Task Partitioning Algorithm for Intra-Task Dynamic Voltage Scaling

Seungyong Oh, Jungsoo Kim, Seonpil Kim and Chong-Min Kyung
Dept. of Electrical Engineering & Computer Science
KAIST (Korea Advanced Institute of Science and Technology)
Email: {syoh, jskim, spkim} @vslab.kaist.ac.kr, kyung@ee.kaist.ac.kr

Abstract—Dynamic voltage scaling (DVS) is a very powerful technique in reducing dynamic power consumption of CMOS circuits. Recent studies showed that intra-task DVS method which adjusts the voltage level during program execution can achieve significant energy reduction. However, the overhead of large number of voltage switching becomes a limitation for its practical implementation. In this paper, we propose a novel task partitioning algorithm for intra-task DVS which partitions a given task so that the DVS can be applied more effectively with the minimum number of voltage switching. Experimental result using H.264 decoder software shows that the proposed algorithm reduces the energy consumption by up to 25% over the conventional method.

I. INTRODUCTION

Power-efficiency is one of the biggest concerns in modern embedded system design. Many techniques have been developed to reduce the energy consumption of the microprocessor which is one of the major contributors to energy consumption of embedded system. Dynamic voltage scaling (DVS) reduces the power consumption of a processor using a quadratic dependence of active power consumption on supply voltage.

DVS can be classified into two different methods according to its scaling granularity. The first one is inter-task DVS which adjusts the voltage/frequency level between tasks. The task in this context means the independent software which can be scheduled by an operating system. The second one is intra-task DVS which adjusts the voltage/frequency level a number of times in a single task boundary. This method is based on the fact that the execution cycle of the software is not deterministic but has a profile of large variation. Hence, it can utilize the time slack caused by run-time variation of execution cycle efficiently.

The effectiveness of intra-task DVS method has been studied through many researches. In most cases, intra-task DVS method outperforms the inter-task DVS. However, it also has a limitation. The main drawback comes from the fact that a large number of voltage switching should be executed to achieve a large energy reduction. Because the intra-task DVS has to adjust its supply voltage level dynamically during the task’s execution, the overhead which comes from voltage switching is unavoidable. In fact, it has been shown that intra-task DVS becomes more powerful when the number of voltage switching increases [3] [7].

This voltage switching overhead becomes a main bottleneck in its practical implementation. Even a current state-of-the-art DC-DC converter takes tens of μsec in voltage switching [1]. As the number of voltage switching increases, the time consumed for voltage switching also increases, thus the time for execution of real application decreases. Hence, the voltage switching has to be performed very efficiently to minimize this switching overhead in intra-task DVS.

The goal of our research is to maximize the effectiveness of intra-task DVS by minimizing the voltage switching overhead for hard real-time application. More specifically, we propose an algorithm which partitions the program into number of code sections in a way that voltage switching is only performed when it is necessary. In our experiment, it is shown that the number of voltage-switching can be reduced drastically while keeping the same amount of energy reduction. Furthermore, additional energy reduction is also achievable by proposed method.

The rest of paper is organized as follows. In section II, we present related works on intra-task DVS. Section III describes the details of proposed method called task partitioning algorithm. Next, Section IV shows the experimental result of proposed method. Finally, we draw our conclusions in section V.

II. RELATED WORK

Lee and Sakurai [3] proposed an idea of the intra-task DVS method. A. Azevedo et al. [4] used the program check point to apply intra-task DVS. Their works utilized the time slack by analyzing the worst-case execution cycle of remaining task. However, the variation of remaining workload and execution path was not considered. Seo et al. [5] proposed the virtual execution path. In this work, they tried to find the optimal execution path among many paths caused by branches. As in [4], they also took the worst-case execution cycle as a remaining workload of each path. Hong et al. [7] proposed profile-based remaining workload predicting method. In this work, statistically-optimal remaining workload was determined using each performance regions’ profile information.

All these previous works have been focused on which level the voltage has to be adjusted to. However, no work has been done on at which points of the program’s execution the DVS should be applied considering voltage switching overhead.
III. PROPOSED METHOD

A. Preliminary

![Intra-task DVS](chart.png)

Fig. 1. Intra-task DVS

1) Intra-task Dynamic Voltage Scaling: In intra-task Dynamic Voltage Scaling, the task is divided into a number of code sections as shown in Fig.1. This code section becomes the basic unit of voltage scaling. In the same code section, the voltage is maintained at the same level and the voltage can be switched to a different level between code sections. If we can estimate the remaining workload (how many cycles are left) exactly, it would be possible to lower the frequency as much as possible for each section by using this simple equation.

\[ f_i = \frac{w_{Ri}}{t_i} \]  

where \( w_{Ri} \) is the remaining workload of the i-th code section, \( t_i \) is a run-time parameter which denotes the remaining time from the start time of i-th code section to task’s deadline. Many methods to find the value of \( w_{Ri} \) have been proposed in the researches introduced in the previous section.

2) Remaining workload prediction: The prediction method for remaining workload of each code section \( (w_{Ri}) \) makes a significant effect on the performance of intra-task DVS. In this paper, we used the profile-based remaining workload prediction method in [7]. In [7], the value of \( w_{Ri} \) which minimizes the average energy consumption is determined as an analytical solution of the energy equation. The \( w_{Ri} \) for each code section is stored in the table, and is referenced to calculate the proper frequency level using (1) during task’s execution.

3) Energy model: We assume that the amount of energy spent for executing a code section is expressed as

\[ E \propto f^2 \times n_{total} \]  

where \( f \) is the performance level (i.e. frequency) and \( n_{total} \) is the total execution cycle of the code section. The validity of this equation is shown in [6] [7].

4) Hard real-time application: The real-time application can be classified into two types. The first one is soft real-time application which has to keep a certain level of quality of service (QoS) while its execution does not necessarily meet the deadline. To these applications, occasional deadline misses happen so the quality degradation is accompanied. The other is hard real-time application which has to meet the deadline in any circumstances. This type of application guarantees the quality of the output and is the target of this work.

B. Motivation

Our proposed method is based on the two basic observations. The following two subsections explain those examples, and then details of proposed algorithm will be presented.

1) Profile variation: The first observation is that DVS should be applied more frequently to the code section which has a large variation in its execution cycle.

![Example 1](chart2.png)

Fig. 2. Example 1: Difference of energy consumption when variance of execution cycle is considered

In Fig.2, one node represents a code section and its average execution cycle is shown below each node’s name. In case A, the task is partitioned so that each section has similar execution cycles. The large variation in execution cycle of \( n_1 \) is shown as a profile. In case B, however, \( n_1 \) is divided into two nodes, and the \( n_2 \) and \( n_3 \) are merged into one node so that the variance of each node becomes smaller. The predicted remaining workload of each node is assumed to be the sum of average cycle of the node and its subsequent nodes for simplicity. For example, remaining workload prediction of \( n_1 \) in case A is 1k cycles.

When the same execution cycles with the predicted value are actually taken, there is no difference between two cases in their energy consumption. However, when the execution cycle of the \( n_1 \) in case A become 1/3 and 2 times of its average value, and the same happens to the \( n_1 \) and \( n_2 \) in case B, the energy consumption of case B becomes smaller than that of case A by 0.8% and 12% respectively.

This result shows that the code section with a large variance is better to be divided into smaller code sections so that the frequency can be adapted to its variation quickly. It also indicates that the node with a small variance can be merged into one node to reduce the voltage transition overhead.

2) Remaining workload: When the actual execution cycle of the code section differs from its predicted value by [7]’s method, additional energy called misprediction penalty is consumed. This misprediction penalty for the same variation grows as remaining workload decreases.
In Fig.3, total execution cycles for both cases are same. In case A, however, remaining workload after \( n_0 \) is 200 cycles larger than that of case B. When 100 cycles of misprediction occurred in both cases, the energy consumption in case B is much larger than that of case A as shown in Fig.3.

This example shows that for the same amount of misprediction, the node with the smaller remaining workload has a larger energy penalty. In other words, as remaining workload decreases (program proceeds), misprediction penalty for the same variation grows. This is because when the remaining workload is large, the penalty for misprediction can be amortized over a large number of remaining cycles.

3) Energy penalty for misprediction: Now, we present derivation of the energy penalty equation.

\[
\begin{align*}
E_{\text{opt}} &= \left( \frac{x_0 + x_1}{T} \right)^2 x_0 + \left[ \frac{x_1}{T - \left( \frac{x_0 + x_1}{T} \right)} \right]^2 x_1 \\
E_{\text{miss}} &= \left( \frac{x_0 + x_1}{T} \right)^2 (x_0 + \Delta x) + \left[ \frac{x_1}{T - \left( \frac{x_0 + \Delta x}{T} \right)} \right]^2 x_1
\end{align*}
\]

where \( E_{\text{opt}} \) and \( E_{\text{miss}} \) are the energy consumption of the node when the remaining workload is correctly predicted and mispredicted with amount of \( \Delta x \) of the energy. The energy misprediction penalty is expressed as a difference between these two values.

\[
E_{\text{pnt}} = E_{\text{miss}} - E_{\text{opt}} = \left( \frac{x_0 + x_1}{T} \right)^2 \Delta x + \left[ \frac{x_1}{x_1 - \Delta x} \right]^2 - 1 \] (3)

This penalty increases as execution cycle variation grows and the remaining workload decreases as we expected.
denotes the worst-case execution cycle of the node. When this condition is not satisfied, the frequency of the node has to be increased until it meets the condition.

IV. EXPERIMENTAL RESULT

We performed our experiment on ARM-based embedded system platform. Our platform consists of ARM946 processor with cache enabled, AHB bus and memory controller with SDRAM.

The maximum and the minimum voltage level for the processor was set to 1.2V and 0.6V with 12 discrete voltage/frequency levels. The frequency can vary from 120MHz to 480MHz with its corresponding voltage level. At run-time, the frequency level is determined as the smallest value among available frequency levels larger than \( f_{\text{opt}} \)\(^{[7]}\).

We also took into account the delay overhead accompanied with DVS. The time spent on executing DVS function call was assumed to be 1k cycles as a constant, and the maximum 22 \( \mu \text{sec} \) of voltage transition time was also assumed without the loss of generality\(^{[1]}\).

Industrial H.264 decoder was used as a hard real-time application. This target application decodes QCIF(176x144) images with 30fps.

In our experiments, the target application was divided into 300 basic code sections after all the big loops in the original source code were unrolled to prevent re-execution of the same section. The task is assumed to be run under non-preemptive scheduling. We used the SoC Designer \(^{[8]}\) cycle-accurate model to profile the execution cycle of each code section.

We applied proposed method with various energy threshold levels. As energy threshold level increases, more nodes have to be merged, thus the total number of code sections is decreased. We also performed the same experiment with the method in \(^{[7]}\) for comparison. In \(^{[7]}\), code sections were merged with their adjacent code sections. The energy consumption measured in each case is shown in Fig. 7

The energy consumed in each case is shown in Table I. The minimum energy consumption (with asterisks) is 50.31mJ with 210 code sections when proposed method is applied while it is 67.58mJ with 180 code sections when conventional method is applied. This result shows that almost 25% of additional energy reduction can be achievable when proposed algorithm is used. The result of task partitioning using both methods is shown in Fig. 8 when the total number of code section is 30.

![Fig. 7. Energy consumption : H.264 software](image)

The number in parenthesis denotes the number of nodes merged in. We can see that in proposed method, a large number of nodes in the beginning are merged together because of its small misprediction penalty.

V. CONCLUSION

We presented a novel task partitioning algorithm for intra-task dynamic voltage scaling. Proposed method partitions the task into code sections by considering misprediction penalty so that intra-task DVS can be applied more effectively. The experimental results show that the proposed algorithm can achieve substantial power reduction for industrial multimedia application.

REFERENCES

1. O. Trescoes and W. Ng. "Variable Output, Soft-Switching DC/DC Converter for VLSI Dynamic Voltage Scaling Power Supply Applications", PESC, 2004