, variable-voltage processor.**

I. Introduction

DYNAMIC voltage scaling (DVS) [1] is one of the most effective approaches in reducing the power consumption of embedded systems, where the supply voltage can be dynamically reduced to the lowest possible extent and still ensure a proper operation when the required performance of the target system is lower than the maximum performance. Since the dynamic energy consumption of CMOS circuits, which dominates the total power consumption, is proportional to the square of the supply voltage V_{dd}, a significant energy reduction is possible with the DVS scheme. Recently, many commercial variable-voltage microprocessors (e.g., [2]–[4]) have been introduced to the mobile embedded market, reflecting the effectiveness of DVS techniques.

In the past, various voltage scheduling algorithms have been proposed for hard real-time systems [5]–[12]. Given multiple tasks, these algorithms assign the proper speed to each task dynamically while guaranteeing all their deadlines. In real-time systems, since the real execution time of each task may be smaller than the worst-case execution time (WCET), workload-variation slack times [13] are generated at run time even though the worst-case utilization of the processor is 1. However, it is difficult to utilize the workload-variation slack times because the exact amount of slack time before the completion of a task cannot be known. Therefore, most DVS scheduling algorithms [7]–[12] transfer the slack time to the following tasks that can utilize it. These techniques exploit the "slack estimation and distribution" strategy for supply voltage determination, which can be summarized as follows: 1) running the current task; 2) estimating the slack time due to early completion of the current task; 3) distributing the slack time to the next tasks and determining the operating speed of the tasks; and 4) running the next tasks. These techniques determine the supply voltage on a task-by-task basis. For each task activation, only one supply voltage is assigned to the task, and it is not changed during the task execution. In this paper, these techniques are called as intertask dynamic voltage scheduling (InterDVS).

While generally effective in reducing the energy consumption of multitask real-time systems, InterDVS has several practical limitations. For example, since a task scheduler in an operating system (OS) determines the supply voltage of a task, it requires OS modifications. Furthermore, it cannot be applied to a single-task environment because there is no other task that can utilize the slack time generated by the completed task. Considering many small-size embedded mobile applications are based on a single-task model, this can be detrimental to a wide adoption of variable-voltage processors in practice.

Even in a multitask environment, InterDVS may not be effective in reducing the energy consumption if one task is dominant in both the slack times and the execution times. In this case, a dominant task (with the highest energy consumption) exploits slack times from other tasks (with small slack times), thus ineffective in reducing the energy consumption. For example, consider a typical mobile videophone application with four tasks shown in Table I. Using the InterDVS algorithm of [8], only 17% energy reduction is observed while an off-line (theoretical) optimal voltage scheduling can achieve 90% power reduction.[1]

In this paper, a novel voltage scheduling framework is proposed that can utilize the workload-variation slack times within the current task, not the following tasks. The proposed

Manuscript received September 2, 2003; revised April 14, 2004. This work was supported by the University IT Research Center Project. This paper was recommended by Associate Editor R. Camposano.

D. Shin is with the Samsung Electronics Co., Seoul, Korea (e-mail: dongkun.shin@samsung.com).

J. Kim is with the School of Computer Science and Engineering, Seoul National University, Gwanak-gu, Seoul 151-742, Korea (e-mail: jihong@davinci.snu.ac.kr).

Digital Object Identifier 10.1109/TCAD.2005.852036

[1]The energy reduction by the InterDVS algorithm of [8] was estimated using a simulation. The off-line optimal schedule was calculated by assuming that the real execution times of tasks are known *a priori*.

TABLE I
TYPICAL VIDEOPHONE APPLICATION

		MPEG-4 Video Encoding	MPEG-4 Video Decoding	VSELP Speech Encoding	VSELP Speech Decoding
Period (= Deadline) (msec)		66.667	66.667	40.000	40.000
Worst Case Execution Time (msec)		50.386	9.826	1.844	1.383
Average Execution Time (msec)		13.099	1.460	0.907	0.680
Normalized Energy Consumption	InterDVS [8]	0.826			
	Off-line Optimal	0.106			

*In normalizing energy consumption values, the base case of normalization is DVS-unaware systems using only power-down mode.

technique is called intra-task DVS (IntraDVS) because it adjusts the voltage and the clock speed within a task. The technique identifies the slack time generated from the workload variation and adjusts the clock/voltage to utilize the slack time at run time. To enable the run-time clock/voltage adjustment, the application code is preprocessed based on the static timing analysis and the profile information of program execution.

The workload-variation slack times are identified by observing the changes of the remaining execution times due to control flow. Depending on the technique of how to predict the remaining execution time, two kinds of IntraDVS algorithms are described, one using worst-case execution information and the other using average-case execution information. Although average-case execution information is exploited in the second algorithm, the timing constraints of a hard real-time program are still guaranteed.

The proposed IntraDVS algorithm has the following features: 1) it fully exploits all workload-variation slack times, achieving a significant improvement in the energy consumption; 2) it is applicable to a single-task environment, since it controls the supply voltage within each task[2]; 3) it provides an automatic conversion tool that converts DVS-unaware programs into DVS-aware ones, meaning that a programmer requires no knowledge on DVS, making the proposed algorithm very practical; and 4) it enables each individual task to control the supply voltage independent of other tasks without any support from operating systems. Therefore, it can be directly applied to a conventional DVS-unaware OS without any modification.

Based on the proposed IntraDVS algorithm, a software tool called automatic voltage scaler (AVS) that automatically converts a DVS-unaware program into an equivalent low-energy program has been developed. In the experiment using a real DVS-enabled system, the low-energy version of a Moving Pictures Expert Group (MPEG)-4 encoder/decoder consumed only 35–51% of the energy consumption from the original program running on a fixed-voltage system with a power-down mode.

The rest of this paper is organized as follows. Section II summarizes related works on DVS and compares them with the work here. While the framework and the overall descriptions of the proposed IntraDVS are introduced in Section III, the details of the IntraDVS algorithm are described in Section IV. The experimental results are presented in Section V. Section VI concludes with a summary and future works.

[2]This does not necessarily mean that the proposed IntraDVS algorithm is not applicable to multitask environments. When used under an InterDVS algorithm, which assigns a time slot and a speed for each task at run time, the proposed algorithm can additionally adjust the execution speed at the intra-task level. The proposed IntraDVS algorithm forces each task to use only the time slot assigned by the OS scheduler in multitask environments. When an InterDVS algorithm determines a time slot for a task at run time, the IntraDVS has only to adjust the initial start speed of the task based on the assigned time slot.

II. RELATED WORKS

For hard real-time systems where timing constraints must be strictly satisfied, a fundamental energy delay tradeoff makes it more challenging to adjust the supply voltage dynamically while minimizing the energy consumption and guaranteeing the timing requirements. For this reason, extensive studies have been recently carried out on the InterDVS problems [5]–[12], [14]–[16]. The energy efficiencies of state-of-the-art InterDVS algorithms have been evaluated in [17].

For the IntraDVS technique, several kinds of approaches have been introduced. An intra-task voltage scheduling where each task is partitioned into fixed-length segments has been proposed in [13]. After the completion of each segment, the supply voltage is adjusted depending on the slack time made by the previous segment. Although [13] shows a significant improvement in the energy reduction, it provides no systematic methodology for developing DVS-aware intra-task applications. For example, there exists no systematic guideline of selecting the best program locations where the voltage scaling code is inserted. Consequently, the programmer himself should find out proper locations based on his own knowledge. It implies that the technique described in [13] is very difficult to be applied to practical applications since average programmers are generally not familiar with low-energy software issues as well as timing analysis techniques.

To provide a systematic methodology for developing DVS-aware intra-task applications, [18] and [19] proposed two IntraDVS techniques based on the program analysis. This paper generalizes the earlier IntraDVS heuristics in a unified framework and extensively evaluates their performance.

Another approach of IntraDVS is based on the stochastic method [20], [21]. This technique is motivated by the idea that it is usually better to "start at low speed and accelerate execution later when needed" than to "start at high speed and reduce the speed later when the slack time is found" in the program execution. However, it requires the probability density function of execution times of a task to calculate the optimal speed schedule. Furthermore, the stochastic IntraDVS requires OS modification

TABLE II
VARIABLE VOLTAGE PROCESSORS

Processor		Clock Range	Voltage Range	Transition Time
Commercial	Transmeta's Crusoe [2]	200–700(megahertz)	1.1–1.65(volt)	300 μs
	AMD's Mobile K6 [3]	192–588(megahertz)	0.9–2.0(volt)	200 μs
	Intel PXA250 [4]	100–400(megahertz)	0.85–1.3(volt)	500 μs
	Compaq's Itsy [24]	59.0–206.4(megahertz)	1.0–1.55(volt)	189 μs
	TI's TMS320C55x [23]	6–200(megahertz)	1.1–1.6(volt)	3.3 ms($1.6 \rightarrow 1.1$ V) 300 μs($1.1 \rightarrow 1.6$ V)
Academic	Burd *et al.* [1]	5–80(megahertz)	1.2–3.8(volt)	520 μs
	LART [25]	59–251(megahertz)	0.79–1.65(volt)	5.5 ms($1.65 \rightarrow 0.79$ V) 40 μs($0.79 \rightarrow 1.65$ V)

like InterDVS and cannot utilize all the slack times. In this paper, the authors compare the energy efficiency of the stochastic IntraDVS technique with the proposed heuristics.

A different kind of IntraDVS is also proposed in [22]. While the IntraDVS here estimates the slack times of a task based on the changes of the remaining execution cycles due to the control flow, the technique in [22] finds the slack times based on the architectural characteristic of the microprocessor. By identifying the program regions in which the CPU is mostly idle due to memory stalls, the technique in [22] slows down the clock speed of the CPU in the regions for energy reduction with negligible performance loss. However, this technique should be carefully used in real-time systems due to performance degradation. Since this technique is different from our IntraDVS in the kind of slack times exploited, it can be used together with our IntraDVS for better energy performance.

The DVS technique proposed for multimedia applications in [14] uses a special technique distinguishing it from other InterDVS techniques evaluated in [17]. Their algorithm fully utilizes the idle intervals with buffers in a variable speed processor and determines the minimum buffer size to achieve the maximum energy saving. The energy performance of the technique is strongly dependent on the available buffer size. This technique has a limitation that it can be used only when a system can buffer multiple input data or output results.

III. INTRA-TASK VOLTAGE SCHEDULING FRAMEWORK

A. Machine Model

Recently, many variable-voltage processors have been announced. Table II shows the representative commercial variable-voltage processors and academic trials to implement variable-voltage processors. These processors provide finite numbers of voltage and clock levels within the voltage/clock range specified. Each processor requires a time delay to change the voltage/clock level. Most of the variable-voltage processors, except Transmeta's Crusoe, provide the software mechanisms for users to be able to control the voltage and clock level such as TI's Power Scaling Library [23]. Throughout this paper, the authors make the following assumptions on a target variable-voltage processor: 1) the processor provides a special instruction `change_f_V`($f_{\rm clk}$) that dynamically controls the clock frequency $f_{\rm clk}$ and its corresponding voltage $V_{\rm dd}$ of the processor; 2) $f_{\rm clk}$ and $V_{\rm dd}$ have continuous values within the operational range of the processor; 3) when the processor changes from ($f_{\rm clk1}$, $V_{\rm dd1}$) to ($f_{\rm clk2}$, $V_{\rm dd2}$), there is a clock/voltage transition overhead (VTO) period of $C_{\rm VTO}$ cycles[3]; and 4) during clock/voltage transition, the processor stops running.

Assumptions 1), 3), and 4) are valid for most variable processors. Especially, assumption 4) is conservative considering recent variable-voltage processors such as TI's TMS320C55x, which stops during the clock transition but operates during the voltage transition. Assumption 2) is not realistic because most variable-voltage processors provide only discrete voltage/clock levels. However, the work presented here can support the processors with a finite number of voltage/clock levels with a slight modification of the speed selection algorithm.

B. Basic Idea

Consider a hard real-time program P with a deadline of 2 s shown in Fig. 1(a). The control flow graph (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 the number of execution cycles of the corresponding basic block. The back edge from b_5 to $b_{\rm wh}$ models the while loop of the program P.

In developing hard real-time systems where tasks have strict timing constraints (i.e., deadlines), the WCETs of the tasks are estimated in advance (prior to run time) to guarantee that required timing constraints are met. Such WCETs can be predicted by existing analysis tools that produce safe and accurate prediction results [26], [27]. Using a WCET analysis tool, the authors can find the path $p_{\rm worst} = (b_1, b_{\rm wh}, b_3, b_4, b_5, b_{\rm wh}, b_3, b_4, b_5, b_{\rm wh}, b_3, b_4, b_5, b_{\rm wh}, b_{\rm if}, b_{\rm call4}, b_8, b_{10}, b_{11}, b_7)$ as the worst-case execution path (WCEP) for the example program P, assuming that the maximum number of while loop

[3]The clock/voltage transition time is different depending on the source voltage and the target voltage. However, it was assumed that there is a fixed voltage transition time for a simple explanation. Since the authors represent the fixed clock/voltage transition overhead period by the number of cycles, it can vary depending on the current clock frequency. For a simpler analysis, it was assumed that $C_{\rm VTO}$ cycles were counted under the maximum clock frequency.

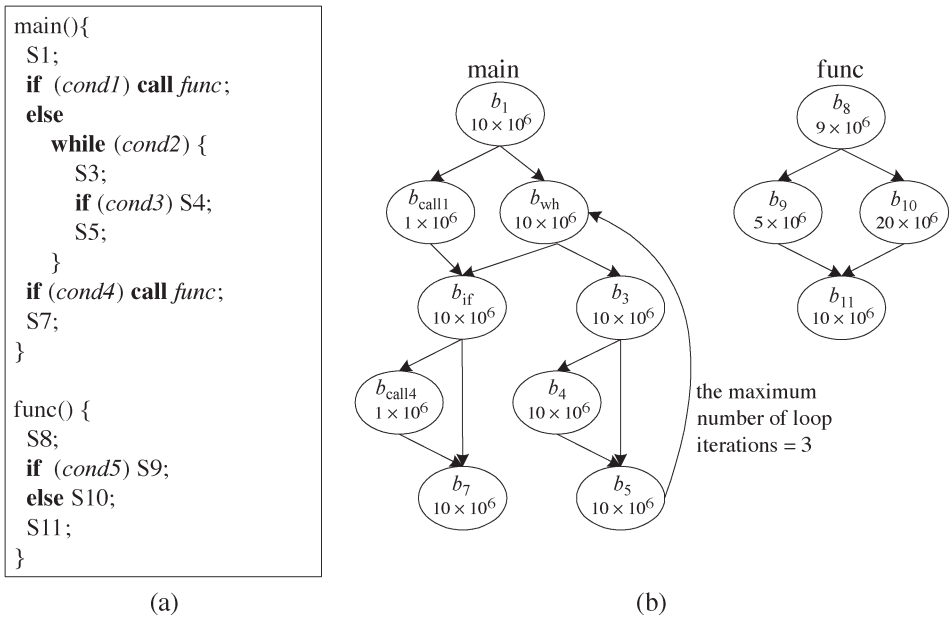

Fig. 1. Example program P: (a) an example real-time program with the 2-s deadline and (b) its CFG representation G_P.

iterations is set to 3 by the user. The predicted execution cycles of p_{worst} are, therefore, 200×10^6 cycles, which are the worst-case execution cycles (WCECs). If a target processor operates at the maximal clock frequency of 100 MHz, the program P completes its execution in 2 s, resulting in no slack time.

However, there are large execution time variations among different execution paths. In particular, the average-case execution paths (ACEPs) complete their executions much earlier than the WCEP(s) does [8]. For the example program shown in Fig. 1(b), there exist 51 different execution paths. While the WCEP p_{worst} takes 200×10^6 cycles, 12 of 51 possible execution paths take less than 100×10^6 cycles. For such short execution paths, the workload-variation slack times are generated. If we were able to identify them in the early phase of its execution, the clock speed can be lowered substantially, thus saving a significant amount of energy consumption. Consider the path $p_1 = (b_1, b_{\text{call1}}, b_8, b_9, b_{11}, b_{\text{if}}, b_{\text{call4}}, b_8, b_{10}, b_{11}, b_7)$ of Fig. 1(b) whose execution takes 95×10^6 cycles. In the ideal case, the execution can start with a clock speed of 47.5 MHz without violating the 2-s deadline if it can be perfectly predicted that the actual execution path will be p_1 before the processor starts b_1. Unfortunately, it is not generally known in advance which execution path will be taken by the next program execution. Therefore, it cannot start with the 47.5-MHz clock speed, although this will improve the energy efficiency significantly.

The solution for this problem is to adjust the clock speed within the task depending on the workload variations. For example, when the program control flow follows the execution path p_1 of Fig. 1(b), the clock speed can be reduced at edge (b_1, b_{call1}) because this control flow dose not follow the WCEP. In the proposed IntraDVS algorithm, the authors identify the appropriate program locations where the clock speed should be adjusted and insert the clock and voltage scaling codes to the selected program locations at compile time. The branching edges of the CFG, i.e., branch or loop statements, are the candidate locations for inserting voltage scaling codes because that is where changes of the remaining execution cycles occurred.

C. Problem Modeling

Consider a real-time task τ with the deadline D. The task τ is represented by its CFG G_τ. If the task τ has N basic blocks, $b_1, b_2, \ldots, b_N$, G_τ consists of N nodes. (It was assumed that b_1 is the entry basic block of the task τ.) Each basic block b_i is associated with its basic block information (BBI) structure. The BBI structure $\text{BBI}(b_i)$ of the basic block b_i consists of three entries: $C_{\text{EC}}(b_i)$, $S(b_i)$, and $E(b_i)$. $C_{\text{EC}}(b_i)$ denotes the number of clock cycles[4] needed to execute b_i. $S(b_i)$ represents the processor speed in clock frequency at which b_i is executed. (It was assumed that the supply voltage is proportional to the clock speed.) $E(b_i)$ is defined as $C_{\text{EC}}(b_i) \times S(b_i)^2$. $E(b_i)$ is the metric for the energy consumption during the execution of b_i.

Similar notations are defined for execution paths. p_i denotes an execution path of a task τ. p_i can be expressed as a sequence of basic blocks. $C_{\text{EC}}(p_i)$ represents the number of execution cycles when p_i is executed. For paths, a notation Φ^{b_i} is used, which means the set of all the partial execution paths starting from basic block b_i.

The IntraDVS algorithm is used to find out how much the clock speed should be changed at each edge (b_i, b_j) in the CFG of a target program to minimize the total energy consumption of the program, satisfying the timing constraint of the program. That is, we should find the value $r_{i,j} = S(b_j)/S(b_i)$ for each edge (b_i, b_j). This ratio is called the speed update ratio (SUR).

[4]Note that the BBI definition above is represented in execution cycles, instead of execution time. This is because, as the clock speed is adjusted on a variable-voltage processor, the execution time is changing for a given basic block, but the number of execution cycles remains constant. Given the number of execution cycles, the execution time can be computed by multiplying the clock cycle time.

For an execution path $p_m = (b_1, \ldots, b_{n_m})$, $S(b_i)$ can be represented as

$$S(b_i) = S_0 \prod_{k=1}^{i} r_{k-1,k} \; (S(b_i) \geq S_{\min} \text{ and } S(b_i) \leq S_{\max})$$

where S_0 is the initial clock speed at the start of a program and $S_{\min}$ ($S_{\max}$) is the minimum (maximum) value of the clock speed provided by the target variable-voltage processor. $E(b_i)$ can also be denoted as

$$E(b_i) = C_{\mathrm{EC}}(b_i) S(b_i)^2 = C_{\mathrm{EC}}(b_i) \left(S_0 \prod_{1}^{i} r_{k-1,k} \right)^2.$$

Then, the energy consumption during the execution of the path p_m is proportional to

$$E(p_m) = \sum_{i=1}^{n_m} E(b_i) = \sum_{i=1}^{n_m} \left(C_{\mathrm{EC}}(b_i) \left(S_0 \prod_{1}^{i} r_{k-1,k} \right)^2 \right).$$

From this formula, the target function can be represented to minimize as

$$\sum_{\forall p_m \in \Phi^{b_1}} E(p_m) \mathrm{prob}(p_m) = \sum_{\forall p_m \in \Phi^{b_1}} \left[\sum_{i=1}^{n_m} \left(C_{\mathrm{EC}}(b_i) \left(S_0 \prod_{1}^{i} r_{k-1,k} \right)^2 \right) \mathrm{prob}(p_m) \right] \quad (1)$$

where $prob(p_m)$ is the probability that the path p_m is executed among all the paths in Φ^{b_1}.

There is a timing constraint for this problem. Since the target program τ should be completed before the deadline D, the timing constraint can be denoted as

$$\forall p_m, \quad \sum_{i=1}^{n_m} \frac{C_{\mathrm{EC}}(b_i)}{S(b_i)} = \sum_{i=1}^{n_m} \frac{C_{\mathrm{EC}}(b_i)}{S_0 \prod_1^i r_{k-1,k}} \leq D. \quad (2)$$

Using a simple inference, it can be concluded that the optimal solution satisfies

$$\forall p_m, \quad \sum_{i=1}^{n_m} \frac{C_{\mathrm{EC}}(b_i)}{S_0 \prod_1^i r_{k-1,k}} = D$$

$$\forall p_m, \quad \sum_{i=1}^{n_m} \frac{C_{\mathrm{EC}}(b_i)}{D \prod_1^i r_{k-1,k}} = S_0.$$

So, S_0 can be estimated when the $r_{i,j}$ for each edge (b_i, b_j) is determined.

Since (1) is a nonlinear equation for $r_{i,j}$, this problem is a nonlinear program (NLP) problem. Generally, there is no polynomial time algorithm for the NLP problem. So, a heuristic algorithm similar to the gradient descent method is proposed.

Fig. 2 shows the heuristic search algorithm. For each $r_{i,j}$, the authors set the initial value of it to 1, which means that there is no speed update at the corresponding edge, and constitute the set R that has all edges in G_τ of the target program. $r_{i,j}$ is changed as the amount of Δr, which is a very small number between 0 and 1, during the successive iterations in Fig. 2. In each iteration, we first find the edge (b_i, b_j) having the largest ΔE. ΔE means the energy gain when $r_{i,j}$ is changed to $r'_{i,j} = r_{i,j} \times \Delta r$. From (1), ΔE be can represented as

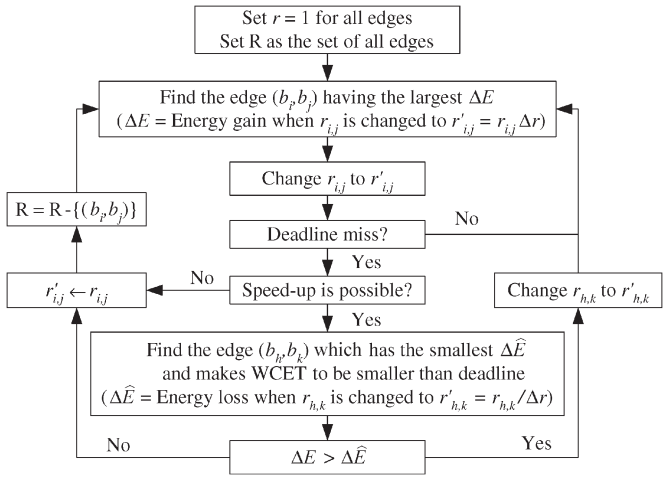

Fig. 2. Heuristic search algorithm for the IntraDVS problem.

$$\Delta E(r_{i,j}) = S_0(1 - \Delta r) \times \sum_{p_m \in \Phi^{b_j}} \mathrm{prob}(p_m) \sum_{k=j}^{n_m} C_{\mathrm{EC}}(b_k) S(b_k)^2.$$

After changing the value of $r_{i,j}$, it should be checked whether the timing constraint of (2) is satisfied in spite of the increased execution time of p_m. If there is no deadline miss, the same process is tried again.

When there is a deadline miss, two kinds of approaches can be chosen. The first approach is to admit that there is no speed up, that is, SURs are always smaller than 1. In this case, the clock speed only decreases as a target program executes. So, there is no chance to solve the deadline miss problem by increasing the clock speed after (b_i, b_j). $r'_{i,j}$ should be restored to $r_{i,j}$ and eliminate the corresponding edge (b_i, b_j) from the set of candidate edges R. The second approach is to admit that there is a speed-up. In this case, since the clock speed can be increased by an edge after (b_i, b_j), $r_{i,j}$ can be decreased despite the deadline miss. To maintain the selected edge (b_i, b_j) in R, the authors should find the other edge, say (b_h, b_k), that has the smallest $\Delta\widehat{E}$. $\Delta\widehat{E}$ means the energy loss when $r_{h,k}$ is changed to $r'_{h,k} = r_{h,k}/\Delta r$. After changing $r_{h,k}$, the potential energy gain should be checked to see whether ΔE is larger than $\Delta\widehat{E}$. If there is no edge satisfying such a condition, $r'_{i,j}$ is restored to $r_{i,j}$ and the corresponding edge (b_i, b_j) is eliminated from R.

This heuristic algorithm shows the following features: 1) The nearest edge from the entry basic block is first selected from R because it has the largest value of ΔE. After the corresponding SUR of the selected edge is fixed, the following edges are

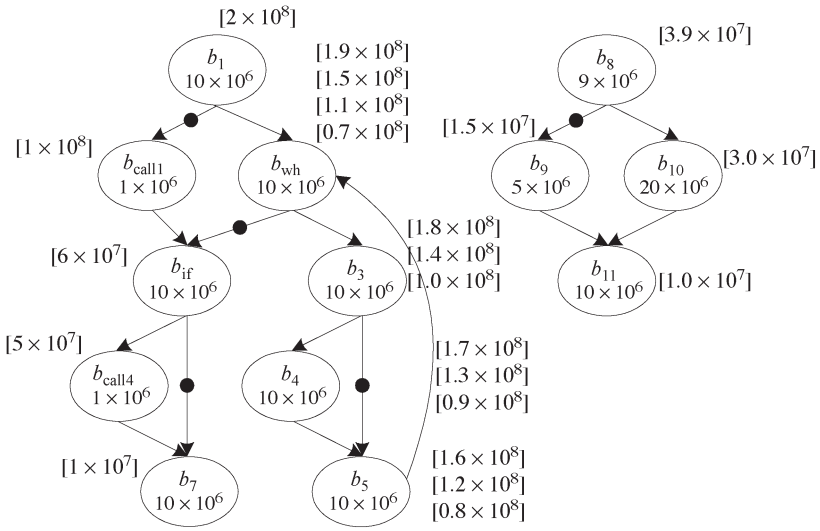

Fig. 3. RWEP-based CFG G_P^{RWEP}.

processed; 2) The nearest edge from the exit basic block is first selected to increase its SUR; 3) In the constraint that SURs should be smaller than 1, $S(b_i)$ is close to the value under which the target processor can execute $\mathrm{MAX}_{p_j \in \Phi^{b_i}}[C_{\mathrm{EC}}(p_j)]$ cycles until the deadline. $\mathrm{MAX}_{p_j \in \Phi^{b_i}}[C_{\mathrm{EC}}(p_j)]$ is the remaining WCEC (RWEC) of b_i. Namely, the SURs are optimal to the remaining WCEP (RWEP); and 4) Without such a constraint, the SURs are determined so that the average energy consumption is minimized, satisfying the deadline constraint. In this case, the determined speed is near the optimal speed value for the ACEP.

From these features, the heuristic algorithm can be modified to a less complex algorithm. An execution path that the control flow will follow is predicted. There can be several methods how to predict the execution path. A simple method is to use the WCEP. Once the execution path is predicted, the initial clock frequency and its corresponding voltage are set, assuming that the task execution will follow the predicted execution path. The predicted execution path is called as the reference path because the clock speed is determined based on the execution path.

When the actual execution deviates from the (predicted) reference path (say, by a branch instruction), the clock speed can be adjusted depending on the difference between the remaining execution cycles of the reference path and that 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.

In the actual implementation of IntraDVS, the reference path does not need to be maintained. To implement the IntraDVS algorithm efficiently, the appropriate program locations where the clock speed should be raised or lowered relative to the current clock speed are identified using a static program analysis technique. For run-time clock speed adjustment, it inserts voltage scaling codes to the selected program locations at compile time. The branching edges of the CFG, i.e., branch or loop statements, are the candidate locations for inserting voltage scaling codes because that is where the prediction miss for the reference path occurred. They are called voltage scaling edges (VSEs) because the clock speed and the voltage are adjusted at these edges. At each VSE (b_i, b_j), the clock speed is determined by the predicted remaining execution cycles (PRECs) of b_i, $C_{\mathrm{PREC}}(b_j)$. The value of $C_{\mathrm{PREC}}(b_j)$ depends on the prediction algorithm.

There are two issues in the IntraDVS. One is how to predict the reference path. Depending on the prediction method, the IntraDVS framework can be implemented into different IntraDVS algorithms. In this paper, two kinds of reference paths were adopted, i.e., RWEP and remaining ACEP (RAEP). Based on the prediction method, there are two different algorithms, i.e., RWEP-based IntraDVS and RAEP-based IntraDVS. In the former, the clock speed is monotonically decreased at all the VSEs. This corresponds to the case where speed-up is not admitted. On the contrary, in the latter, the clock speed may be increased as well at some VSEs. In this case, VSEs are classified into up-VSEs and down-VSEs. The clock speed is increased at an up-VSE while it is decreased at a down-VSE. Another issue is how to select VSEs. In selecting VSEs, the timing overhead due to voltage transition and the inserted codes should be considered. The solutions for these issues are provided in Section IV.

IV. Details of the Proposed IntraDVS Algorithms

A. RWEP-Based IntraDVS Algorithm

In the RWEP-based IntraDVS, $C_{\mathrm{RWEC}}(b_i)$ was used for the PRECs. Fig. 3 shows an augmented CFG G_P^{RWEP} with $C_{\mathrm{RWEC}}(b_i)$ values for the RWEP-based IntraDVS. Using the static timing analysis tools, $C_{\mathrm{RWEC}}(b_i)$ can be computed for each basic block b_i and the graph G_P^{RWEP} can be constructed statically. For the basic blocks related to the while loop (i.e.,

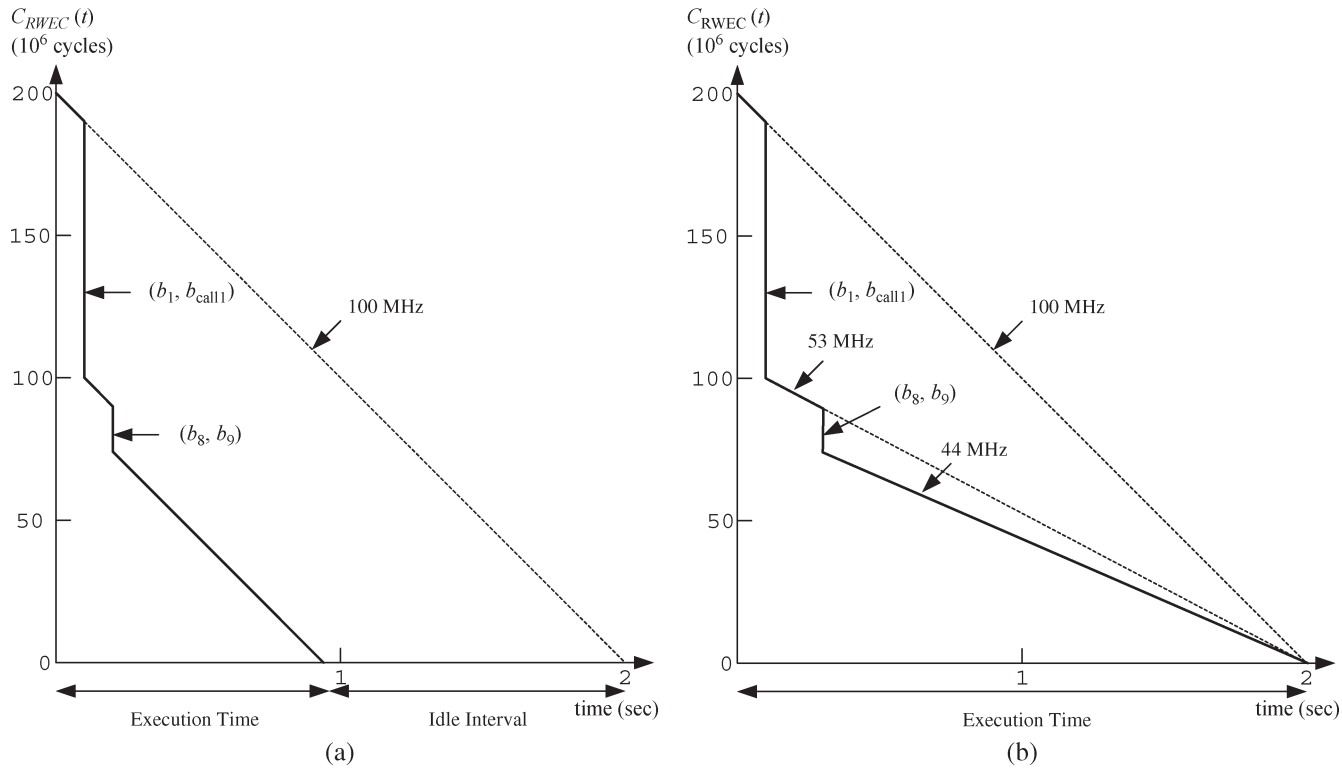

Fig. 4. Changes of $C_{\mathrm{RWEC}}(t)$ over different speed scaling algorithms: (a) no IntraDVS and (b) RWEP-based IntraDVS.

b_{wh}, b_3, b_4, b_5), the corresponding nodes are associated with multiple $C_{\mathrm{RWEC}}(b_i)$ values, reflecting the maximum three iterations of the while loop.

With the graph G_P^{RWEP}, VSEs that drop RWECs faster than the current execution rate can be identified. For example, in Fig. 3, five VSEs are identified, i.e., (b_1, b_{call1}), (b_{wh}, b_{if}), (b_{if}, b_7), (b_3, b_5), and (b_8, b_9). In Fig. 3, these edges are marked by the symbol •. When the thread of execution control branches to the next basic block through one of the VSEs, say (b_1, b_{call1}), the clock speed can be lowered because the remaining work is reduced by the difference between $C_{\mathrm{RWEC}}(b_{\mathrm{wh}})$ and $C_{\mathrm{RWEC}}(b_{\mathrm{call1}})$. By reducing the clock speed so that $C_{\mathrm{RWEC}}(b_{\mathrm{call1}})$ cycles can be completed exactly at the deadline, the proposed technique always meets the required timing constraint. Since the voltage scaling decisions are made at compile time, there exists no run-time overhead directly related to the selection of VSEs. In addition, the compile-time static analysis procedure does not require special programmer interventions other than the ones typically required in developing normal hard real-time programs (e.g., the maximum number of loop iterations).

At the entry basic block b_1, the starting speed is set to C_{WCEC}/D, where C_{WCEC} is the WCEC of the whole program. When $C_{\mathrm{RWEC}}(t)$ is denoted as the RWEC at time t, $C_{\mathrm{RWEC}}(t)$ is linearly decreased at the rate of clock speed along with the program execution, as far as the execution follows the WCEP p_{worst}. However, if the execution deviates from the basic block b_i in the WCEP p_{worst} to other basic block b_j not included in p_{worst}, $C_{\mathrm{RWEC}}(t)$ drops by the difference between $C_{\mathrm{RWEC}}(b_i) - C_{\mathrm{EC}}(b_i)$ and $C_{\mathrm{RWEC}}(b_j)$ after the execution of b_i is completed.

Fig. 4 shows how $C_{\mathrm{RWEC}}(t)$ dynamically changes as the path $p_1 = (b_1, b_{\mathrm{call1}}, b_8, b_9, b_{11}, b_{\mathrm{if}}, b_{\mathrm{call4}}, b_8, b_{10}, b_{11}, b_7)$ of the example program P shown in Fig. 3 is executed. In Fig. 4(a), where no speed scheduling is used, $C_{\mathrm{RWEC}}(t)$ drops at two edges, (b_1, b_{call1}) and (b_8, b_9). Since no speed scheduling is used, $C_{\mathrm{RWEC}}(t)$ is decreased at a rate of 100 MHz, resulting in a slack time interval of 1.05 s. Fig. 4(b) shows the effect of speed scheduling for the same execution path assuming that there is no voltage transition overhead. When RWEC drops, the minimum processor speed that can complete the remaining program execution before the deadline also drops. Thus, processor speed is changed from 100 to 53 MHz when $C_{\mathrm{RWEC}}(t)$ drops right after the execution of b_1 is completed. When $C_{\mathrm{RWEC}}(t)$ drops right after b_8, speed is also changed due to the same reason. $C_{\mathrm{RWEC}}(t)$ is dropped vertically at VSEs in Fig. 4. The number of the reduced cycles of $C_{\mathrm{RWEC}}(t)$ at VSEs is denoted as C_{saved}.

Theoretically, since IntraDVS can fully exploit all workload-variation slack times, all of the task executions are completed exactly at the deadline. However, some slack time can be generated in real variable-voltage processors even though the IntraDVS technique is used. This is because variable-voltage processors provide only finite numbers of clock/voltage levels and require voltage transition times to change the clock/voltage level. These two factors prevent IntraDVS from adjusting the clock/voltage level at all VSE candidates (i.e., branching edges).

Fig. 5 compares how the speed and the voltage change depending on whether the IntraDVS is used or not. Assume that no energy is consumed in an idle state. When the execution follows the path p_1, the energy consumption ratio of Fig. 5(b) to

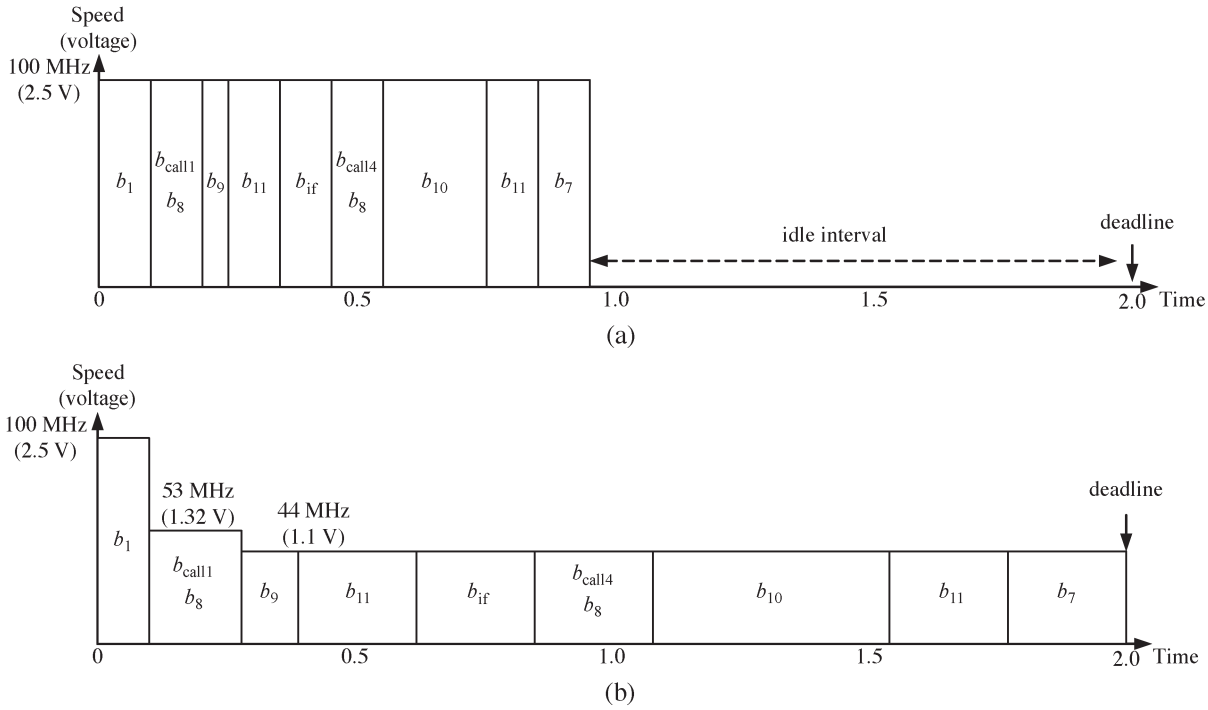

Fig. 5. Speed and voltage changes: (a) without IntraDVS and (b) with the RWEP-based IntraDVS.

Fig. 5(a) is 0.288. With the IntraDVS, the energy consumption is reduced by 71.2%.

B. The Selection of VSEs and Code Transformation

To adjust the clock speed and voltage at run time, the VSEs should be selected at compile time considering the saved cycles and the overhead cycles. Considering its behavior, VSEs are classified into B-type VSEs and L-type VSEs.

1) B-Type VSEs: A B-type VSE corresponds to the CFG edge between two basic blocks that are part of conditional statements such as the `if` statement. For the `if` statement, the WCET is predicted to be the larger of two execution times, one for the `then` path and the other for the `else` path. Assume that the condition of the `if` statement is evaluated in $b_{\rm cond}$, the `then` path starts from $b_{\rm then}$ and the `else` path starts from $b_{\rm else}$. If the condition of the `if` statement is evaluated to be true and the `then` path is shorter than the `else` path, $C_{\rm RWEC}(t)$ is decreased by $[C_{\rm RWEC}(b_{\rm else}) - C_{\rm RWEC}(b_{\rm then})]$. In this case, the speed can be decreased before the $b_{\rm then}$ block is executed by a ratio of $C_{\rm RWEC}(b_{\rm then})/C_{\rm RWEC}(b_{\rm else})$. This value is a SUR and is represented by $r(b_{\rm cond}, b_{\rm then})$.

In adjusting the clock/voltage at VSEs, several instructions are required other than the voltage-changing instruction [`change_f_V`($f_{\rm clk}$)]. The authors denote the number of cycles needed for these extra instructions at a B-type VSE as $C_{{\rm VSO}_B}$. The total number of overhead cycles $C_{{\rm overhead}_B}$ for a B-type VSE, therefore, is given by $C_{\rm VTO} + C_{{\rm VSO}_B}$. The SUR $r(b_i, b_j)$ for a B-type VSE (b_i, b_j) is calculated as

$$r(b_i, b_j) = \frac{C_{\rm RWEC}(b_j)}{C_{\rm RWEC}\left({\rm succ}_{\rm worst}(b_i)\right) - C_{{\rm overhead}_B}} \qquad (3)$$

where ${\rm succ}_{\rm worst}(b_i)$ is the basic block b_k that is an immediate successor of b_i and has the largest $C_{\rm RWEC}(b_k)$ among all the successors of b_i. If $C_{\rm RWEC}(b_j) \geq C_{\rm RWEC}[{\rm succ}_{\rm worst}(b_i)] - C_{{\rm overhead}_B}$, that is, $r(b_i, b_j) \geq 1$, the edge (b_i, b_j) is not selected as a VSE. For a VSE between b_i and b_j, a SUR $r(b_i, b_j)$ is multiplied to the current speed before b_j starts its execution. For example, assuming $C_{{\rm overhead}_B}$ as 0, $S(b_{\rm call1})$ in Fig. 3 is changed from 100 to 53 MHz [$= 100$ MHz $(100 \times 10^6/190 \times 10^6)$].

2) L-Type VSEs: Although WCEC is predicted assuming that a loop will be iterated by the user-provided maximum number of loop iterations, the loop is generally iterated smaller times than the maximum loop bound. In this case, slack time occurs and clock speed can be scaled down. The authors call this type of scaling L-type scaling. L-type VSEs correspond to the loop-exit edges in a CFG. In the L-type scaling, the number of saved cycles $C_{\rm saved}$ for a loop l is given by

$$C_{\rm saved}(l) = C_{\rm WCEC}(l)\,(N_{\rm worst}(l) - N_{\rm exec}(l)) \qquad (4)$$

where $C_{\rm WCEC}(l)$ is the number of WCECs to execute the loop l once, $N_{\rm worst}(l)$ is the number of user-provided maximum loop bound value for the loop l, and $N_{\rm exec}(l)$ is the number of actual loop iterations measured at run time. Consider the edge ($b_{\rm wh}$, $b_{\rm if}$) in Fig. 3. Assuming $N_{\rm exec}(l) = 1$, and $C_{{\rm overhead}_L} = 0$, $S(b_{\rm if})$ is updated as

$$\begin{aligned} S(b_{\rm if}) &= S(b_{\rm wh})\frac{C_{\rm RWEC}(b_{\rm if})}{C_{\rm RWEC}(b_{\rm if}) + C_{\rm saved}(l) - C_{{\rm overhead}_L}} \\ &= S(b_{\rm wh})\frac{60 \times 10^6}{60 \times 10^6 + 40 \times 10^6 \times (3-1)} \\ &= S(b_{\rm wh})r(b_{\rm wh}, b_{\rm if}). \end{aligned} \qquad (5)$$

When $S(b_{\rm wh})$ is 100 MHz, $S(b_{\rm if})$ is reduced to 43 MHz before executing $b_{\rm if}$.

Unlike a B-type VSE, calculating the SUR for an L-type VSE requires the run-time information such as $N_{\rm exec}(l)$.[5] The SUR may be larger than 1 depending on the value of $N_{\rm exec}(l)$ and $C_{{\rm overhead}_L}$. To avoid this problem, the authors select an L-type VSE in two phases. First, a loop-exit edge of a loop l is selected as an L-type candidate VSE if $C_{\rm WCEC}(l) > C_{{\rm overhead}_L}$, which means that if $N_{\rm exec}(l) < N_{\rm worst}(l)$, the SUR is always smaller than 1. When $N_{\rm exec}(l) = N_{\rm worst}(l)$, the speed is not changed but the timing behavior of an original program is changed due to the code inserted to check whether $N_{\rm exec}(l) = N_{\rm worst}(l)$ or not. Among the L-type candidates, the final L-type VSEs are chosen by the algorithm explained in Section IV-B-5. Although L-type VSEs are more complicated than B-type VSEs, the contribution of L-type VSEs on the overall energy reduction is much bigger since slack times from loop executions are generally much larger than those from conditional statements.

3) VSEs in Loops or Functions: The RWECs of nodes in loops or functions are changed depending on the iteration number of the loop or the location from which the function is called. So, the SUR of a VSE in loop or function cannot be represented as a fixed value. Instead, the SUR is represented by a formula using the run-time information as variables. For example, in Fig. 3, the SUR of (b_3, b_5) is represented as that shown at the bottom of the page, where l is the loop in Fig. 3. $r(b_3, b_5)$ is 0.957 at the first iteration but 0.947 at the second iteration of the loop l.

For a basic block b_i in the function f, the authors define $C_{\rm RWEC}(b_i)$ by the RWECs of b_i in the scope of the function f because the remaining execution cycles of b_i in the overall program are changed depending on where the function f is called. Therefore, $C_{\rm RWEC}(b_j)$ is needed to get the SUR of b_i, where b_j is the return point of the function f. For example, the SUR of (b_8, b_9) in Fig. 3 can be calculated as

$$r(b_8, b_9) = \frac{C_{\rm RWEC}(b_9) + C_{\rm RWEC}(b_{\rm if})}{C_{\rm RWEC}(b_{10}) + C_{\rm RWEC}(b_{\rm if})}$$

when the function is called from $b_{\rm call1}$.

However, denoting the SUR as the formula using the run-time information is an obstacle to select VSEs statically and insert the voltage scaling code. To avoid this problem, the authors propose a simple solution. All the possible SUR values of edge (b_i, b_j) are denoted as $r^1_{i,j}, \ldots, r^n_{i,j}$. For example, the VSE candidate (b_3, b_5) in Fig. 3 has three possible SUR values, $r^1_{3,5}$, $r^2_{3,5}$, $r^3_{3,5}$, because it can be executed three times in the loop. If any $r^k_{i,j}$ is smaller than 1, all other $r^h_{i,j}$ are also smaller than 1. This is based on the simple formula

$$\frac{a+l}{b+l} < 1 \text{ and } \frac{a-l}{b-l} < 1 \qquad \text{if } \frac{a}{b} < 1,\ a > l,\ \text{and } b > l.$$

If the authors represent any SUR $r^k_{i,j}$ of a VSE candidate in loop or function as a/b, the other SUR $r^h_{i,j}$ of the VSE can be denoted by $(a+l)/(b+l)$ or $(a-l)/(b-l)$, where l is the execution cycles between two instances of the VSE candidate. Since $(a+l)/(b+l)$ or $(a-l)/(b-l)$ is also smaller than 1 if $(a/b) < 1$, the authors can say that $r^h_{i,j}$ is also smaller than 1 if $r^k_{i,j} < 1$. This means that it is sufficient to check only one instance of the VSE candidate to know whether an edge in loop or function can be selected to a VSE.

4) Code Transformation: Since the SUR value of a VSE may not be determined as a fixed value at compile time as mentioned, the target program should calculate the SUR using the VSE information at run time. The VSE information consists of five elements, i.e., `Type`, `preRWEC`, `postRWEC`, `loopWCEC`, and `MI`. `preRWEC` and `postRWEC` are $C_{\rm RWEC}(b_i) - C_{\rm EC}(b_i)$ and $C_{\rm RWEC}(b_j)$ for a VSE (b_i, b_j), respectively. These values are used to calculate the SUR. For B-type VSEs, the SUR is `postRWEC`/`prePWEC`. For L-type VSEs, `loopWCEC` and `MI` are used additionally. `loopWCEC` is the WCECs of the L-type VSE's corresponding loop [$C_{\rm WCEC}(l)$ in (4)]. `MI` is the maximum number of loop iteration.

Fig. 6 shows the code examples for VSEs. Voltage scaling codes for VSE include a code segment that calculates the SUR and updates the current speed by multiplying with the SUR (code B and code L in Fig. 6). These voltage scaling codes are different in B-type VSE and L-type VSE. In L-type VSE, the voltage scaling codes calculate the SUR using the iteration number [$\mathrm{LoopIterNum}(b_{\rm wh})$] of the corresponding loop. Therefore, several codes are needed to maintain the iteration number of the loop (codes 2 and 3 in Fig. 6).

To calculate $C_{\rm RWEC}(b_i)$, where b_i is in the loop l, a code is needed to transfer the $C_{\rm post}(l)$ (the remaining execution cycles after a loop l) to the loop (code 1 in Fig. 6). The same code is inserted for functions (code 5 in Fig. 6). As the loop or function can be nested, $C_{\rm post}(l)$ is saved at a stack. When the loop or function is completed, the stack index (top) is decreased (codes 4 and 6 in Fig. 6).

5) Overall Selection Algorithm: While voltage scaling codes for B-type VSEs do not increase the $C_{\rm WCEC}$ of a given program, those for L-type VSEs can increase $C_{\rm WCEC}$ depending on the number of loop iterations executed. If a loop iterates its maximum number of iterations (i.e., the maximum number of loop iterations given by user) and the loop exit edge was selected as a candidate L-type VSE, $C_{\rm WCEC}$ of the program will increase by the number of cycles to execute the code checking the number of loop iterations. This increase,

[5]Note that the selection of L-type VSEs is done in compile time. The run-time information such as $N_{\rm exec}(l)$ is necessary when calculating the SUR.

$$r(b_3, b_5) = \frac{C_{\rm EC}(b_5) + C_{\rm EC}(b_{\rm wh}) + C_{\rm WCEC}(l)\,(N_{\rm worst}(l) - N_{\rm exec}(l)) + C_{\rm RWEC}(b_{\rm if})}{C_{\rm EC}(b_4) + C_{\rm EC}(b_5) + C_{\rm EC}(b_{\rm wh}) + C_{\rm WCEC}(l)\,(N_{\rm worst}(l) - N_{\rm exec}(l)) + C_{\rm RWEC}(b_{\rm if})}$$

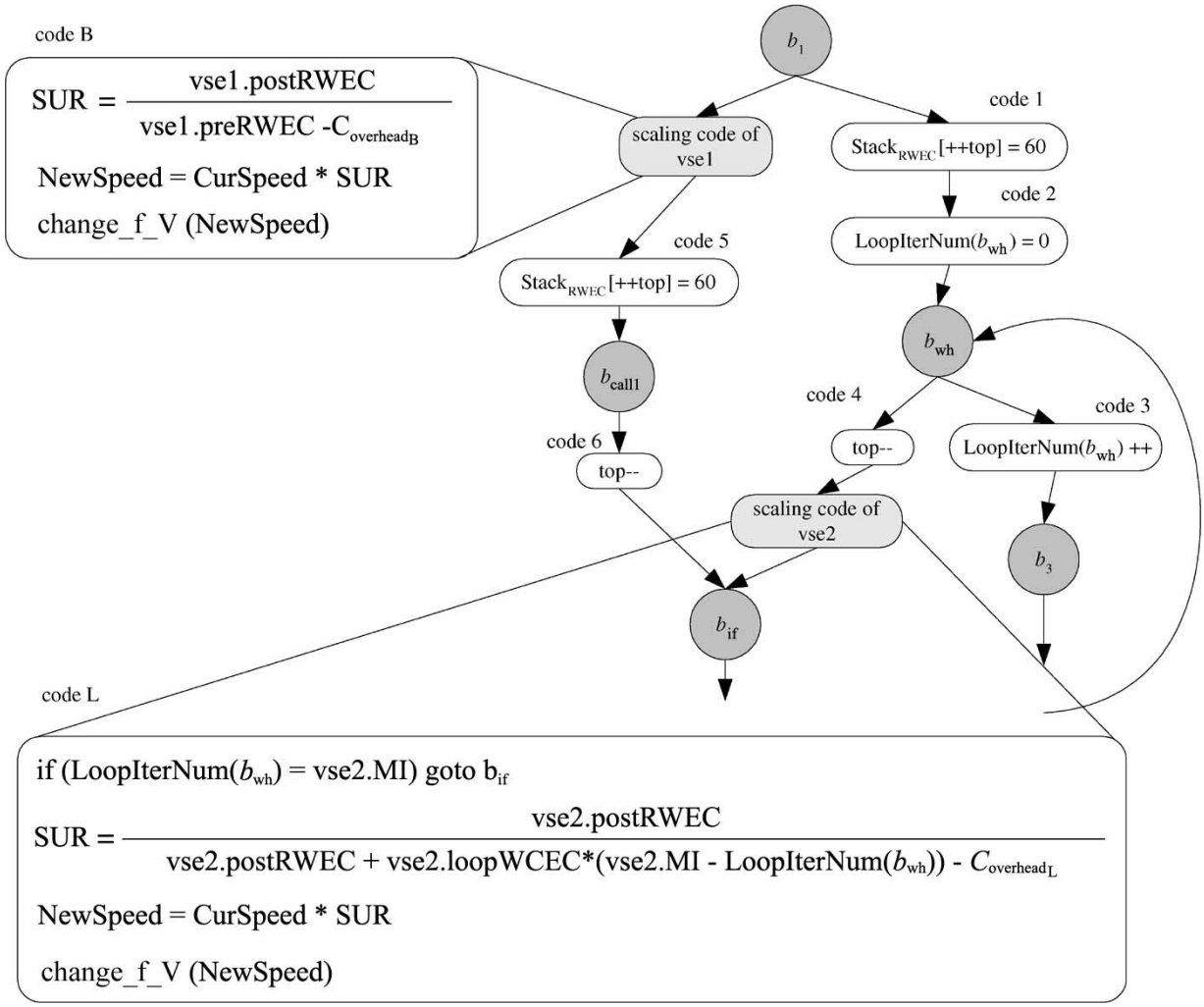

Fig. 6. Code generation for VSEs.

if accumulated, may make the modified program violate the timing constraint of the original program.

To avoid this problem, the final L-type VSEs are selected from the candidate L-type VSEs by the algorithm shown in Fig. 7. Assuming that the target processor can execute M cycles at its full speed within the given deadline interval, the authors first select the candidate VSEs using the selection algorithms in the previous section. Then the increase of WCECs is calculated. To calculate the increase, it should be known how much cycles are needed for voltage scaling codes. The authors denote these values as $C_{\mathrm{inc}}(B)$ and $C_{\mathrm{inc}}(L)$. If the total WCECs C_{WCEC} are larger than M, some candidate VSEs are excluded until the increase in C_{WCEC} will be smaller than M. How many candidate VSEs should be excluded is easily known with the values of $C_{\mathrm{inc}}(B)$ and $C_{\mathrm{inc}}(L)$. The VSE with little effect on energy reduction is preferred to be excluded. $C_{\mathrm{RWEC}}(b_i)$'s are recomputed after some candidate VSEs are excluded. When $C_{\mathrm{WCEC}} < M$ is satisfied, the final VSEs are determined.

Since the voltage scaling codes need several registers to determine the processor speed, the authors should know which registers are free at the VSE. If there is no free register, the voltage scaling codes should spill some live registers. This overhead is added to C_{inc}. Another simple method is to assign some dedicated registers only for voltage scaling codes, but it is inefficient when the number of registers is not sufficient.

C. RAEP-Based IntraDVS Algorithm

Although the RWEP-based IntraDVS 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 ACEP as a reference path. ACEP is defined to be an execution path with the largest possibility to be executed. The ACEP can be decided by observing the execution profile information.

It is easily understood that using ACEP instead of WCEP is more energy efficient. For a typical program, about 80% of the program execution occurs in only 20% of its code, which is called the hot paths [28]. To achieve high energy efficiency, an IntraDVS algorithm should be optimized so that these hot paths are energy efficient. If one of the hot paths is used as a reference path, 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 [29]. In this case, even other paths (that are not the hot paths) become

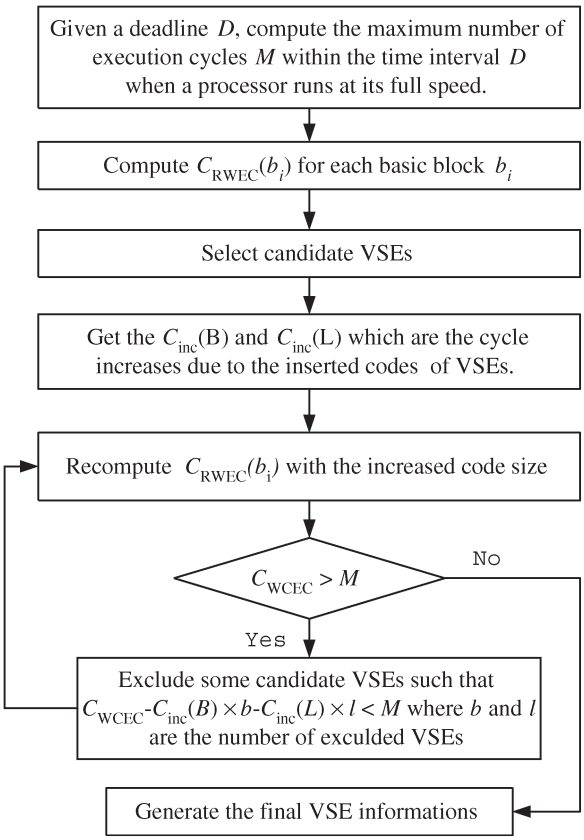

Fig. 7. Overall VSE selection algorithm.

more energy efficient because they can start with a lower clock speed than when the WCEP is used as a reference path.

In RAEP-based IntraDVS, ACEP is taken as the reference path since it is the best representative of the hot paths. Fig. 8 shows an RAEP-based CFG G_P^{RAEP} with $C_{\mathrm{RAEC}}(b_i)$ values that represent the remaining average-case execution cycles (RAECs) among all the paths that start from b_i. The bold edges in G_P^{RAEP} mean that it has a higher probability to be followed between either branching edges. In Fig. 8, the initial reference path is $(b_1, b_{\mathrm{call1}}, b_8, b_9, b_{11}, b_{\mathrm{if}}, b_{\mathrm{call4}}, b_8, b_9, b_{11}, b_7)$. With the reference path, $C_{\mathrm{RAEC}}(b_i)$ is computed. For example, $C_{\mathrm{RAEC}}(b_{\mathrm{if}}) = C_{\mathrm{EC}}(b_{\mathrm{if}}) + C_{\mathrm{RAEC}}(b_{\mathrm{call4}})$. At RAEP-based IntraDVS, there are up-VSEs (marked by $\circ$ in Fig. 8) as well as down-VSEs (marked by $\bullet$ in Fig. 8). Fig. 9 shows how the speed and the voltage change in a RAEP-based scheduling. The speed changed from 40 to 64 MHz at the edge (b_8, b_{10}) because this is an up-VSE with an SUR value of 1.6 $[= (40 \times 10^6)/(25 \times 10^6)]$. Compared to the RWEP-based IntraDVS algorithm, the RAEP-based IntraDVS algorithm can achieve more energy reduction if the execution path follows the reference path.

Although the RAEP-based scheduling is more energy effective than the RWEP-based scheduling, the pure RAEP-based approach cannot meet the timing requirements of hard real-time applications. This is because it does not satisfy the timing constraints for all the execution paths if ACEP is used as reference path. For example, consider the case when 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 fails to meet its deadline even if the processor runs at its maximum speed during the remaining paths.

To overcome the deadline miss problem of the pure RAEP-based IntraDVS algorithm, the reference path is modified whenever the deadline miss situations are recognized. Assume that the reference path is $p_{\mathrm{ref}} = (b_1, \ldots, b_i, b_{i+1}, \ldots, b_N)$, b_i is a branching node whose child 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 $S \times r(b_i, b_{\mathrm{miss}})$, is larger than the maximal clock speed (S_{max}) of the processor, it indicates that the deadline will be missed if the current execution branches to b_{miss}. This is because the remaining time T_R to the deadline is $T_R = (C_{\mathrm{RAEC}}(b_{i+1})/S)$ and $S_{max} \times T_R < C_{\mathrm{RAEC}}(b_{\mathrm{miss}})$. There are $M = [C_{\mathrm{RAEC}}(b_{\mathrm{miss}}) - S_{max} \times T_R]$ cycles that miss the deadline. In order to avoid the deadline miss, the authors increment $C_{\mathrm{RAEC}}(b_k)$ by M for all $k \leq i$. That is, the authors modify the reference path by adding a new virtual basic block b_v between b_i and b_{i+1} where $C_{\mathrm{EC}}(b_v)$ is set to M. The virtual basic block is used only to prevent the deadline miss during the speed assignment step at compile time and is not executed at run time.

Fig. 10(a) and (b) 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-μs deadline because the speed at (b_3, b_5) should be raised to 120 MHz (i.e., 60 MHz $\times$ 2). Because $10/3$ cycles[6] are missed from the deadline, a virtual block b_v is added between b_3 and b_4, as shown in Fig. 10(b). $C_{\mathrm{EC}}(b_v)$ is set to 4 $(= \lceil 10/3 \rceil)$. With the added b_v, $C_{\mathrm{RAEC}}(b_1)$ and $C_{\mathrm{RAEC}}(b_3)$ are modified to 34 and 24, respectively, and the SURs are recalculated. For example, $r(b_3, b_5)$ is modified to 1.43 $(= 20/14)$ from 2.

Fig. 10(c) and (d) shows the speed changes for the paths (b_1, b_3, b_4) and (b_1, b_3, b_5), respectively, where the RWEP-based IntraDVS, the RAEP-based IntraDVS, and the modified RAEP-based IntraDVS are compared. The modified RAEP-based scheduling is significantly more efficient than the RWEP-based scheduling for the hot paths [Fig. 10(c)], which dominates the overall energy efficiency. Moreover, it also satisfies the deadline requirement [Fig. 10(d)], while the pure RAEP-based scheduling algorithm does not.

D. Comparisons of RWEP-Based IntraDVS and RAEP-Based IntraDVS Algorithms

Two kinds of IntraDVS algorithms were proposed. The RAEP-based IntraDVS outperforms the RWEP-based IntraDVS in energy efficiency because the scheduled speed does not fluctuate much. However, the RAEP-based IntraDVS algorithm needs the reference path modification to guarantee the timing constraint as shown in the previous subsection, which is very complicated and time consuming. It also inevitably needs the profiling information, while the RWEP-based IntraDVS only requires the WCET analysis.

[6] $20 \text{ cycles} - 100 \text{ MHz} \times (10 \text{ cycles}/60 \text{ MHz}) = 10/3 \text{ cycles}$.

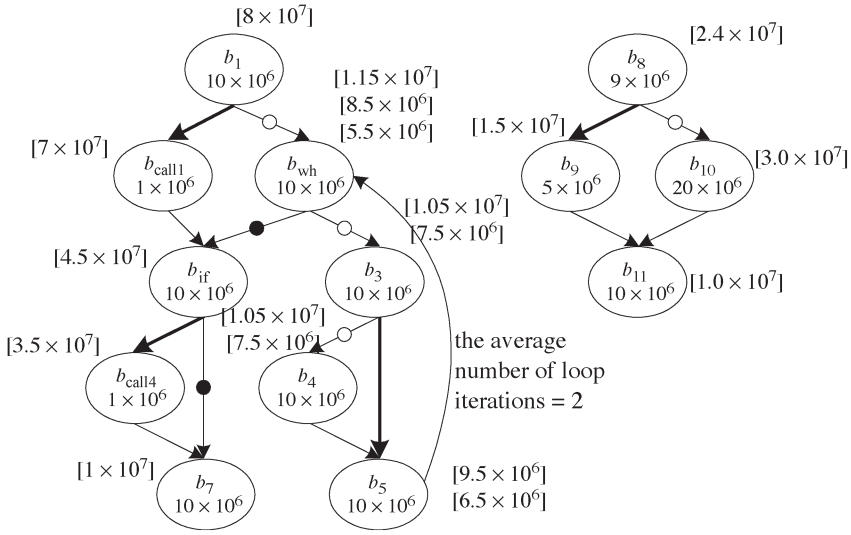

Fig. 8. RAEP-based CFG G_P^{RAEP}.

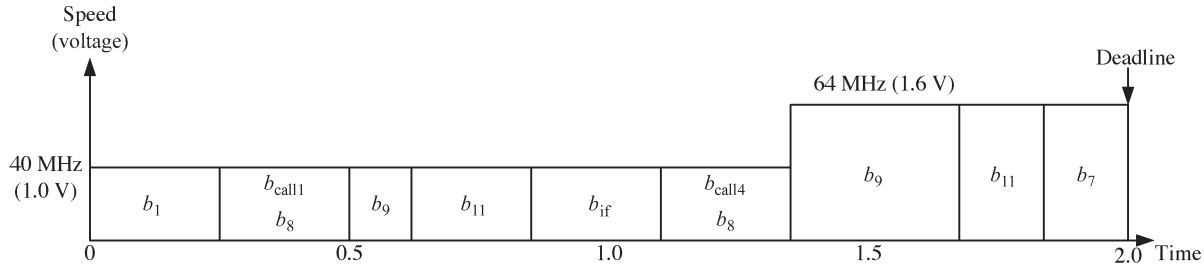

Fig. 9. Speed and voltage changes by the RAEP-based IntraDVS.

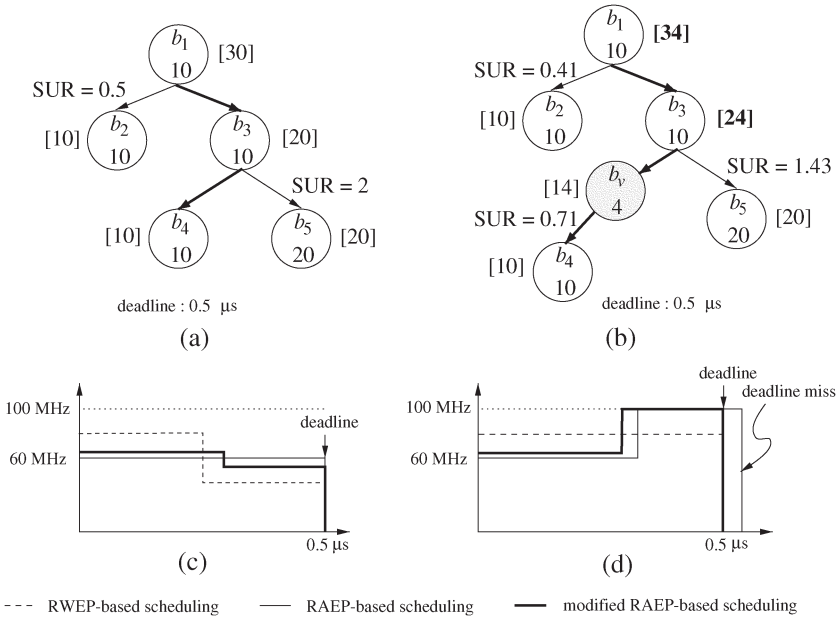

Fig. 10. Modified RAEP-based IntraDVS: (a) an original G_P^{RAEP}, (b) a modified G_P^{RAEP}, and (c) and (d) the speed change graphs for three IntraDVS algorithms for the paths (b_1, b_3, b_4) and (b_1, b_3, b_5), respectively.

Another problem of the RAEP-based IntraDVS is that the time slot between the release time and the deadline of a task should be determined statically. This does not matter in single-task environments. However, it is a serious problem in multitask environments since the time slot for a task is changed depending on the release time. Unfortunately, it prohibits from processing reference path modification at compile time. One solution for this problem is to prepare multiple configurations at each VSE for different values of the task's time slots.

E. Comparisons of Off-Line IntraDVS and On-Line IntraDVS Algorithms

In the proposed IntraDVS algorithm, there are some unselected VSEs because VSE is selected or neglected by considering the transition time overhead of speed change. Unfortunately, the slack times generated by these unselected VSEs cannot be exploited by the IntraDVS algorithm. This limitation comes from the fact that the proposed algorithm does not use the run-time timing information. The authors call such a speed assignment an off-line speed assignment method.

The described IntraDVS can be improved if the authors can get an elapsed time at run time. In order to reclaim these slack times, the new clock speed should be set to the remaining cycles divided by the remaining time (= deadline–elapsed time). This method is called an on-line speed assignment method. For the on-line speed assignment method, a target system should support efficient real-time counter accesses to get the elapsed time at run time.[7]

When the voltage transition overhead is small and the number of unselected VSEs is small, the off-line assignment method works relatively well compared to the on-line assignment method. On the other hand, when the voltage transition overhead is large, the on-line speed assignment becomes more effective than the off-line speed assignment because the number of unselected VSEs increases.

V. Experimental Results

A. AVS

Based on the proposed IntraDVS algorithm, the authors have developed the AVS, a software tool that automates the development of hard real-time programs on a variable-voltage processor. AVS takes a DVS-unaware (thus regular) program P and its timing requirements as inputs, and produces a DVS-aware low-energy program P_{DVS} that satisfies the same timing requirements. The converted program P_{DVS} contains a voltage scaling code that handles all the idiosyncrasy of scaling clock/voltage on a variable-voltage processor. Using AVS, DVS-unaware hard real-time programs can be converted to DVS-aware low-energy programs in a completely transparent fashion to software developers.

Fig. 11 shows the overall structure of the AVS. It imports a high-level language program (such as source codes) and timing requirements (such as a deadline), and converts them into a DVS-aware program. The converted program satisfies the same functional and temporal requirements of P, but it consumes much less energy than P. The AVS consists of four main modules, i.e., compiler, timing analyzer, VSE selector, and code transformer. The compiler surveys the input program structure and generates the inputs for the timing analyzer. The timing analyzer analyzes the timing behavior of the program and estimates the PRECs of all the basic blocks in the input program. To estimate the predicted execution cycles, the timing

[7]Although many embedded processors for real-time applications have an on-board real time clock (RTC), the resolution of RTC can be too low to use for IntraDVS. Moreover, the system call to access an RTC may incur a large overhead. So, the authors specify this assumption.

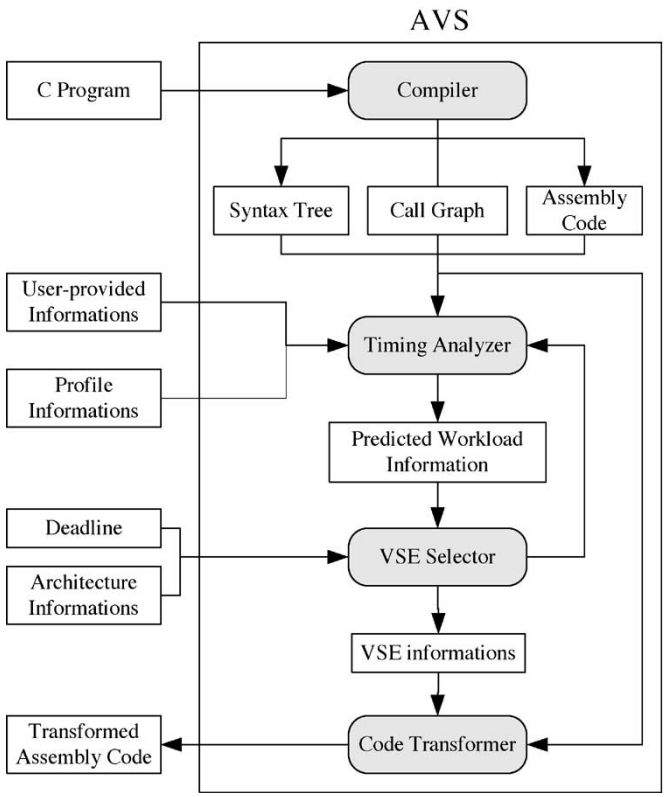

Fig. 11. AVS.

analyzer uses the user-provided information (e.g., the maximum number of loop iteration) and the profile information (for RAEP-based IntraDVS). A modified version of timing tools developed by Lim *et al.* [27] was employed. Their original timing tools estimate the WCET of a whole program traversing the program's syntax tree. Since AVS needs the RWEC of each basic block, the authors have modified the original timing tool to satisfy the purpose. The timing analyzer transfers the results to the VSE selector.

The VSE selector is responsible for the speed setting of the DVS. This module determines the locations for voltage scaling. For this work, it requires the deadline of a program and the processor's scaling information, i.e., voltage scaling overhead. The code transformer modifies the original program and generates the DVS-aware program. It inserts the voltage scaling codes at the selected voltage scaling locations. These codes were described in Section IV in detail.

B. Experiments of RWEP-Based IntraDVS

To evaluate the energy reduction performance of IntraDVS algorithms, the authors have experimented with an MPEG-4 video encoder and decoder. They experimented with both a simulation system and a real DVS-enabled system. The experimental results on a real DVS-enabled system are described in Section V-D. For a simulation, an energy simulator for the simulation experiments was developed. The energy simulator takes an assembly program and its execution trace as inputs and calculates the total energy consumption by emulating the program execution on the target variable-voltage processor. It was assumed that both DVS-aware and DVS-unaware systems

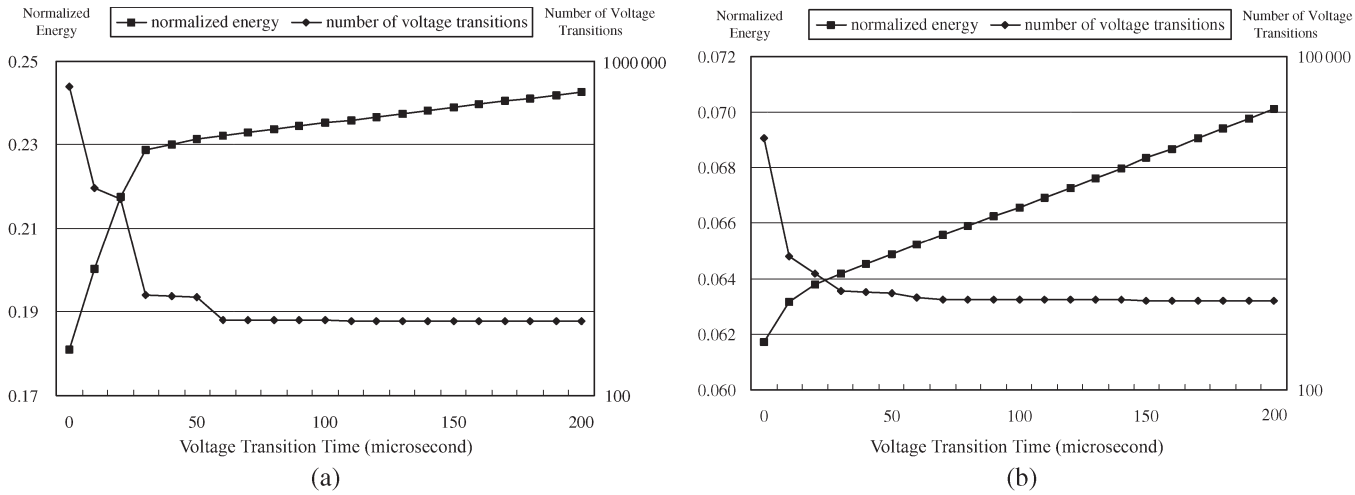

Fig. 12. Normalized energy consumption and the number of voltage transitions of the AVS-converted MPEG-4 encoder and decoder programs: (a) MPEG-4 encoder and (b) MPEG-4 decoder.

enter into a power-down mode when the system is idle. The supply voltage for a given clock frequency is obtained from $f_{\rm clk} = 1/T_D \propto (V_{\rm dd} - V_T)^{\alpha}/V_{\rm dd}$ [30], where $V_{\rm dd}$, V_T, and α are assumed to be 2.5 V, 0.5 V, and 1.3, respectively. Since recent frequency synthesizers and DC–DC converters achieve a clock/voltage transition time of less than 200 μs, the clock/voltage transition overhead $C_{\rm VTO}$ is assumed to be 0–20 000 cycles, corresponding to 0–200 μs of transition time with 100 MHz of clock frequency. For nonzero $C_{\rm VTO}$ values, the processor stops its execution and enters into a power-down mode during the clock/voltage transition.

Fig. 12(a) and (b) shows the energy consumption of the AVS-converted MPEG-4 encoder and decoder programs, respectively. Simulated results were normalized over the energy consumption of the original program running on a DVS-unaware system with the power-down mode. It was assumed that the power-down mode consumes 5% of the energy consumed in a normal mode [8]. The AVS-converted MPEG-4 encoder and decoder programs consume less than 25% and 7% of the original program, respectively. The large difference of energy efficiencies between the two programs is due to the different timing behaviors of the two programs. There is a large difference between WCET and average-case execution time (ACET) of the MPEG-4 decoder while WCET of the MPEG-4 encoder is relatively close to ACET. Fig. 12(a) and (b) also shows the number of voltage transitions that represents how many times voltage scaling codes were executed during the program execution. When $C_{\rm VTO} < 3000$ cycles $(= 30\ \mu{\rm s})$ in the MPEG-4 encoder, the number of voltage transitions decreases sharply and the energy consumption increases rapidly. When $C_{\rm VTO} > 5000$ cycles $(= 50\ \mu{\rm s})$ in both the MPEG-4 encoder and decoder, the energy consumption does not increase rapidly. This is because the number of discarded VSEs (due to clock/voltage transition overhead) is small.

The number of VSEs, which represents how many copies of voltage scaling codes were inserted into the AVS-converted program, indicates the degree of code size increment by inserting voltage scaling codes using in-line expansions. For the AVS-converted MPEG-4 encoder and decoder programs, about 20 VSEs are inserted when $C_{\rm VTO} > 5000$ cycles, meaning that insertion of voltage scaling code hardly increases the total code size. This is because a small number of VSEs occupy quite a large portion of the total power reduction.

C. Experiments of RAEP-Based IntraDVS

To compare the power reduction performance of the RAEP-based IntraDVS algorithm with the RWEP-based IntraDVS algorithm, the authors have experimented with an MPEG-4 video encoder. In the RAEP-based IntraDVS, the probability of branch edges and the average number of loop iterations in a CFG of the MPEG-4 video encoder are estimated using the profiled information. A probability of 0.5 is assigned to the branch edges for which the authors cannot collect the execution profiles with sample test bit streams.

Fig. 13(a) shows how the normalized starting speed changes over various slack factor values. The slack factor, defined by $(\text{deadline} - WCET/\text{deadline})$, represents the fraction of time that a processor becomes idle after WCET. The execution times of modified ACEPs (by the procedure described in Section IV-C) for the MPEG-4 encoder are 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 IntraDVS algorithm.

Fig. 13(b) compares the energy consumption of two IntraDVS 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 encoder, the modified RAEP-based IntraDVS algorithm reduces the energy consumption up to 34% over the RWEP-based IntraDVS algorithm.

Note that there is a large gap between the energy consumption of RWEP-based and RAEP-based IntraDVS 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 IntraDVS, there are many execution paths that can still take advantage of the RAEP-based speed settings. That is, in order to meet the timing constraint, virtual blocks are

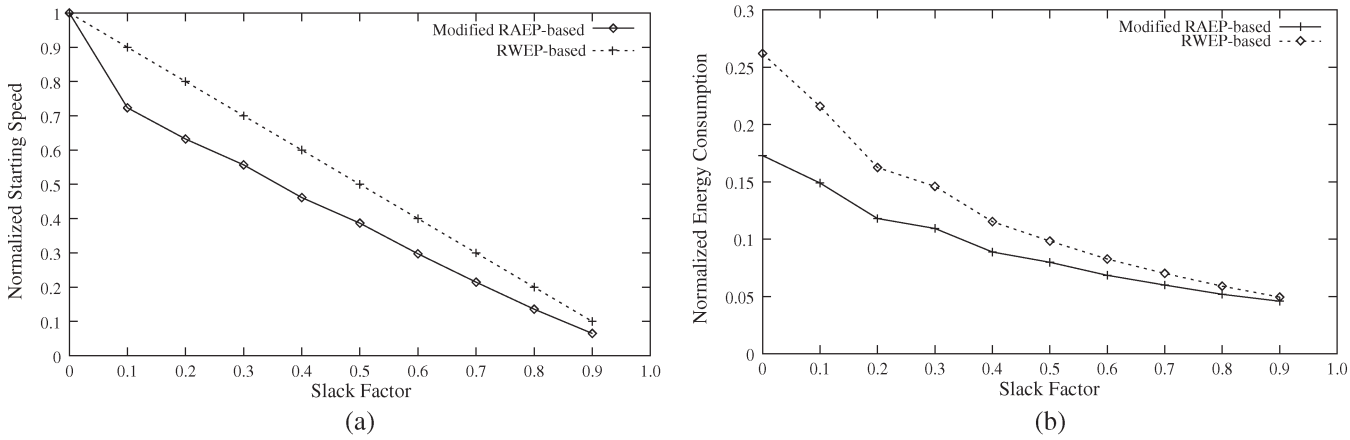

Fig. 13. Normalized starting speed and energy consumption of the RWEP-based IntraDVS and the RAEP-based IntraDVS versus the slack factor: (a) starting speed and (b) energy consumption.

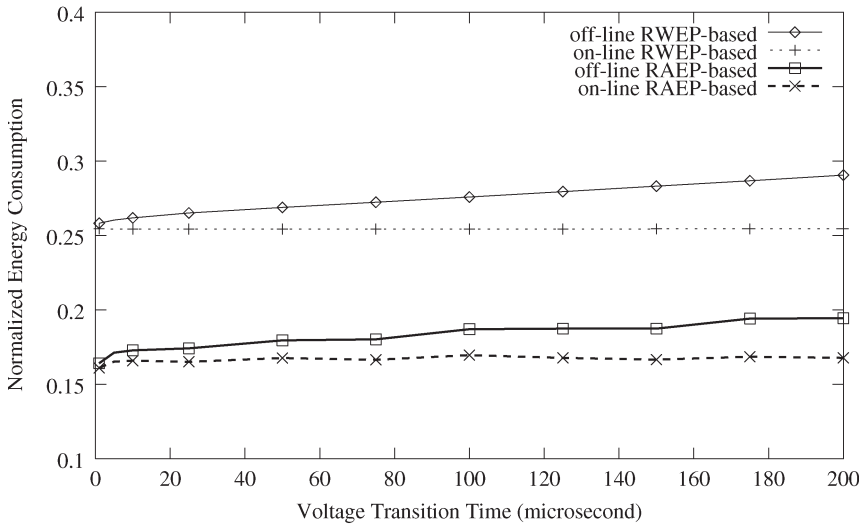

Fig. 14. Normalized energy consumptions of the on-line and off-line speed assignment methods (varying the threshold value).

added so that the initial speed is set to the same speed as in the RWEP-based IntraDVS algorithm. However, the (partial) paths following the virtual blocks can take advantage of the RAEP-based speed settings. As the slack factor increases, the energy consumption gap decreases because the supply voltages of both IntraDVS 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.

Fig. 14 shows the energy consumption of the off-line speed assignment and the on-line speed assignment described in Section III-B.The authors assumed that on-line speed assignment does incur additional 40 overhead cycles. When the voltage transition time is small, there are little differences in energy consumption between the on-line and the off-line speed assignment methods. However, the difference increases up to 10% as the voltage transition time increases because a large voltage transition time deselects more VSE candidates.

D. Experiments on a Real DVS-Enabled System

For experiments on a real DVS-enabled system, the authors used Itsy pocket computer v2.6 from Compaq [24] as their experimental platform. The platform is equipped with a StrongARM SA-1100 processor as the main processor. The SA-1100 processor uses the phase-locked loop (PLL), allowing to change the CPU core frequency to one of 11 levels between 59.0 and 226.4 MHz. Furthermore, Itsy v2.6 has a programmable core voltage regulator; the supply voltage can scale to one of 30 levels between 1.00 and 2.00 V. To change the clock and the voltage level, there is an overhead time during change. The overhead time is different depending on the current and target values of clock level and is 189 μs at maximum. Itsy runs the Linux operating system (ver. 2.0.30) with a kernel support for dynamic voltage scaling. Applications can access the DVS function by the ioctl system call to the "/dev/clkspeed" device file.

The RWEP-based IntraDVS algorithm was used for this experiment. Since it is difficult to establish the execution time model of a StrongARM SA-1100 for the timing analyzer module in AVS, the authors analyzed the timing behavior of a target application by the hybrid method that uses both the static analysis technique and the execution time profiling on the Itsy platform. Fig. 15 shows the flow of experiment for the Itsy platform. The target application (e.g., mpeg4dec.c) is compiled into

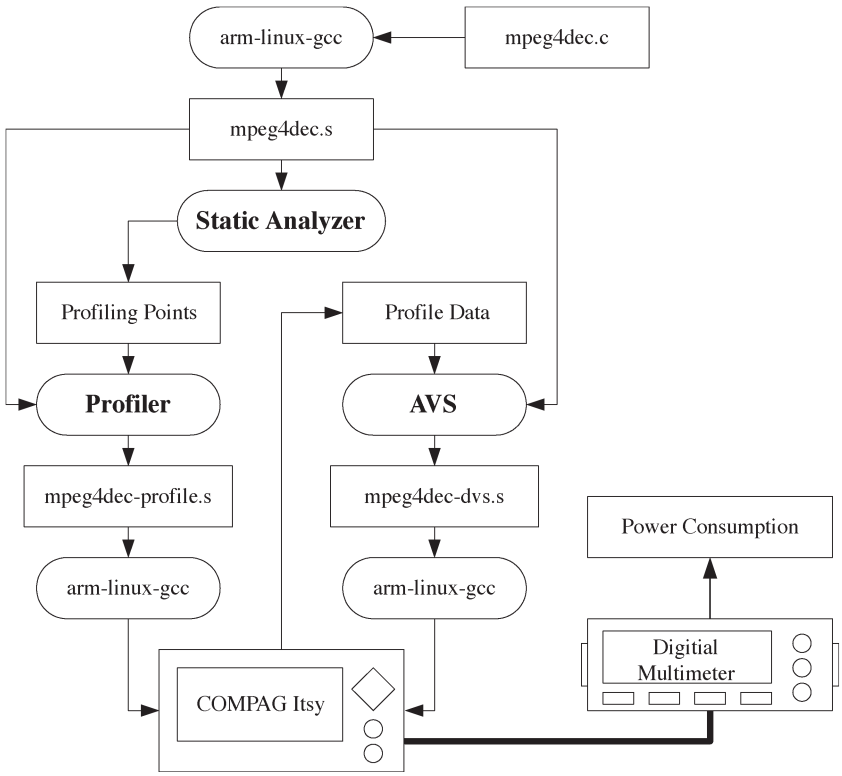

Fig. 15. Flow of experiment for the Itsy platform.

an assembly code (mpeg4dec.s) by the arm-linux-gcc compiler. The static analyzer selects the execution time profiling points by the static timing analysis technique. It enables efficient profiling for the execution time by choosing only a small number of candidate VSEs from the target application. The static analyzer uses the assumption of one cycle for each instruction for the timing analysis. The profiler generates the annotated assembly code (mpeg4dec-profile.s) that has the execution time profiling code at the location specified by the static analyzer. The profile-enabled assembly code is compiled and executed at the Itsy platform. During the execution, the target application outputs the profile data that has the timing information of the candidate VSEs. Using these data, AVS analyzes the timing information of candidate VSEs, selects the VSEs, and generates the DVS-aware assembly code (mpeg4dec-dvs.s). At VSEs in the DVS-aware assembly code, the DVS function is called to adjust the clock speed and voltage. The digital multimeter measures the power consumption of the Itsy system while the DVS-aware code is executed at the Itsy platform.

The authors experimented with the same MPEG-4 programs used at the simulation experiments. It was assumed that each task processes ten frames (IPPPPIPPPP) before deadline.[8] Table III shows the comparison between DVS-aware programs and DVS-unaware programs. The energy reductions are 49% and 65% for the decoder and the encoder, respectively. The execution times of DVS-aware programs are shorter than WCETs because there are unselected VSEs and the Itsy platform provides discrete clock and voltage levels. Especially, when the adjusted clock speed reaches 54 MHz, which is the lowest clock speed, the speed adjustments at VSEs do not occur. For DVS-aware programs, Table III also shows the number of selected VSEs and the number of functions and loops where management codes (such as codes 1–6 in Fig. 6) are inserted. Since these numbers are small, it can be concluded that the additional overhead for IntraDVS is little. It can also be known that each VSE and management code requires very small instructions in the Itsy platform as shown in Table IV.[9] The AVS does not significantly increase the code size of target programs as shown in Table III.

Fig. 16(a) and (b) shows the graphs of power consumption measured during the executions of the DVS-aware MPEG-4 program and the DVS-unaware MPEG-4 program. The digital multimeter sampled the power consumption at the period of 20 ms. The DVS-aware programs set the clock and voltage to the maximum level at the start of the execution and reduce the clock and voltage at VSEs. The DVS-unaware programs execute at full speed (and the maximum power) until completion and have an idle interval. These experiments verify the effectiveness of the IntraDVS technique on a real variable-voltage system.

E. Comparisons of IntraDVS Algorithms

The energy efficiency of the reference path-based IntraDVS algorithm ($\texttt{IntraDVS}-\texttt{P}$) and the stochastic IntraDVS algorithm ($\texttt{IntraDVS}-\texttt{S}$) [20], [21] has been evaluated using an MPEG-4 video decoder and an MPEG-4 video encoder.

[8]Due to the low resolution of the multimeter, the authors increased the size of the task artificially for a better observation.

[9]Since the number of instructions for VSEs is different depending on the location, the average values are shown.

TABLE III
DVS EXPERIMENTS ON Itsy

Factors		MPEG-4 Decoder		MPEG-4 encoder	
		DVS-aware	DVS-unaware	DVS-aware	DVS-unaware
Energy (millijoule)		0.11	0.22	0.28	0.81
Normalized Energy		0.51	1	0.35	1
Execution Time (second)		1.18	0.46	5.34	1.54
WCET (second)		3.2		21.0	
Selected VSEs	B-VSE	2	-	1	-
	L-VSE	1	-	2	-
Management Code	Function	2	-	3	-
	Loop	5	-	8	-
Normalized Code Size		1.09	1	1.08	1

TABLE IV
MANAGEMENT CODE OVERHEAD

Management Code	Code Number in Fig. 6	Number of Assembly Instructions
Loop Enter	code 1 and 2	30
Loop Header	code 3	16
Loop Exit	code 4	16
Function Enter	code 5	14
Function Return	code 6	11
B-type VSE	code B	$\sim$ 200
L-type VSE	code L	$\sim$ 200

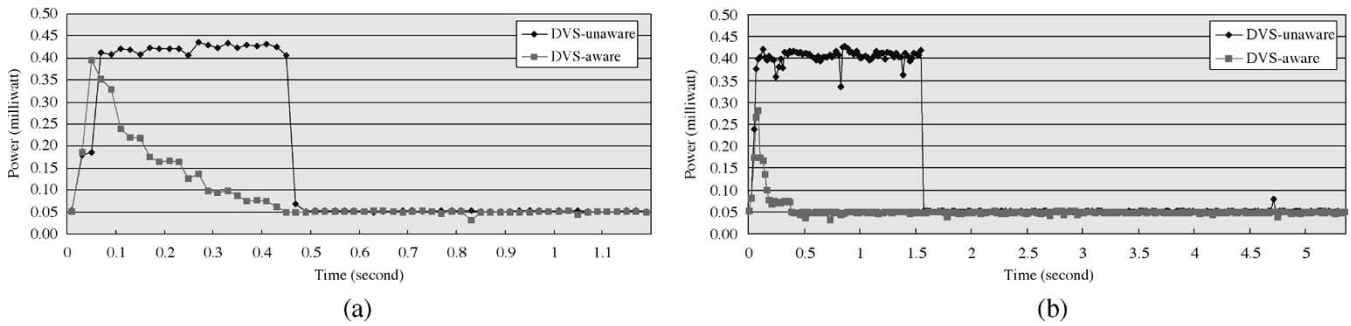

Fig. 16. Power estimation of MPEG-4 program: (a) MPEG-4 decoder and (b) MPEG-4 encoder.

The execution times of both the MPEG-4 decoder and encoder were assumed to follow a normal distribution $N_o = N[m_1, (m_2/6)^2]$, where $m_1 = (1/2) \times \text{WCET}$ and $m_2 = (9/10) \times \text{WCET}$.

Since the energy efficiency of $\texttt{IntraDVS} - \texttt{S}$ largely depends on the slack ratio[10] given in the on-line phase and the accuracy of the execution time distribution used in the off-line profiling, the authors performed experiments varying these two factors. Fig. 17 shows the relative energy consumption ratio of $\texttt{IntraDVS} - \texttt{S}$ over $\texttt{IntraDVS} - \texttt{P}$. If the ratio is larger (smaller) than 1, $\texttt{IntraDVS} - \texttt{S}$ performs better (worse) than $\texttt{IntraDVS} - \texttt{P}$. In Fig. 17, the N_o line represents the case when the actual execution times follow the assumed N_o distribution. The N_c line indicates the case where the actual execution times follow different normal distributions from the assumed N_o, where $N_c = N[1.5m_1, (m_2/7)^2]$.

When the slack ratio is less than 1.2, $\texttt{IntraDVS} - \texttt{P}$ outperforms $\texttt{IntraDVS} - \texttt{S}$ because $\texttt{IntraDVS} - \texttt{P}$ spends more time in the lower speed region than $\texttt{IntraDVS} - \texttt{S}$. When the slack ratio is increased, $\texttt{IntraDVS} - \texttt{S}$ spends more time in the lower speed region than $\texttt{IntraDVS} - \texttt{P}$. Fig. 17 also shows that $\texttt{IntraDVS} - \texttt{P}$ works better than $\texttt{IntraDVS} - \texttt{S}$ when the distribution of actual execution times is significantly different from the assumed distribution, as shown in the N_c line.

Another comparison between two IntraDVS algorithms has been presented in [31]. Since the energy performance of the proposed IntraDVS algorithm is dependent on the internal task structure, two IntraDVS algorithms were compared by varying the unawareness factor, which represents how early the authors

[10]The slack ratio is defined as the ratio of WCET to the assigned execution time.

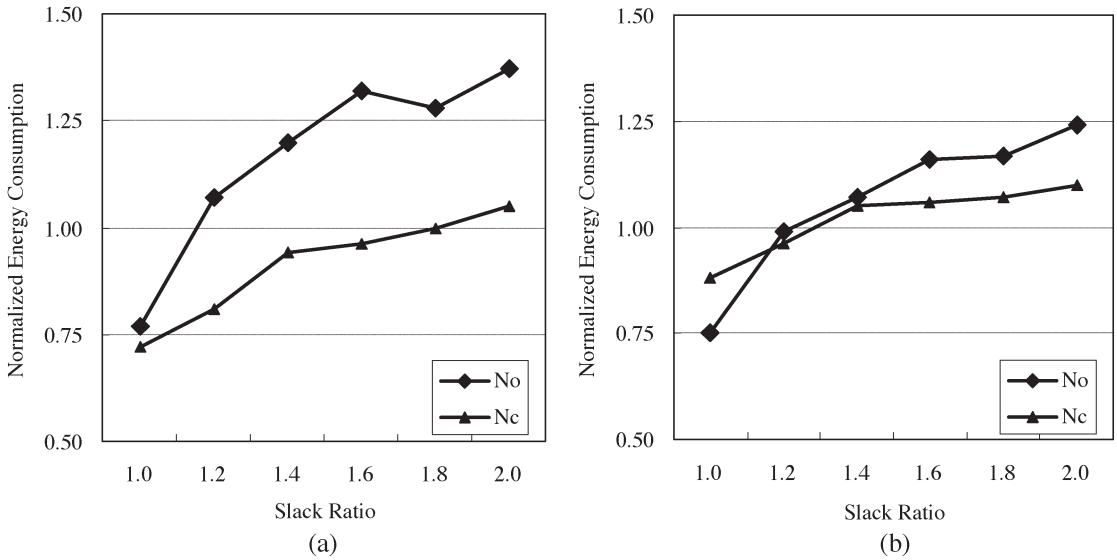

Fig. 17. Energy consumption ratio of IntraDVS − P and IntraDVS − S: (a) MPEG-4 decoder and (b) MPEG-4 encoder.

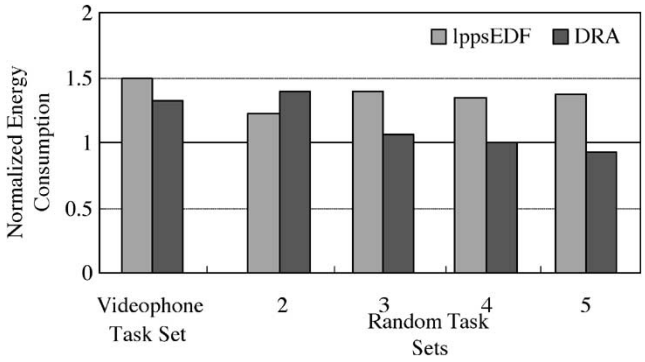

Fig. 18. Comparison of InterDVS and IntraDVS in multitask environments.

can know the exact value of the execution cycles of a task during the task execution. When the unawareness factor is small (large) (i.e., VSEs are located close to the entry (exit) block), the performance of the proposed IntraDVS is better (worse) than that of the stochastic IntraDVS.

F. Comparisons of InterDVS and IntraDVS

Although the IntraDVS algorithm is mainly proposed for single-task environments, it is also useful for multitask environments. To evaluate the feasibility of the IntraDVS algorithm in multitask environments, the authors compared the energy performance of the IntraDVS algorithm with several InterDVS algorithms. Fig. 18 shows the normalized energy consumptions of two typical InterDVS algorithms, i.e., lppsEDF [32] and DRA [10], over the RWEP-based IntraDVS algorithm. The videophone task set shown in Table I and the randomly generated task sets with 2, 3, 4, and 5 tasks were used in the experiment. Note that the randomly generated task sets do not represent normal program execution. Also note that they are not favorable to the IntraDVS because they do not have a dominant task as in the videophone task set.

As shown in Fig. 18, IntraDVS outperforms InterDVS in the videophone task set. In the randomly generated task sets, IntraDVS defeats lppsEDF, but it shows similar or worse performance with DRA when it has three or more tasks. This is because DRA uses a more sophisticated algorithm to utilize the slack time. Especially, when the number of task is five, the DRA algorithm shows a better performance than IntraDVS. From the above results, the authors can know that IntraDVS outperforms InterDVS even in multitask environments regardless of the existence of dominant tasks when the number of tasks is small. However, it could be better to use InterDVS when the real-time system has many tasks.

To observe the effect of the task set characteristics, the conventional InterDVS algorithms and the proposed IntraDVS algorithm were simulated with two different task sets. Two tasks sets, task sets A and B, have six tasks but the characteristics are quite different. Task set A is homogeneous (i.e., the tasks in A have similar periods and WCETs) while task set B is heterogeneous (i.e., the tasks in B have large variations in their periods and WCETs). Fig. 19 shows the normalized energy consumption of several InterDVS algorithms such as lppsEDF [32], ccEDF [11], laEDF [11], and DRA [10] over IntraDVS. Except for DRA, the IntraDVS algorithm outperforms most InterDVS algorithms tested.

It was also observed that InterDVS algorithms show poor energy performance when the worst-case processor utilization[11] (WCPU) is small. This is because the following tasks cannot fully exploit all the slack times generated by past and current tasks when the WCPU is too small. Some slack times disappeared before the following tasks are released. In this case, IntraDVS performs much better than InterDVS because it utilizes all the slack times.

The performance of the DRA algorithm is significantly different in the two task sets. As shown in Fig. 19(a), DRA

[11]WCPU means the total WCETs of all task instances divided by the hyper period.

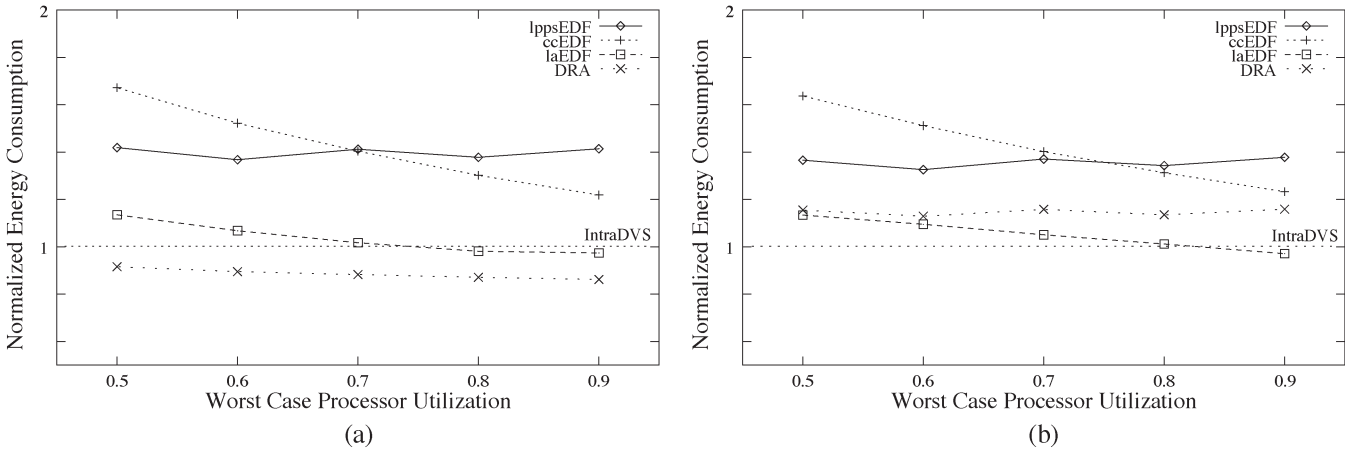

Fig. 19. Comparison of InterDVS and IntraDVS in different task sets: (a) task set A and (b) task set B.

outperforms both laEDF and IntraDVS when the task set A is used. However, when the task set B is used, DRA is inferior to IntraDVS, as shown in Fig. 19(b). This is because the slack estimation method in DRA does not work well with nonuniform task utilizations. From these results, it can be known that IntraDVS is better than InterDVS especially when the WCPU is small and the task set is heterogeneous.

VI. Conclusion

In this paper, the authors proposed an intra-task voltage scheduling framework for low-energy hard real-time applications. By statically analyzing the timing behavior of a dynamic voltage scheduling (DVS)-unaware real-time program, the proposed technique automates two time-consuming and complicated steps of applying intra-task voltage scheduling to DVS-unaware programs. First, the proposed technique automatically selects appropriate program locations where the supply voltage can be changed to minimize the energy consumption, satisfying the timing constraint. Second, the proposed technique inserts to the selected program locations a voltage scaling code in a completely transparent fashion to programmers. By automating these two steps, the proposed algorithm makes it possible for programmers without any knowledge on DVS to develop DVS-aware programs on a variable-voltage processor. The converted program by the proposed scheduling algorithm has a unique characteristic in that it always completes its execution near the deadline, thus resulting in no slack time. By lowering the execution speed and corresponding voltage to the maximum allowable extent, the proposed algorithm achieves a significant energy reduction ratio.

Two kinds of implementations of the intra-task DVS (IntraDVS) algorithm have been described, i.e., remaining worst-case execution path (RWEP)-based IntraDVS and remaining average-case execution path (RAEP)-based IntraDVS. While the worst-case timing information was used in the former, the average-case timing information was used for a better energy performance in the latter.

Based on the proposed intra-task voltage scaling framework, the authors have built an automatic voltage scaling tool, the automatic voltage scaler (AVS). AVS automatically transforms a DVS-unaware program to a DVS-aware low-energy program with the same functional behavior and timing requirements.

The experimental results using simulations with a Moving Pictures Expert Group (MPEG)-4 video encoder and decoder showed that AVS using RWEP-based IntraDVS improves the energy efficiency of the programs by a factor of 4–14 over the programs running on a non-DVS system with a power-down mode. The RAEP-based IntraDVS improved the energy efficiency by up to 34% over the RWEP-based IntraDVS. In the experiment using a real DVS-enabled system providing a finite number of clock/voltage levels, the low-energy version of an MPEG-4 encoder/decoder consumed only 35–51% of the energy consumption from the original program running on a fixed-voltage system with a power-down mode. The energy efficiencies of the IntraDVS algorithm and intertask DVS (InterDVS) algorithms were also compared. Although the IntraDVS algorithm generally outperformed the InterDVS algorithms, the relative energy efficiency was dependent on the task set characteristics.

Work in this paper can be extended in several directions. The execution time of an application depends on the run-time events (e.g., cache hit and miss) as well as the control flow. The authors plan to integrate such run-time information into the proposed framework so that the energy efficiency could be further improved. The energy efficiency of the InterDVS algorithms and the IntraDVS algorithm has been compared in this paper. Since none of them always outperforms the other, the energy efficiency of a DVS technique may be further improved by integrating the IntraDVS algorithm with the InterDVS algorithms.

Acknowledgment

The authors appreciate the anonymous reviewers whose helpful comments improved both the contents and the presentation of the paper.

References

[1] T. Burd, T. Pering, A. Stratakos, and R. Brodersen, "A dynamic voltage scaled microprocessor system," in *Proc. IEEE Int. Solid-State Circuits Conf.*, San Francisco, CA, 2000, pp. 294–295.

[2] *Transmeta Crusoe* [Online]. Available: http://www.transmeta.com/crusoe/longrun.html

[3] AMD, Inc., AMD PowerNow Technology, 2000.
[4] Intel, Inc., The Intel(R) XScale(TM) Microarchitecture Technical Summary, 2000.
[5] F. Yao, A. Demers, and S. Shenker, "A scheduling model for reduced CPU energy," in *Proc. 36th Annu. Symp. Foundations Computer Science*, Milwaukee, WI, 1995, pp. 374–382.
[6] 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. 19th IEEE Real-Time Systems Symp.*, Madrid, Spain, 1998, pp. 178–187.
[7] T. Okuma, T. Ishihara, and H. Yasuura, "Real-time task scheduling for a variable voltage processor," in *Proc. Int. Symp. System Synthesis*, Boca Raton, FL, 1999, pp. 24–29.
[8] Y. Shin and K. Choi, "Power conscious fixed priority scheduling for hard real-time systems," in *Proc. Design Automation Conf.*, New Orleans, LA, 1999, pp. 134–139.
[9] Y. Lee and C. M. Krishna, "Voltage-clock scaling for low energy consumption in real-time embedded systems," in *Proc. 6th Int. Conf. Real-Time Computing Systems and Applications*, Hong Kong, 1999, pp. 272–279.
[10] H. Aydin, R. Melhem, D. Mosse, and P. M. Alvarez, "Dynamic and aggressive scheduling techniques for power-aware real-time systems," in *Proc. IEEE Real-Time Systems Symp.*, London, U.K., 2001, pp. 95–105.
[11] P. Pillai and K. G. Shin, "Real-time dynamic voltage scaling for low-power embedded operating systems," in *Proc. 18th ACM Symp. Operating Systems Principles (SOSP)*, Banff, Canada, 2001, pp. 89–102.
[12] W. Kim, J. Kim, and S. L. Min, "A dynamic voltage scaling algorithm for dynamic-priority hard real-time systems using slack time analysis," in *Proc. Design Automation and Test Europe*, Paris, France, 2002, pp. 788–794.
[13] S. Lee and T. Sakurai, "Run-time voltage hopping for low-power real-time systems," in *Proc. Design Automation Conf.*, Los Angeles, CA, 2000, pp. 806–809.
[14] C. Im, H. Kim, and S. Ha, "Dynamic voltage scheduling technique for low-power multimedia applications using buffers," in *Proc. Int. Symp. Low Power Electronics and Design*, Huntington Beach, CA, 2001, pp. 34–39.
[15] F. Gruian and K. Kuchcinski, "LEneS: Task-scheduling for low-energy systems using variable voltage processors," in *Proc. Asia South Pacific Design Automation Conf.*, Yokohama, Japan, 2001, pp. 449–455.
[16] G. Quan and X. S. Hu, "Energy efficient fixed-priority scheduling for real-time systems on variable voltage processors," in *Proc. Design Automation Conf.*, Las Vegas, NV, 2001, pp. 828–833.
[17] W. Kim, D. Shin, H.-S. Yun, J. Kim, and S. L. Min, "Performance comparison of dynamic voltage scaling algorithms for hard real-time systems," in *Proc. IEEE Real-Time and Embedded Technology and Applications Symp. (RTAS)*, San Jose, CA, 2002, pp. 219–228.
[18] D. Shin, J. Kim, and S. Lee, "Intra-task voltage scheduling for low-energy hard real-time applications," *IEEE Des. Test. Comput.*, vol. 18, no. 2, pp. 20–30, 2001.
[19] D. Shin and J. Kim, "A profile-based energy-efficient intra-task voltage scheduling algorithm for hard real-time applications," in *Proc. Int. Symp. Low Power Electronics and Design*, Huntington Beach, CA, 2001, pp. 271–274.
[20] F. Gruian, "Hard real-time scheduling using stochastic data and DVS processors," in *Proc. Int. Symp. Low Power Electronics and Design*, Huntington Beach, CA, 2001, pp. 46–51.
[21] J. R. Lorch and A. J. Smith, "Improving dynamic voltage scaling algorithms with PACE," in *Proc. Association for Computing Machinery (ACM) SIGMETRICS Conf.*, Cambridge, MA, 2001, pp. 50–61.
[22] C. H. Hsu and U. Kremer, "The design, implementation, and evaluation of a compiler algorithm for CPU energy reduction," in *Proc. ACM SIGPLAN Conf. Programming Languages, Design, and Implementation*, San Diego, CA, 2003, pp. 38–48.
[23] Texas Instruments, Inc., Using the Power Scaling Library on the TMS320C5510, 2002.
[24] R. Hamburgen, D. Wallach, M. Viredaz, L. Brakmo, C. Waldspurger, J. Bartlett, T. Mann, and K. Farkas, "Itsy: Stretching the bounds of mobile computing," *IEEE Comput.*, vol. 34, no. 4, pp. 28–36, 2001.
[25] J. Pouwelse, K. Langendoen, and H. Sips, "Voltage scaling on a low-power microprocessor," in *Proc. Mobile Computing Conf. (MOBICOM)*, Rome, Italy, 2001, pp. 251–259.
[26] C. A. Healy, D. B. Whalley, and M. G. Harmon, "Integrating the timing analysis of pipelining and instruction caching," in *Proc. 16th IEEE Real-Time Systems Symp.*, Pisa, Italy, 1995, pp. 288–297.
[27] S.-S. Lim, Y. H. Bae, G. T. Jang, B.-D. Rhee, S. L. Min, C. Y. Park, H. Shin, K. Park, and C. S. Kim, "An accurate worst case timing analysis for RISC processors," *IEEE Trans. Softw. Eng.*, vol. 21, pp. 593–604, Jul. 1995.
[28] T. Ball and J. R. Larus, "Using paths to measure, explain, and enhance program behavior," *IEEE Comput.*, vol. 33, no. 7, pp. 57–65, 2000.
[29] T. Ishihara and H. Yasuura, "Voltage scheduling problem for dynamically variable voltage processors," in *Proc. Int. Symp. Low Power Electronics and Design*, Monterey, CA, 1998, pp. 197–202.
[30] T. Sakurai and A. Newton, "Alpha-power law MOSFET model and its application to CMOS inverter delay and other formulas," *IEEE J. Solid-State Circuits*, vol. 25, no. 2, pp. 584–594, 1990.
[31] F. Gruian, "On energy reduction in hard real-time systems containing tasks with stochastic execution times," in *Proc. IEEE Workshop Power Management Real-Time and Embedded Systems*, Taipei, Taiwan R.O.C., 2001, pp. 11–16.
[32] Y. Shin, K. Choi, and T. Sakurai, "Power optimization of real-time embedded systems on variable speed processors," in *Proc. Int. Conf. Computer-Aided Design*, San Jose, CA, 2000, pp. 365–368.

Dongkun Shin (S'01) received the B.S. degree in computer science and statistics, the M.S. degree in computer science, and the Ph.D. degree in computer science and engineering from the Seoul National University, Seoul, Korea.

He is a Senior Engineer at Samsung Electronics Company, Seoul. His research interests include low-power systems, computer architecture, and embedded and real-time systems.

Dr. Shin is a Member of ACM.

Jihong Kim (M'00) received the B.S. degree in computer science and statistics from the Seoul National University, Gwanak-gu, Seoul, Korea, and the M.S. and Ph.D. degrees in computer science and engineering from the University of Washington, Seattle.

He is an Associate Professor in the School of Computer Science and Engineering, Seoul National University. His research interests include embedded systems, computer architecture, Java computing, and multimedia and real-time systems.

Dr. Kim is a Member of ACM.