SimDVS: An Integrated Simulation Environment for Performance Evaluation of Dynamic Voltage Scaling Algorithms*

Dongkun Shin, Woonseok Kim, Jaekwon Jeon, Jihong Kim, and Sang Lyul Min

School of Computer Science and Engineering
Seoul National University, Seoul, Korea

Abstract. We describe SimDVS, a unified simulation environment for evaluating dynamic voltage scaling (DVS) algorithms, and present the evaluation results for three case studies using SimDVS. In recent years, DVS has received a lot of attention as an effective low-power design technique, and many research groups have proposed various DVS algorithms. However, these algorithms have not been quantitatively evaluated, making it difficult to understand the performance of a new DVS algorithm objectively relative to the existing DVS algorithms. The SimDVS environment provides a framework for objective performance evaluations of various DVS algorithms. Using SimDVS, we compare the energy efficiency of the intra-task DVS algorithm and inter-task DVS algorithms, and evaluate various heuristics for a hybrid DVS approach. We also show that more efficient DVS algorithms may incur higher system overheads, degrading the overall energy efficiency of the DVS algorithms.

1 Introduction

For battery-operated mobile embedded devices such as personal digital assistants (PDAs) and cellular phones, power consumption is an important design constraint. As an effective low-power design technique, dynamic voltage scaling (DVS) recently has received a lot of attention. For example, several commercial variable-voltage microprocessors [1, 2, 3] were introduced last 2 years, and many DVS algorithms applicable to these microprocessors [4, 5, 6, 7, 8, 9, 10, 11, 12] have been proposed, especially targeting for hard real-time systems.

Although proposed DVS algorithms are shown to be effective in reducing the energy consumption of a target system under their own experimental scenarios, these algorithms have not been quantitatively evaluated each other under a unified evaluation framework. The lack of comprehensive evaluation studies makes it difficult to understand the energy efficiency of a new DVS algorithm relative to that of the existing DVS algorithms. In this paper, we describe SimDVS, an

* This work was supported by grant No. R01-2001-00360 from the Korea Science & Engineering Foundation. Woonseok Kim and Sang Lyul Min were supported in part by the Ministry of Science and Technology under the National Research Laboratory program.

B. Falsafi and T.N. Vijaykumar (Eds.): PACS 2002, LNCS 2325, pp. 141–156, 2003.

integrated simulation environment for DVS algorithms, which can be used in comparing the energy efficiency of various DVS algorithms.

1.1 DVS Algorithms for Hard Real-Time Systems

For hard real-time systems, DVS algorithms can be categorized into two classes, inter-task DVS (InterDVS) and intra-task DVS (IntraDVS). InterDVS algorithms determine the supply voltage on task-by-task basis while IntraDVS algorithms adjust the supply voltage within an individual task boundary. InterDVS algorithms are further divided depending on the scheduling policy employed, say, the earliest-deadline-first (EDF) or rate-monotonic (RM) scheduling policies. Table 1 summarizes the recent DVS algorithms proposed for hard real-time systems, six EDF InterDVS algorithms and two RM InterDVS algorithms. InterDVS algorithms under the same scheduling policy are different mainly in the way how slack times are estimated. For example, `lppsEDF` conservatively estimates available slack times at scheduling points. On the other hand, `lpSHE` employs an aggressive technique in estimating the slack times available.

IntraDVS algorithms can be divided into two subcategories, path-based IntraDVS algorithms [11] and stochastic IntraDVS algorithms [12], depending on how to estimate slack times and how to adjust speeds. In the path-based IntraDVS algorithms, the voltage and clock speed are statically determined by computing the differences between the execution cycles of the predicted execution path (e.g., the worst case execution path (WCEP)) and the execution cycles of the execution path actually taken. When the actual execution deviates from the predicted execution path (say, by a branch instruction), the change in workload is detected and the clock speed is adjusted.

The stochastic IntraDVS algorithms adjust the execution speed within a task boundary based on the probability density function of execution times of a task. This method is based on an observation that, from the energy consumption point of view, it is better to execute with a lower speed in the beginning and increase the execution speed later when necessary. Under the stochastic method, the clock

Table 1. Recent DVS algorithms proposed for hard real-time systems

Category	Scheduling Policy	DVS Algorithms
InterDVS	EDF	`lppsEDF` [7]
		`ccEDF` [9]
		`laEDF` [9]
		`DRA` [8]
		`AGR` [8]
		`lpSHE` [13]
	RM	`lppsRM` [7]
		`ccRM` [9]
IntraDVS	Path-based Method	`intraShin` [11]
	Stochastic Method	`intraGruian` [12]

speed is *raised* at the specific times, regardless of the execution paths taken. Unlike the path-based IntraDVS algorithms that can utilize all the slack times from the task execution in scaling the execution speed, the stochastic IntraDVS algorithms may leave some slack times unused when the actual execution takes a short execution path (other than WCEP).

1.2 Our Contribution

The SimDVS simulation environment was developed to help quantitative performance analysis and evaluation of DVS algorithms by providing a unified evaluation framework. The current version of SimDVS supports all the DVS algorithms listed in Table 1. In addition to the DVS algorithms, SimDVS includes utility programs that are useful for DVS comparative studies. For example, SimDVS provides a tool that automatically generates a task set with specific task characteristics.

In order to demonstrate the effectiveness of SimDVS, we present three case studies in this paper. First, we compare the energy efficiency of an IntraDVS algorithm and InterDVS algorithms. Second, we evaluate if hybrid DVS algorithms (that adopt both the IntraDVS approach and the InterDVS approach) can perform better than pure IntraDVS algorithms or pure InterDVS algorithms. Third, we show that more efficient DVS algorithms may experience higher system overheads (e.g., more context switches), possibly degrading the overall energy efficiency of the DVS algorithms.

The rest of the paper is organized as follows. In Section 2, we present the overview of SimDVS. The detailed description of main modules of SimDVS is given in Section 3. We present three case studies in Section 4 and conclude with a summary in Section 5.

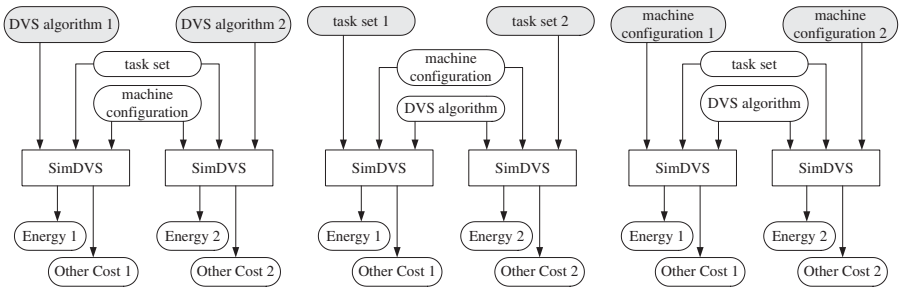
2 Overview of SimDVS

2.1 Design Goals

In order to effectively evaluate various DVS algorithms under a variety of simulation scenarios, SimDVS was architected to meet the following design goals:

1. New DVS algorithms based on different DVS approaches should be easily integrated into SimDVS.
2. Variations in simulation scenarios should be easily supported. Simulation scenarios can differ, for example, in task workloads, variations in executed paths, and task set specifications.
3. Different type of variable-voltage processors should be easily supported.

Fig. 1 shows three examples of using SimDVS for evaluating DVS algorithms. As shown in Fig. 1(a), SimDVS can be used to compare the energy efficiency of different DVS algorithms using the same task set specification under the same machine configuration. Using this evaluation, one can decide the best DVS algorithm for the given application on the given hardware platform.



(a) DVS algorithm comparison

(b) Performance variations with different task sets

(c) Performance variations with different machine configurations

Fig. 1. Three SimDVS usage examples

SimDVS can be used as well when evaluating a given DVS algorithm under various evaluation conditions. For example, one can evaluate how the energy efficiency of the DVS algorithm changes with different task sets (as shown in Fig. 1(b)). A similar evaluation can be performed with different machine configurations (as shown in Fig. 1(c)).

If properly instrumented, SimDVS can collect information on various system events or performance parameters other than energy consumption. These extra profiling data are useful when understanding how DVS algorithms affect the general behavior of a target system. The current version of SimDVS can collect the frequency of speed changes and the number of context switches.

2.2 Architectural Organization

Fig. 2 shows an architectural overview of SimDVS. SimDVS consists of three main modules: 1) the InterDVS Module, 2) the IntraDVS Module and 3) the IntraDVS Preprocessing Module. SimDVS takes two inputs: 1) a task set specification or a DVS-aware control flow graph (CFG) of an input binary program, respectively, for an InterDVS algorithm or an IntraDVS algorithm, and 2) a target machine specification. As outputs, the energy consumption of the input task(s) is estimated. If required, other profiling data are also collected.

The InterDVS Module is responsible for the whole operation of SimDVS. It simulates a given task set under a selected scheduling policy using a slack estimation and distribution heuristic. The IntraDVS Module simulates IntraDVS algorithms using the Intra-Task Simulator. The input to the IntraDVS Module is pre-processed by the tools available in the IntraDVS Preprocessing Module. For faster simulations of path-based IntraDVS algorithms, we simulate the CFG of the input program instead of the input program itself. For a comparison study, the current version of SimDVS supports ten DVS algorithms listed in Table 1.

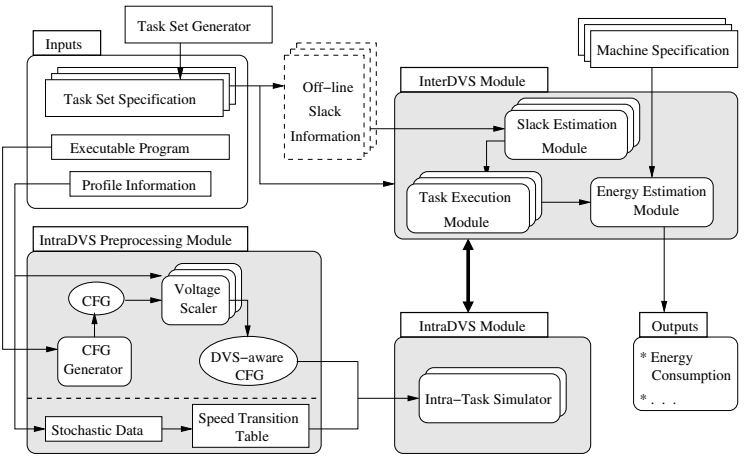


Fig. 2. An overview of the SimDVS simulation environment

3 Main Modules of SimDVS

In this section, we describe main modules of SimDVS (shown in Fig. 2) in detail.

3.1 SimDVS Inputs

The task set specification describes various task set characteristics that affect the energy efficiency of a DVS algorithm while the machine specification describes the machine characteristics that affect the energy efficiency of a DVS algorithm.

Task Set Specification The energy-efficiency of DVS algorithms can be affected by the characteristics of a given task set such as the number of tasks, the task execution time distributions, and the worst case processor utilization (WCPU). Therefore, when evaluating DVS algorithms, it is necessary to understand how the performance of the DVS algorithms varies depending on task sets with different characteristics.

In SimDVS, the characteristics of a task set T=(τ_1, τ_2, $\cdots$, τ_n) is specified in a script file that contains the following information on each task τ_i of the task set T:

- ID_i: the identifier of τ_i.
- P_i: the period of τ_i.
- D_i: the deadline of τ_i.
- WCET_i: the worst case execution time (WCET) of τ_i.
- BCET_i: the best case execution time (BCET) of τ_i.
- $\mathrm{Distribution}_i$: the execution time distribution of τ_i.

In order to automatically generate a task set with specific characteristics, the Task Set Generator is used. The Task Set Generator takes the following information as inputs and generates as an output the corresponding script file for a task set satisfying the requirements:

- The number of tasks.
- The range and variation of periods.
- The ratio between BCET_i and WCET_i.
- The worst case processor utilization of a task set.
- The scheduling policy (e.g., EDF or RM).

The Task Set Generator creates schedulable task sets only, under the scheduling policy specified. For EDF scheduling, if the worst case processor utilization of the task set is lower than or equal to 1.0, the task set is schedulable. For RM, the schedulability is verified using the *exact schedulability condition* described in [14].

Machine Specification A machine specification includes the available voltage and clock levels of a target variable-voltage processor. The machine specification reflects the characteristics of the target variable-voltage processor. Using SimDVS, with the target task sets and DVS algorithms fixed, the DVS-related architectural exploration is possible in designing variable-voltage processors. By a default, SimDVS uses the machine specification described in [15]. If necessary, other machine specifications are easily supported. The current version of SimDVS includes the specifications of Intel's *XScale* [3], AMD's *K6-2+* [2], and Transmeta's *Crusoe* [1] processors.

3.2 InterDVS Module

The InterDVS Module, responsible for scheduling tasks, plays a role of a real-time scheduler in a hard real-time system. It takes as an input a task specification for periodic tasks and simulates each task based on the specified scheduling policy (e.g., RM or EDF). To simulate an InterDVS scheduling algorithm, the InterDVS Module consists of two submodules, one for estimating available slack times and the other for distributing the slack times to each task. The slack estimation is done by the Slack Estimation Module which computes the total available time for the scheduled task while the slack distribution is done by the Task Execution Module which determines the operating speed for the scheduled task and simulates the execution of the task. For a new InterDVS algorithm, these two submodules should be re-defined.

Slack Estimation Module The implementation of this module is different depending on how the target InterDVS algorithm estimates the available slack times. This module is integrated with the InterDVS Module using the getAvailableTime function. This function receives the task identifier and the start time of the task as inputs, and returns the total available time for the task. Some

DVS algorithms (e.g., [12]) may need off-line pre-processing steps for computing total available times during run time. For these algorithms, the Slack Estimation Module can take off-line slack analysis results as an additional input.

Task Execution Module This module has two roles. First, it determines the voltage level and clock speed based on the available time for the current task and the WCET of the task. Although most existing DVS algorithms employ a greedy approach in distributing the available slack times, if a DVS algorithm adapts different slack distribution methods, they can be supported in this module. Using the available voltage levels specified in the machine specification input, this module sets the voltage level and clock speed. With the assigned clock speed, the activated task instance consumes all the assigned time interval if its execution takes the WCEP.

Second, this module simulates the task execution itself. In this module, a real workload for each task is generated based on the input workload variation factors (i.e., Distribution_i), and the unused time as well as the elapsed time is calculated out of the available time interval. This module also sends the execution time information and speed information to the Energy Estimation Module. When the IntraDVS algorithm is used, this module calls the Intra-Task Simulator of the IntraDVS Module to simulate IntraDVS.

Energy Estimation Module This module takes the timing and speed information from the Task Execution Module, and computes the energy consumption of the current task execution using the current machine specification. Energy consumption is calculated using the equation $E \propto N_{cycle} \cdot {V_{dd}}^2$, where N_{cycle} and V_{dd} denote the number of execution cycles and the supply voltage, respectively.

3.3 IntraDVS and Its Preprocessing Module

In order to support the simulation of the IntraDVS algorithms, voltage scaling points within a task boundary should be determined during the off-line phase. The submodules in the IntraDVS Preprocessing Module are responsible for making intra-task voltage scaling decisions, which are passed to the IntraDVS Module using a DVS-aware CFG or a Speed Transition Table. To reflect the execution behavior of real applications, the CFG Generator produces a CFG from SimpleScalar 2.0 [16] binary programs. Each node of the CFG has several node attributes (e.g., the number of instructions in a basic block) that are necessary for simulation.

Voltage Scaler This module is used for the path-based IntraDVS algorithms. It takes the CFG of a target application, and extracts the timing information from the CFG. By analyzing the CFG, this module computes the remaining predicted execution times (RPETs) for each basic block. Based on the RPETs computed, the voltage scaling edges in the CFG are selected using the algorithm described in [11]. As an output, this module generates the DVS-aware CFG which includes the voltage scaling information.

Speed Transition Table To simulate the stochastic IntraDVS algorithm, the stochastic data, i.e., the cumulative distribution function for task execution times, is either provided by user or collected using some profiling runs. Based on the stochastic data, the Speed Transition Table, which describes when the execution speed is changed to which speed, is constructed.

Intra-Task Simulator This module simulates the task execution using a given DVS-aware CFG and a Speed Transition Table for the path-based IntraDVS algorithms and the stochastic IntraDVS algorithm, respectively. During simulation, it adjusts the voltage and clock speed based on the voltage scaling information specified as a part of the input CFG or the Speed Transition Table.

To simulate the path-based IntraDVS algorithms, the Intra-Task Simulator requires the information on the execution path actually taken. The Intra-Task Simulator generates an execution path trace from the input CFG by randomly selecting one of two branching edges and setting the number of loop iterations by a random number between 0 and N_{max_loop} where N_{max_loop} is the maximum number of loop iterations. To change the execution times of the selected execution paths, SimDVS controls two parameters, α and β, which can be specified by user. α represents the probability of selecting the branching edge with a longer remaining time while β indicates the ratio of the average number of loop iterations to the maximum number of loop iterations for each loop. For example, if β is set to 0.5, the average number of loop iterations becomes close to the half of the maximum number of loop iterations. When both α and β are set to 1.0, the execution time of the selected path is close to WCET. Fig. 3 illustrates how the execution cycles of a task change when α and β vary. As α and β increase, the execution time tends to increase.

Once the execution time t for the simulated task is determined by the Task Execution Module (in the InterDVS Module), the Intra-Task Simulator generates the execution path trace whose execution time is close to the task execution time t. To help the path generation step, the IntraDVS Preprocessing Module maintains the database of execution path traces with their (α, β) values and corresponding execution times.

4 Case Studies

In this section, we present three case studies that demonstrate how SimDVS can be used in evaluating various DVS algorithms.

4.1 Performance Evaluation of InterDVS and IntraDVS

First, we compared the energy efficiency of InterDVS algorithms and an IntraDVS algorithm. For the evaluation study, we compared the energy efficiency of four EDF InterDVS algorithms, the `lppsEDF`, `ccEDF`, `laEDF` and `DRA` algorithms, with that of the `intraShin`[1] algorithm. (These algorithms are listed in

[1] Before the `intraShin` algorithm is applied for each task instance, the time slot for each task instance is assigned by the off-line InterDVS algorithm described in [9].

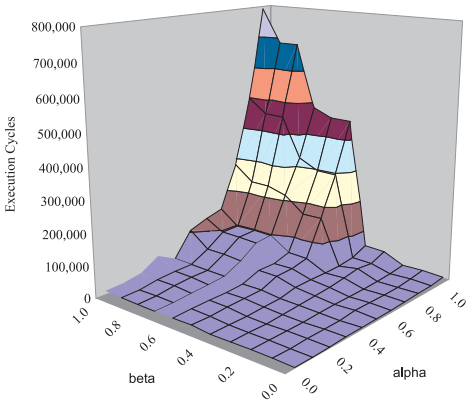


Fig. 3. Changes in the number of execution cycles when α and β vary

Table 1.) As test cases, we used two different task sets, `A` and `B`. The task set `A` is homogeneous (i.e., the tasks in `A` have similar periods and WCETs) while the task set `B` is heterogeneous (i.e., the tasks in `B` have large variations in their periods and WCETs).

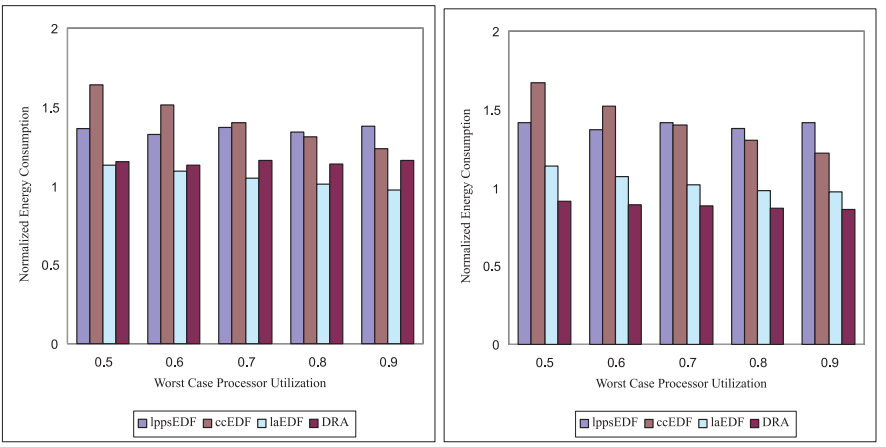
Fig. 4 shows the normalized energy consumption of the InterDVS algorithms over that of `intraShin`. Except for a few cases, the `intraShin` algorithm outperforms the InterDVS algorithms tested. We can observe that the relative performance of the InterDVS algorithms is getting worse when the worst case processor utilization gets smaller. This is because unused slack times are increasing, in the InterDVS algorithms, when WCPU becomes smaller. On the other hand, `intraShin` utilizes all the slack times, resulting in higher energy reductions.

The `DRA` algorithm's performance is significantly different with two task sets. As shown in Fig. 4(a), `DRA` outperforms both `laEDF` and `intraShin` when the task set `A` is used. However, when the task set `B` is used, Fig. 4(b) shows that `DRA`'s performance is inferior to that of `laEDF` and `intraShin`. This is because the slack estimation method used in `DRA` does not work well when the task utilizations are not uniform.

4.2 Performance Evaluation of Hybrid Methods

We have compared the energy efficiency of the InterDVS algorithms and the IntraDVS algorithm in the previous section. However, there are cases where pure IntraDVS or pure InterDVS dose not work well. Fig. 5 illustrates such cases.

In Fig. 5(a), when an InterDVS algorithm is used, the slack time generated by the task τ_1 cannot be used by the task τ_2 because the release time of the task τ_2 is same to the deadline of the task τ_1. This slack time could be used if the task τ_1 were scheduled using an IntraDVS algorithm. On the other hand, in Fig. 5(b), when an IntraDVS algorithm is used, all the slack times generated by the task τ_1 are used by the task τ_1. However, this slack distribution is unbalanced. If we



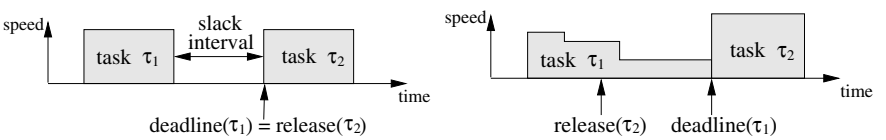
(a) Task set A (b) Task set B

Fig. 4. Normalized energy consumption of InterDVS algorithms over `intraShin`

used InterDVS, we could get a more efficient schedule by distributing the slack time of τ_1 for the task τ_2.

In this section, we investigate whether hybrid DVS algorithms (HybridDVS algorithms) with both IntraDVS and InterDVS features perform better than pure IntraDVS algorithms or pure InterDVS algorithms. Although both `intraShin` and `intraGruian` can be used for performance comparison, we use `intraShin` as the base IntraDVS algorithm. This is because `intraShin` is less likely to generate dynamic slack times, thus making the distinctions among the different HybridDVS methods clearer.

HybridDVS algorithms select either the *intra mode* or the *inter mode* when slack times are available during the execution of the current task. At the inter mode, the slack time is used not for the current task but for the following tasks. Therefore, the speed of the current task is not changed by the slack time produced by the current task. At the intra mode, all the slack time is used for the current task, reducing its own execution speed.



(a) The case where InterDVS can not utilize the slack time.

(b) The case where the slack distribution is not balanced due to IntraDVS.

Fig. 5. Cases where pure InterDVS or pure IntraDVS performs poor

Table 2. Heuristics for HybridDVS algorithms

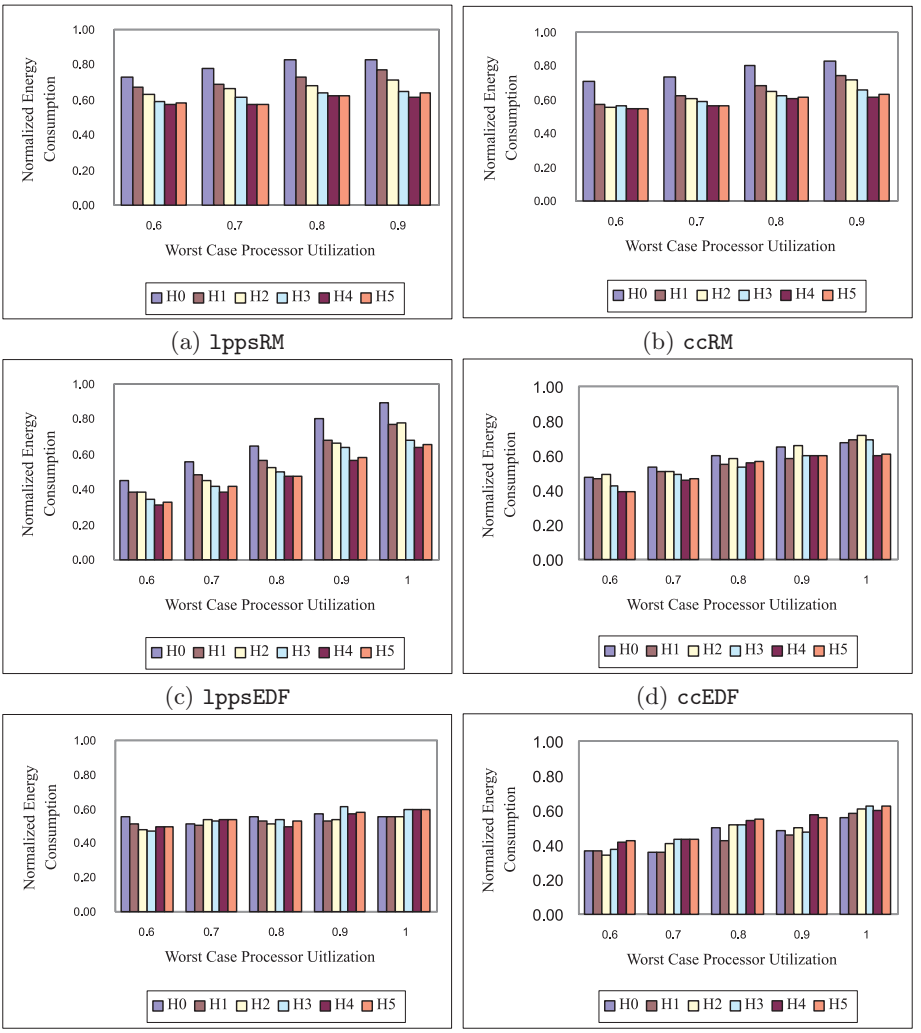
Heuristic	Description
H0	always uses the inter mode (i.e., the pure InterDVS approach).
H1	uses the inter mode as a default but uses the intra mode if no activated task exists.
H2	uses the inter mode at first, but changes into the intra mode when the unused slack time is more than a predefined amount of slack time.
H3	alternates the intra mode and the inter mode keeping the balance of slack consumption in each mode.
H4	uses the intra mode at first, but changes into the inter mode when the current task has used a predefined amount of slack time.
H5	always uses the intra mode.

Table 2 summarizes six heuristics for HybridDVS algorithms we consider in this section. The heuristics are different in that how close they are to the pure IntraDVS approach or pure InterDVS approach. H0 is identical to the pure InterDVS approach. H1 and H2 are closer to the pure InterDVS approach while H4 and H5 are closer to the pure IntraDVS approach. H1 uses the intra mode only when there is no following task which can utilize the slack time from the current task.

We have evaluated six heuristics in Table 2 with six InterDVS algorithms in Table 1. Fig. 6 shows the energy efficiency comparison results of the HybridDVS algorithms over the `power-down` method varying WCPUs. In the `power-down` method, active tasks execute with the full speed. When there is no active task, the system enters into the power-down mode. The HybridDVS algorithms, H1, H2, H3 and H4, generally reduce the energy consumption by 5∼20% over that of the pure DVS algorithms, H0 and H5.

Fig. 6 shows that the energy efficiency of HybridDVS algorithms are strongly affected by the efficiency of on-line slack estimation methods used by each InterDVS algorithm. In `laEDF`, `DRA`, `AGR`, and `lpSHE` where slack times are aggressively identified, it is a good idea that some (or all) of slack time produced by the current task is passed to the following tasks (as in Fig. 5(b)). However, in `lppsEDF/RM` and `ccEDF/RM` where slack times are less aggressively identified, there are many cases where the current slacks are wasted unless used by the current task (as in Fig. 5(a)). In this case, it is better for the current task to utilize most of the slack time generated. Therefore, if a HybridDVS algorithm is based on `laEDF`, `DRA`, `AGR`, or `lpSHE`, H1 and H2 are better choices. On the other hand, for `lppsEDF/RM` and `ccEDF/RM`, H4 and H5 are better choices.

Fig. 7 shows the spectrum of HybridDVS heuristics, and summarizes well-matching hybrid heuristics for each InterDVS algorithm. For example, if `laEDF` is extended to a HybridDVS algorithm, H1 is a good candidate for a matching hybrid heuristic. However, if `lppsRM` is modified for a hybrid DVS algorithm, H4 is a better hybrid heuristic.



(a) `lppsRM` (b) `ccRM`

(c) `lppsEDF` (d) `ccEDF`

(e) `AGR` (f) `laEDF`

Fig. 6. Energy efficiency comparison results of the HybridDVS algorithms

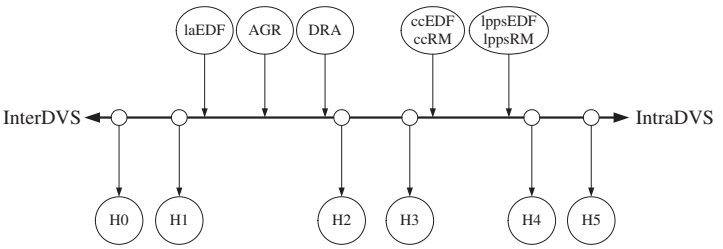


Fig. 7. Spectrum of HybridDVS heuristics

4.3 Overhead Measurement of InterDVS Algorithms

In designing an InterDVS algorithm, it is common to assume that the voltage scaling overhead is negligible. However, since efficient InterDVS algorithms generally lengthen the active execution intervals of tasks, InterDVS may affect other system performance factors, possibly causing negative side effects on the overall energy efficiency. In this section, using SimDVS, we evaluate how the InterDVS algorithm affects the number of context switches. In particular, we investigate whether it increases significantly.

The example task set in Table 3 illustrates that DVS can increase the number of context switches due to preemption. As shown in Fig. 8(a), the example task set can be scheduled with the maximum frequency f_{max} under the EDF scheduling policy. In this case, there is no preemption. When the same task set is scheduled by an InterDVS algorithm, the schedule may look like one shown in Fig. 8(b). If we assume that there is no system overhead as well as energy overhead due to extra preemption, the schedule in Fig. 8(b) consumes less energy than one in Fig. 8(a) because it operates under the slower speed f_{lower}. However, the energy-efficient schedule in Fig. 8(b) increases the number of context switches due to extra preemption. For example, since the execution time of τ_1 is increased by the InterDVS algorithm, τ_1 cannot complete its execution before $t = 2$, thus it is preempted by the second instance of τ_2.

Fig. 9 shows how the number of context switches changes with the InterDVS algorithms. The number of context switches is measured varying the number of tasks and the number of voltage levels available in the target machine. The results are normalized by the number of context switches by the `power-down` method. The `laEDF`, `DRA` and `lpSHE` algorithms show high rates of increase in preemption. When a task is preempted, the number of cache misses and memory accesses may

Table 3. An example task set

Task	Period	WCET	Actual Execution Time
τ_1	6	1	1
τ_2	2	1	0.5

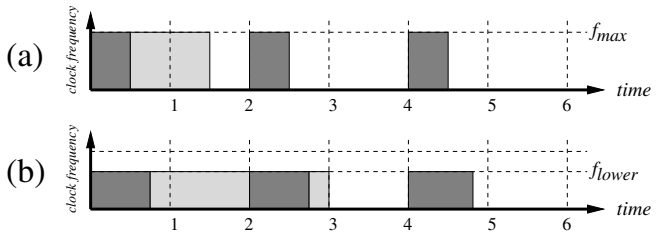
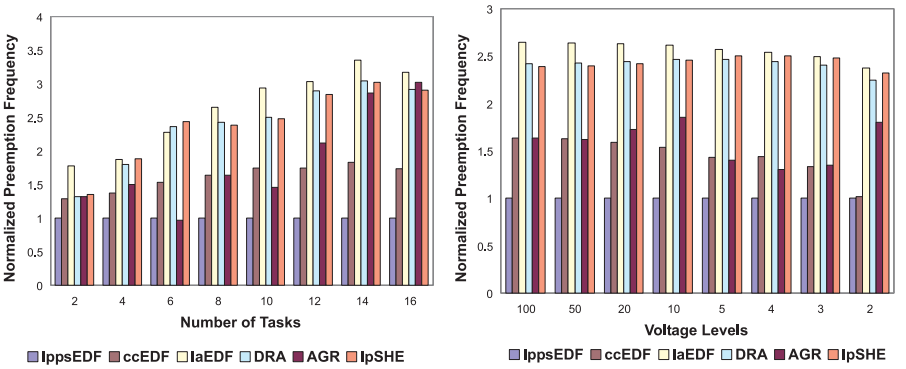


Fig. 8. Extra preemption due to DVS



(a) Number of Tasks (b) Voltage Levels

Fig. 9. Changes in the number of context switches

increase as well. Therefore, more energy is consumed in the memory and bus. Furthermore, as the number of context switches increases, the *live length*[2] of a task is lengthened. If the live length of a task is increased, more memory blocks are simultaneously required, increasing the probability of page faults.

Our experimental results indicate that when the preemption cost is considered, DVS algorithms need to be evaluated differently. For example, although `lppsEDF` is less energy-efficient than other InterDVS algorithms under the assumption of no context switching overhead, it might be more efficient when a context switch consumes a significant amount of energy because tasks under `lppsEDF` preempt each other less frequently.

5 Conclusion

In this paper, we have described SimDVS, a unified simulation environment for performance comparison of dynamic voltage scaling algorithms, and demonstrated its effectiveness as a DVS evaluation tool using three case studies. Based

[2] The time duration between the arrival time and completion time of a task instance.

on the modular design structure, SimDVS supports both IntraDVS algorithms and InterDVS algorithms and allows an easy integration of new DVS algorithms such as HybridDVS algorithms.

Using SimDVS, we compared the energy efficiency of the IntraDVS algorithm and InterDVS algorithms. Although, the IntraDVS algorithm generally outperformed the InterDVS algorithms, the relative energy efficiency was dependent on the task set characteristics.

As the second case study, we also evaluated various heuristics for the HybridDVS algorithms which use both IntraDVS and InterDVS features. The heuristics close to the pure InterDVS algorithm worked better when they are based on the aggressive InterDVS algorithms while the heuristics close to the pure IntraDVS algorithm performed better when they are based on the non-aggressive InterDVS algorithms.

Finally, we showed that more efficient DVS algorithms may suffer from more system overhead such as the context switches.

Acknowlegement

The RIACT at Seoul National University provided research facilities for this study.

References

[1] M. Fleischmann. Crusoe Power Management: Reducing The Operating Power with LongRun. In *Proc. of HotChips 12 Symposium*, 2000. 141, 146

[2] Advanced Micro Devices, Inc. AMD PowerNow Technology, 2000. 141, 146

[3] Intel, Inc. The Intel(R) XScale(TM) Microarchitecture Technical Summary, 2000. 141, 146

[4] F. Yao, A. Demers, and S. Shenker. A Scheduling Model for Reduced CPU Energy. In *Proc. of 36th Annual Symposium on Foundations of Computer Science*, pages 374–382, 1995. 141

[5] I. Hong, G. Qu, M. Potkonjak, and M. B. Srivastava. Synthesis Techniques for Low-Power Hard Real-Time Systems on Variable Voltage Processor. In *Proc. of Real-Time Systems Symposium*, pages 178–187, 1998. 141

[6] T. Ishihara and H. Yasuura. Voltage Scheduling Problem for Dynamically Variable Voltage Processors. In *Proc. of International Symposium On Low Power Electronics and Design*, pages 197–202, 1998. 141

[7] Y. Shin, K. Choi, and T. Sakurai. Power Optimization of Real-Time Embedded Systems on Variable Speed Processors. In *Proc. of International Conference on Computer-Aided Design*, pages 365–368, 2000. 141, 142

[8] H. Aydin, R. Melhem, D. Mosse, and P. M. Alvarez. Dynamic and Aggressive Scheduling Techniques for Power-Aware Real-Time Systems. In *Proc. of Real-Time Systems Symposium*, 2001. 141, 142

[9] P. Pillai and K. G. Shin. Real-Time Dynamic Voltage Scaling for Low-Power Embedded Operating Systems. In *Proc. of 18th ACM Symposium on Operating Systems Principles (SOSP'01)*, 2001. 141, 142, 148

[10] G. Quan and X. Hu. Energy Efficient Fixed-Priority Scheduling for Real-Time Systems on Variable Voltage Processors. In *Proc. of Design Automation Conference*, pages 828–833, 2001. 141

[11] D. Shin, J. Kim, and S. Lee. Intra-Task Voltage Scheduling for Low-Energy Hard Real-Time Applications. *IEEE Design and Test of Computers*, 18(23):20–30, Mar. 2001. 141, 142, 147

[12] F. Gruian. Hard Real-Time Scheduling Using Stochastic Data and DVS Processors. In *Proc. of International Symposium on Low Power Electronics and Design*, pages 46–51, 2001. 141, 142, 147

[13] 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. of Design, Automation and Test in Europe (DATE'02)*, pages 788–794, 2002. 142

[14] J. Lehoczky, L. Sha, and Y. Ding. The Rate Monotonic Scheduling Algorithm: Exact Characterization and Average Case Behavior. In *Proc. of Real-Time Systems Symposium*, pages 166–171, 1989. 146

[15] T. Burd, T. Pering, A. Stratakos, and R. Brodersen. A Dynamic Voltage Scaled Microprocessor System. In *Proc. of International Solid-State Circuits Conference*, pages 294–295, 2000. 146

[16] D. Burger and T. M. Austin. The SimpleScalar Tool Set, version 2.0. Technical Report 1342, University of Wisconsin-Madison, CS Department, Jun. 1997. 147