Journal of
Low Power Electronics
Vol. 1, 1–11, 2005

Performance Evaluation of Dynamic Voltage Scaling Algorithms for Hard Real-Time Systems

Woonseok Kim, Dongkun Shin, Han-Saem Yun, Jihong Kim,* and Sang Lyul Min

School of Computer Science and Engineering, Seoul National University ENG4190, Seoul, Korea 151–742

(Received: 26 August 2005; Accepted: 29 November 2005)

Dynamic voltage scaling (DVS) is an effective low-power design technique for embedded real-time systems, adjusting the clock speed and supply voltage dynamically. In this paper, we evaluate state-of-art DVS algorithms recently proposed for hard real-time periodic task sets. We compare the energy efficiency of the proposed DVS algorithms under various task/system configurations. Experimental results both from the simulation tool and a real H/W-based DVS platform are presented. Our results provide important insights in understanding the performance differences among the proposed DVS algorithms in a unified fashion.

Keywords: Low-Power Design, Energy-Aware Systems, Real-Time Systems, Embedded Systems, Performance Analysis.

1. INTRODUCTION

Dynamic voltage scaling (DVS), which adjusts the supply voltage and clock frequency dynamically, is an effective low-power design technique for embedded real-time systems. Since the energy consumption E of CMOS circuits has a quadratic dependency on the supply voltage, lowering the supply voltage is one of the most effective ways of reducing the energy consumption.

With the recent growth in the portable and mobile embedded device market, where a low-power consumption is an important design requirement, several commercial variable-voltage microprocessors were developed. Targeting these microprocessors, many DVS algorithms have been proposed or developed, especially for hard real-time systems.[1–8, 10–12, 17, 19] Since lowering the supply voltage also decreases the maximum achievable clock speed,[14] various DVS algorithms for hard real-time systems have the goal of reducing supply voltage dynamically to the lowest possible level while satisfying the tasks' timing constraints.

Although each DVS algorithm is shown to be quite effective in reducing the energy/power consumption of a target system under its own experimental scenarios, these recent DVS algorithms have not been quantitatively evaluated under a unified framework, making it a difficult task for low-power embedded system developers to select an appropriate DVS algorithm for a given application/system. A quantitative analysis of the energy-efficiency is particularly important because most of these DVS algorithms are based on both static and dynamic slack analysis techniques whose performance is difficult to predict analytically. In addition, their energy efficiency fluctuates significantly depending on the workload variations, task set characterizations, and execution paths taken; further requiring a quantitative comparison study.

In this paper, we quantitatively evaluate the energy efficiency of several recent DVS algorithms proposed for hard real-time systems using a unified DVS simulation environment called SimDVS.[16] In order to better observe the impact of DVS algorithms on system behavior, we also perform similar experiments using DVS Evaluation Workbench (DEW), which is an XScale-based DVS evaluation environment. We focus on preemptive hard real-time systems in which periodic real-time tasks are scheduled, under the Earliest-Deadline-First (EDF) algorithm or the Rate-Monotonic (RM) algorithm (which represent the most widely used real-time system models).

2. CLASSIFICATION OF DVS ALGORITHMS

For hard real-time systems, there are two types of voltage scheduling approaches depending on the voltage scaling granularity: intra-task DVS (IntraDVS) and inter-task DVS (InterDVS). The intra-task DVS algorithms[3, 15] adjust the voltage within an individual task boundary, while the

*Author to whom correspondence should be addressed.
Email: jihong@davinci.snu.ac.kr

Table I. Classification of DVS slack estimation techniques.

	Slack estimation technique	Scaling decision
IntraDVS	(1) Path-based method (2) Stochastic method	Off-line
InterDVS	(3) Maximum constant speed (4) Stretching to NTA (Next Task Arrival) (5) Priority-based slack-stealing (6) Utilization updating (7) Short-term work-demand analysis	On-line

inter-task DVS algorithms determine the voltage on a task-by-task basis at each scheduling point. The main difference between the two approaches is whether the slack times are used for the current task or for the tasks that follow. InterDVS algorithms distribute the slack times from the current task for the following tasks, while IntraDVS algorithms use the slack times from the current task for the current task itself.

In this paper, we focus on inter-task DVS algorithms for periodic hard real-time systems, and Table I summarizes representative slack estimation techniques used in these algorithms. When we assume periodic tasks, the slack time of tasks can be estimated in several different ways. Since the periods and WCETs of tasks are given, we can statically analyze the slack times in offline phase. During run time, additional slack times, which are generated from the difference between the WCETs and actual execution times of tasks, can be dynamically estimated. For example, based on the periodicity and WCET of tasks, we can estimate statically given slack times and exploit those slack times to lower the clock speed, i.e., the worst case processor utilization can be estimated and the clock speed can be adjusted based on that as in the maximum constant speed heuristic.[17] Since the arrival time of tasks are known *a priori*, when a single task is active, its execution can be extended to the earliest arrival time of the next task with the lowered clock speed and voltage as in the stretching to NTA (Next Task Arrival) technique.[17] Furthermore, since the WCETs of tasks are also known, we can estimate how much work should be done before the deadline of the scheduled task using the short-term work-demand analysis heuristic.[8]

While the above techniques exploit static information only (such as WCET and task period), other techniques utilize dynamic information such as workload variation of tasks. Generally, the schedulability of the given task set is tested based on tasks' WCETs in hard real-time systems (in order to guarantee the feasible schedule of tasks). However, the execution time of each task is usually less than its WCET, and the actual processor utilization during run time is usually lower than the worst case processor utilization. Thus, when a task completes its execution much earlier than its WCET, the expected processor utilization can be recalculated based on the actual execution time of completed task, and the clock speed can be adjusted based on

Table II. Target DVS algorithms.

Category	Scheduling Policy	DVS Policy	Used Methods (Numbers indicate corresponding techniques in Table I) $(n)^*$ indicates an improved version of n
InterDVS	EDF	lppsEDF [17]	$(3)+(4)$
		ccEDF [12]	(6)
		laEDF [12]	$(6)^*$
		DRA [1]	$(3)+(4)+(5)$
		AGR [1]	$(4)^*+(5)$
		lpSHE [7]	$(3)+(4)+(5)^*$
	RM	lppsRM [17]	$(3)+(4)$
		ccRM [12]	$(3)+(4)^*$
		lpWDA [8]	$(4)+(7)$

that as in utilization updating. Also, when a higher-priority task completes its execution earlier than its WCET, the following lower-priority tasks can use the slack time from the completed higher-priority task, and the clock speed can be lowered based on the slack time as in priority-based slack stealing. (A more detailed description is given in Ref. [10].)

Table II summarizes the DVS algorithms selected for the comparative study. Nine InterDVS algorithms are chosen, three[8, 12 17] of which are based on the RM scheduling policy, while the other six algorithms[1, 7, 12, 17] are based on the EDF scheduling policy. The "used methods" column of Table II shows the DVS techniques employed by each target DVS algorithm. For example, in lppsEDF and lppsRM which were proposed by Shin et al. in Ref. [17], a slack time of a task is estimated using the maximum constant speed and stretching-to-NTA methods.

The ccRM algorithm proposed by Pillai et al.[12] is similar to lppsRM in the sense that it uses both the maximum constant speed and the stretching-to-NTA methods. However, while lppsRM can adjust the voltage and clock speed only when a single task is active, ccRM extends the stretching to NTA method to the case where multiple tasks are active.

Pillai et al. also proposed two other DVS algorithms,[12] ccEDF and laEDF, for the EDF scheduling policy. These algorithms estimate a slack time of a task using the utilization updating method. While ccEDF adjusts the voltage and clock speed based on run-time variations in the processor utilization alone, laEDF takes a more aggressive approach by estimating the amount of work required to be completed before NTA.

DRA and AGR, which were proposed by Aydin et al. in Ref. [1], are two representative DVS algorithms that are based on the priority-based slack stealing method. The DRA algorithm estimates the slack time of a task using the priority-based slack stealing method along with the maximum constant speed and the stretching-to-NTA methods. Aydin et al. also extended the DRA algorithm and proposed another DVS algorithm called AGR for more aggressive slack estimation and voltage/clock scaling. In AGR, in addition to the priority-based slack stealing, more slack times are identified by computing the amount of work required to be completed before NTA.

lpSHE is another DVS algorithm based on the priority-based slack stealing method.[7] Unlike DRA and AGR, lpSHE extends the priority-based slack stealing method by adding a procedure that estimates the slack time from lower-priority tasks that were completed earlier than expected. DRA, AGR, and lpSHE algorithms are somewhat similar to one another in the sense that all of them use the maximum constant speed in the off-line phase and the stretching-to-NTA method in the on-line phase in addition to the priority-based slack stealing method.

lpWDA is (so far) the only DVS algorithm for the RM scheduling policy that is based on a more aggressive slack estimation technique.[8] In theory, a slack estimation method (such as priority-based slack stealing) can be decoupled from a scheduling policy, allowing a smooth extension of the slack estimation method developed for one scheduling policy to another. However, due to the fixed priority in RM scheduling, such an extension can be inefficient as shown in Ref. [8]. For example, the priority-based slack stealing, the majority of slack times come from the unused times of completed higher-priority tasks. However, since each task instance always has the same fixed priority in RM scheduling, this technique does not work as effectively as in EDF scheduling. That is, higher-priority tasks tend to have less slack times than lower-priority tasks in RM scheduling. It is likely that the higher the task priority is, the faster the task execution speed is, making the unbalance in execution speeds more severe and resulting in poor energy efficiency. lpWDA solves this problem by adopting the short-term work-demand analysis method. In this algorithm, for a high-priority task, small slack times are inherited from higher-priority tasks but few tasks preempt the allocated time interval for the task. On the other hand, for a low-priority task, large slack times are passed over to the low-priority task from higher-priority tasks. However, the allocated interval is frequently preempted by higher-priority tasks. Therefore, slack times are more evenly distributed.

3. EVALUATION ENVIRONMENTS

As shown in the previous section, many DVS algorithms have been proposed for hard real-time systems. In this section, we present evaluation results for several key DVS algorithms using SimDVS, a unified simulation environment for DVS algorithms. We also present analysis results obtained from actual measurements using DEW, an XScale-based DVS evaluation environment. Using actual implementation of DVS algorithms on an XScale development board, we can verify the validity of the simulation study and better understand the side effects as well as overheads of DVS, if any.

In evaluating the performance of DVS algorithms, the ultimate metric is the system-wide energy consumption. Since it is difficult to understand the power reduction effect in the system-level, we focus on two alternative metrics in this study: CPU power consumption and other side effects of using DVS. For the latter, we focus on the increase in system overhead due to voltage scaling and the memory overhead.

SimDVS is a software simulator designed for performance evaluation of hard real-time DVS algorithms. It is useful in estimating the energy efficiency of several DVS algorithms under different machine specifications for various task sets. On the other hand, DEW is an XScale-based DVS evaluation environment. Both SimDVS and DEW, which target a single-processor platform, implement all the DVS algorithms listed in Table II.

Two evaluation tools have different pros and cons. Although SimDVS can produce various simulation results of several DVS algorithms under the different machine specifications and task sets fast, it is difficult to capture the overhead and side-effects of DVS—such as context switching overhead, DVS operation delay, memory access behavior, and other delays due to the kernel service. (For a detailed description of SimDVS, see Ref. [10].)

In contrast, DEW is slower than SimDVS (because DEW runs actual applications while SimDVS performs event-driven simulation) and less flexible for experimental studies (because DEW represents a single machine specification). However, it allows to monitor real system behavior under DVS. DEW is based on an XScale evaluation board, the Intel Board DBPXA250. The Intel Board DBPXA250 includes the Intel PXA250 microprocessor which supports dynamic voltage scaling. It supports 13 levels of clock speed (from 99.5 MHz to 398.1 MHz) and four levels of voltage (from 0.85 V to 1.3 V). In DEW, tasks run on top of a POSIX-compliant embedded real-time operating system, VELOS (described in detail in www.hkmds.com). In order to estimate energy consumption and system overhead, we inserted small instrument action codes to thekernel , such as at the context switching points and kernel service routines. At each checkpoint (e.g., the start of context switching and the start of the kernel schedule), the kernel collects system overhead information and send them to the host PC through a debugging tool. Then, the host PC computes the energy consumption based on the collected trace.

4. EXPERIMENTAL RESULTS

4.1. Simulation Results

The energy efficiency of InterDVS algorithms depends significantly on the accuracy of slack estimation and the appropriateness of slack distribution. To evaluate the effectiveness of the slack estimation method used in each InterDVS algorithm, extensive experiments while varying the number of tasks are performed. Then, to evaluate the effect of slack distribution methods, experiments were performed while restricting the amount of slack time that a task can utilize.

4.1.1. Number of Tasks

To evaluate the impact of the number of tasks on the energy efficiency of DVS algorithms, experiments with varying numbers of tasks were performed. For each task set with n tasks (where $n = 2, 4, 6, \ldots, 16$), 100 task sets were randomly generated. The period and the WCET of each task were randomly generated using uniform distribution with the ranges of [10, 100] ms and [1, period] ms, respectively. To eliminate the effect of static slack times, we chose the task sets which have high worst case processor utilization (WCPU); WCPUs are equal to 1.0 for EDF InterDVS algorithms and 0.9 for RM InterDVS algorithms.[a] The execution time of each task instance was randomly drawn from a Gaussian distribution, and the resulting average case processor utilization (ACPU) was set to 0.55.

Figure 1 shows the impact of the number of tasks on the energy consumption.[b] In the figure, the y-axis indicates the normalized energy consumption value over the energy consumption of an application running on a DVS-unaware system with a power-down mode only.[c] As the number of tasks increases, the energy efficiency of lppsEDF, lppsRM, and ccRM that only use the stretching-to-NTA technique do not improve significantly, while that of the other more aggressive InterDVS algorithms improves significantly. This can be explained by the fact that, in the stretching-to-NTA method, the slack time that can be exploited is limited to the time between the completion of a task instance and the arrival time of the next task instance, which is largely independent of the number of tasks in the system. On the other hand, for the other InterDVS algorithms, since the slack times can be taken from any completed task instance, as the number of task increases, each task has more slack sources and can be scheduled with a lowered clock speed.

As shown in Figure 1, the energy efficiency of each algorithm tends to approach a limit as the number of task increases. Actually, when the task set has more than 8 tasks, the results are very similar to that of a 8-task set. We also performed experiments varying the ACPU from 0.1 to 0.9. In these experiments, overall trends are very similar to when ACPU is 0.55.[10] (In the rest of experiments,

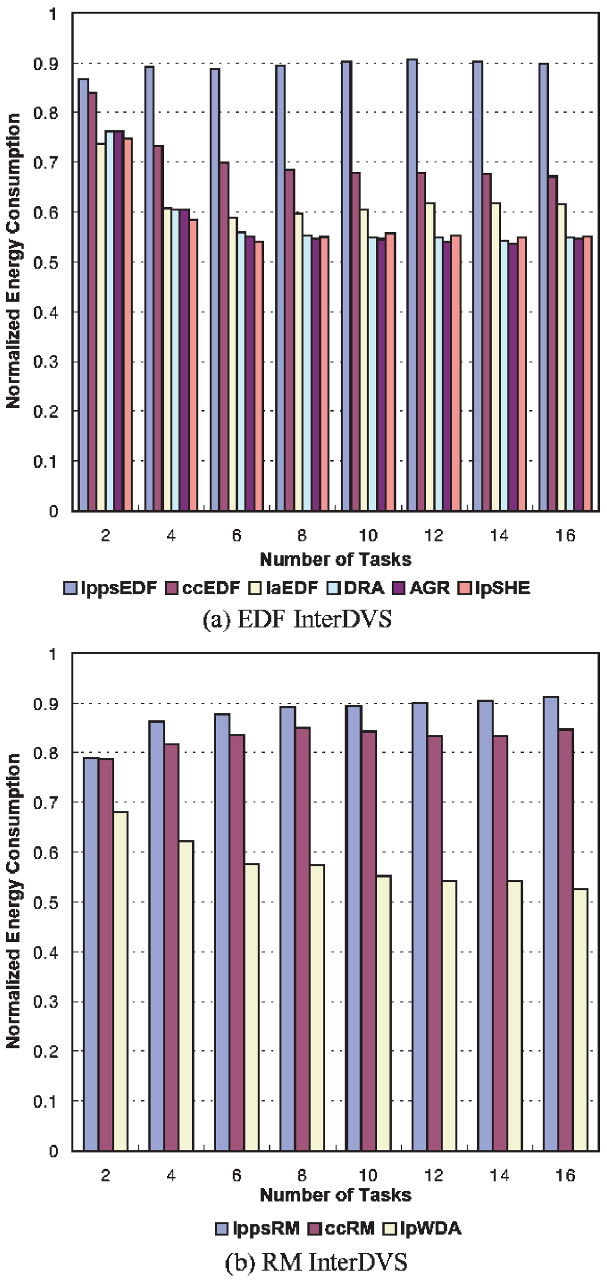


Fig. 1. Impact of the number of tasks.

we report the results only for 8-task sets with ACPU set to 0.55.)

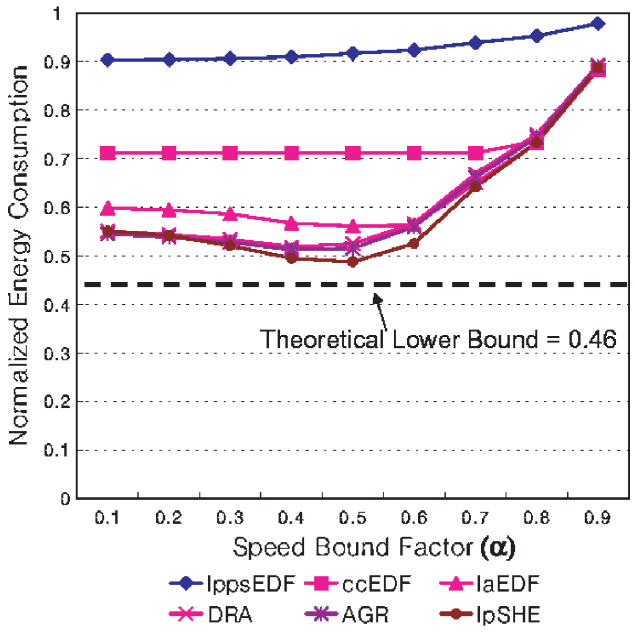
4.1.2. Speed Bound

In the previous experiments, we assumed the greedy method in the slack distribution. That is, all the slack time identified is given to the current task instance. While the greedy policy is simple, it is not the best one. For example, in aggressive InterDVS algorithms such as laEDF, AGR, and lpSHE, dynamically generated slack times may be distributed unevenly among task instances. When the current task instance exhausts its assigned slack time by

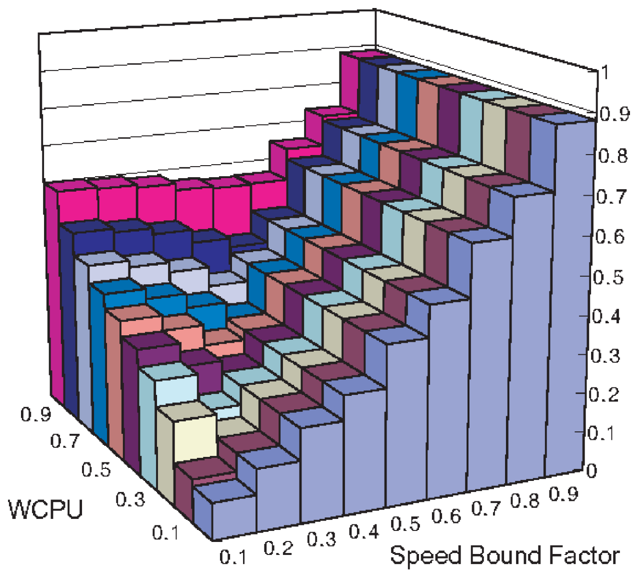
[a]Due to the nature of Rate-Monotonic Scheduling, it is not easy to get task sets which fully utilize the system. Thus, we set the high WCPU for RM InterDVS algorithms as 0.9 instead of 1.0.

[b]Unless stated otherwise in this paper, the energy consumption includes only the energy consumed in a processor core.

[c]In this case, all the tasks are scheduled with maximum clock speed, and the processor enters into the power-down mode when it is idle. The power consumption in the power down mode is assumed to be zero.



(a) Under WCPU=1.0 and ACPU=0.55



(b) Normalized energy consumption of laEDF

Fig. 2. Impact of speed bound.

the greedy distribution policy, task instances that follow may not benefit from slack times at all. In order to understand the effect of different slack distribution policies, we experimented by varying the amount of usable slack times. In the experiments, we specified the lower bound on the clock speed regardless of available slack times.[1, 10] That is, even if a DVS algorithm finds a large amount of slack time, the clock speed is limited to the pre-set lower bound.

Figure 2(a) shows the experimental results for various minimum speeds. In each experiment, it is assumed that the clock speed can be varied within the range of $[\alpha \cdot f_{\max}, f_{\max}]$ with a step size of 1 MHz where $f_{\max} = 100$ MHz and α is the speed bound factor (cf., Even though we change $f_{\max}$, the following experimental results still hold). As α becomes larger, the task instances is scheduled with lowered clock speed less aggressively because the clock scaling is restricted by $\alpha \cdot f_{\max}$. When $\alpha \cdot f_{\max}$ is close to the lowest possible clock speed of the target machine, it is similar to when the greedy slack distribution is used. The experiments were performed varying α from 0.1 to 0.9. In Figure 2(a), the x-axis indicates the speed bound factor. The energy efficiency of InterDVS algorithms (except for lppsEDF and ccEDF) is generally higher when α values are between 0.3 and 0.5. For example, when the speed bound factor is 0.5 in Figure 2(a), an improvement of $6 \sim 11\%$ was achieved over when the greedy policy is used.

In Figure 2(a), it is shown that the energy efficiency of AGR and lpSHE is very close to the theoretical lower bound[d] when the speed bound factor is near 0.5. In fact, one interesting observation is that for the aggressive InterDVS algorithms, the energy efficiency is highest when the speed bound factor was set to ACPU.

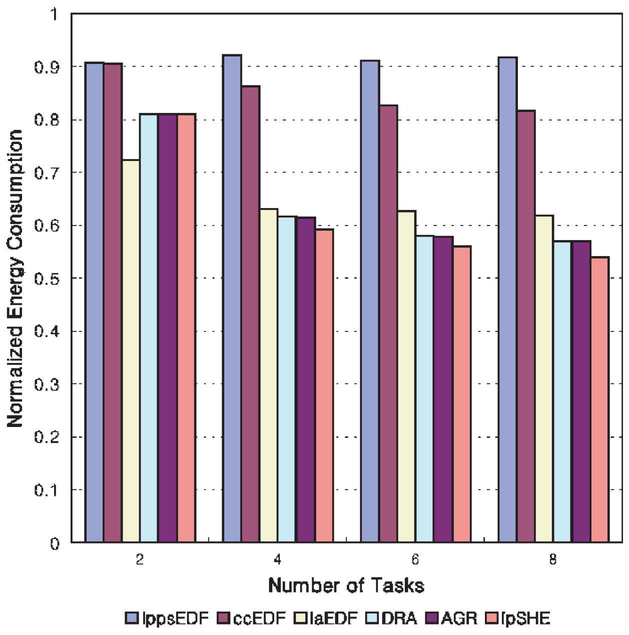
To show the relationship between the speed bound and ACPU, extensive experiments were performed for various task sets while varying ACPU and scaling bound. Figure 2(b) shows the results. (Due to the lack of space, only the results for laEDF (an example of aggressive InterDVSs) are shown. The results for AGR and lpSHE are very similar to that of laEDF. Other relevant results also can be found in Ref. [10].) The results confirm that when the selected speed bound factor is close to ACPU (=0.55 $*$ WCPU), the best energy efficiency is achieved for laEDF. Wireless Environments in Video Streaming.

4.2. Real Platform Evaluation Results

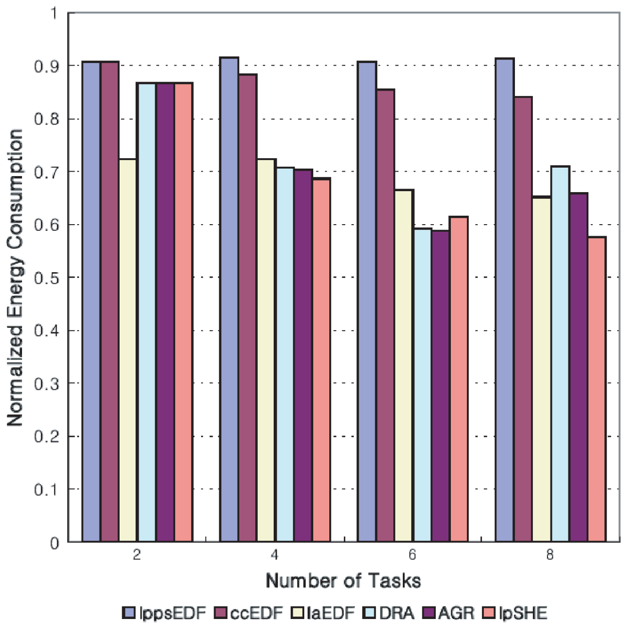
Although SimDVS is a useful tool to experiment with various scenarios under different machine configurations/task specifications (as shown in Ref. [10]), it may not accurately describe the actual system behavior of a real DVS platform. In order to validate the usefulness of SimDVS and better understand the impact of various system overheads on the energy efficiency of DVS algorithms on the real DVS platform, we performed various experiments using DEW.

Figures 3(a) and 3(b) show the normalized energy consumption for the same task sets using SimDVS and DEW, respectively. In these experiments, the same machine specification and the same energy consumption model were used. Each task set consists of $2 \sim 8$ tasks, and its WCET and ACET are set to be 1.0 and 0.5, respectively. 20 task

[d]For EDF scheduling, the theoretical lower bound is computed with the complete execution trace information using Yao's algorithm.[18] For RM scheduling, theoretical lower bound also can be computed using Quan's algorithm.[13] In both cases, the speed lower bound was not applied.



(a) Energy consumption in SimDVS



(b) Energy consumption in DEW

Fig. 3. Evaluation results using SimDVS and DEW.

sets were tested for each task set size.[e] The period of each task were randomly generated using uniform distribution with the range of [1000, 4000] ms. In the experiments using DEW, each periodic task performs simple matrix operations repeatedly. We controlled the execution time of each task instance by adjusting the loop count of matrix. The loop body consists of a 16-KB single basic block. (Since the PXA250 has a 32-way set-associative cache of 32-KB, the 16-KB program does not incur any conflict misses for a single task. However, as the number of tasks increases, the number of conflict misses increases.)

As shown in Figures 3(a) and 3(b), the overall trend on relative energy efficiency among various DVS algorithms is similar in both SimDVS and DEW, partially demonstrating the validity of SimDVS as a research tool. Especially, laEDF and lpSHE show better energy efficiency than that of others in most cases. However, absolute values on the energy consumption are not exactly same; Measurements in DEW were generally higher.

As the main sources of this difference, we consider three factors that may affect the task execution and slack estimation in DVS algorithms: (1) the impact of the system overhead, (2) the effect of system timing resolution, and (3) the influence of the cache and memory system. Using DEW, we analyze how these factors influence the energy efficiency of each DVS algorithm.

4.2.1. Impact of System Overhead

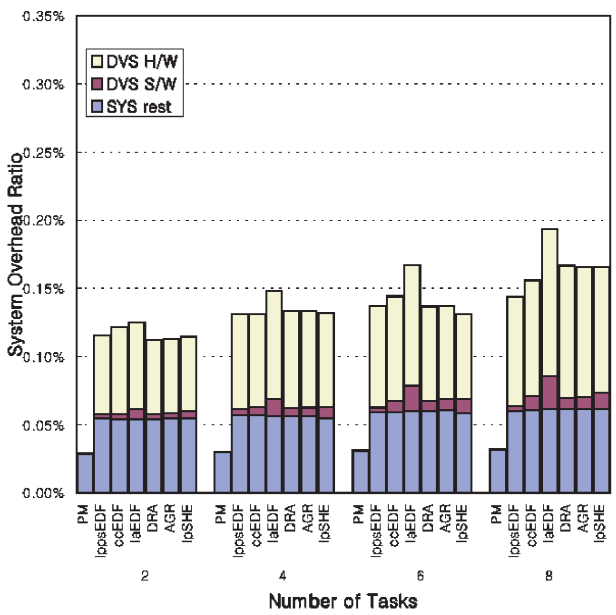
In a real DVS-enabled system, (at least) two kinds of basic overheads exist: a context switching overhead and a tick scheduler overhead. At each context switching, the DVS-enabled kernel (1) selects the next task, (2) computes the slack, (3) changes the clock/voltage, and (4) saves and restores the contexts of the previous task and the selected task, respectively. At each tick scheduling, the DVS-enabled kernel (1) increases the global system clock count, and (2) performs timer-related kernel services. When both overheads are taken into account, the task execution traces from DEW will be different from that of SimDVS.

In order to see whether the system overhead can affect the energy efficiency of a DVS algorithm, we performed additional experiments by varying the execution frequencies of tasks. We increase the task execution frequencies by shortening the periods and WCETs of the same tasks used in Figure 3. Figures 4(a), 4(b), and 4(c) show the changes of system overhead when the task execution frequencies increase by 2 times, 4 times, and 40 times, respectively.[f] In these figures, each bar represents the ratio of the execution time by the system overhead to the total execution time. In the bar, the top part $DVS_{H/W}$ represents the ratio of time delay caused by the clock/voltage scaling hardware, the middle part ($DVS_{S/W}$) indicates the ratio of extra execution times caused by the slack computation in a DVS algorithm, and the bottom part (SYS_{rest}) represents the ratio of the rest of the system overhead such as context switching and timer service. (PM in Figure 4 indicates a power-down only system.)

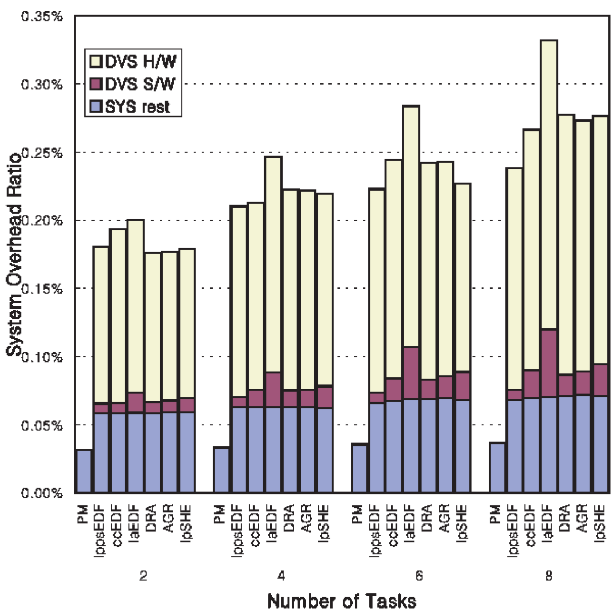
As illustrated in Figure 4, when a DVS algorithm is used, the system overhead increases as the number of tasks

[e]Since the number of task sets and the task parameters are changed, the experimental results in Figure 3 can differ from the results in Figure 1.

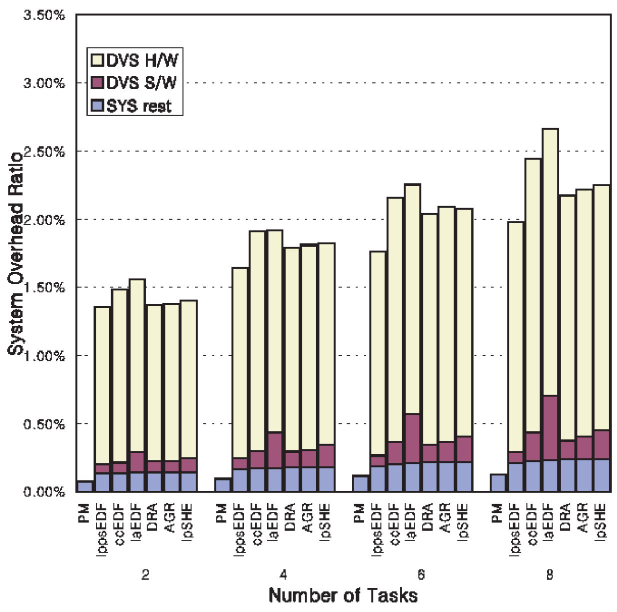
[f]For each cases, WCETs of tasks are also scaled accordingly.



(a) Long-period task set



(b) Medium-period task set



(c) Short-period task set

Fig. 4. System overhead variations in DVS algorithms.

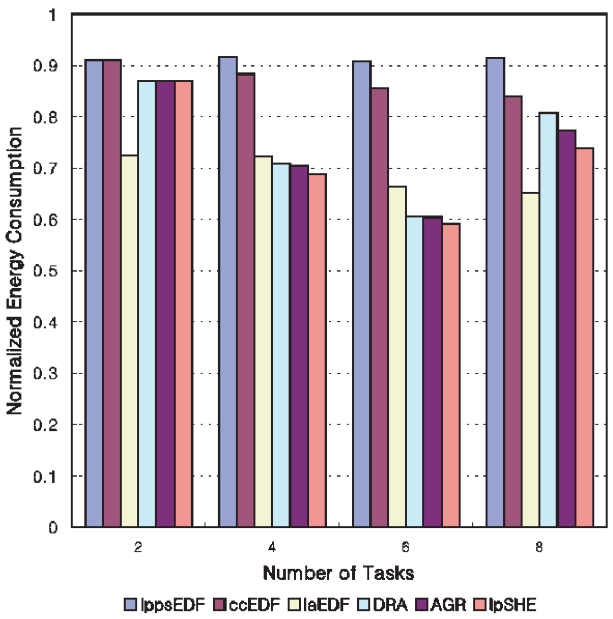
increases. For the task sets with the same number of tasks, the system overhead increases very quickly as the task execution frequency increases. In particular, as shown in Figure 4(b) and 4(c), $DVS_{H/W}$ and $DVS_{S/W}$ parts increase quickly. (Note that, the scale of y-axis in Figure 4(c) is 10 times greater than that of the others.)

It is interesting to observe that the $DVS_{H/W}$ parts are relatively larger in ccEDF and laEDF than in other algorithms. This is because ccEDF and laEDF perform the voltage scaling step more frequently. In ccEDF and laEDF, voltage scaling steps are executed additionally when each task is activated.

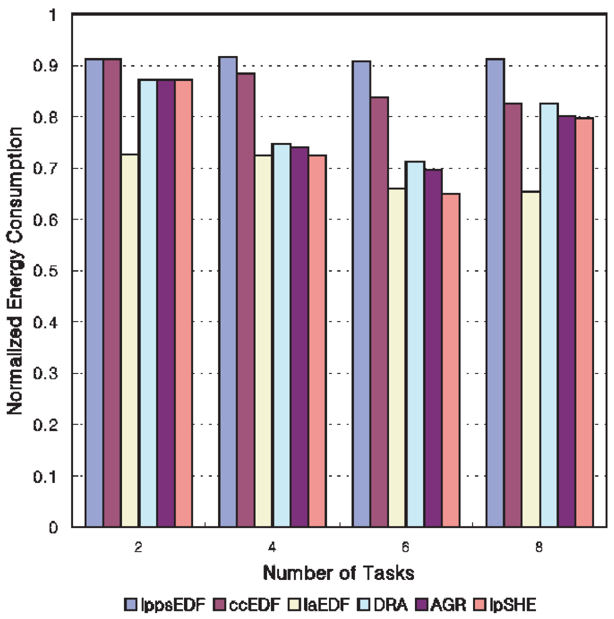
Figures 5(a), 5(b), and 5(c) show the changes in the energy efficiency of DVS algorithms when the execution frequency is increased. In DRA, AGR, and lpSHE, the increased system overhead (due to the increased execution frequency) significantly affect the energy efficiency. However, in lppsEDF, ccEDF, and laEDF, the energy efficiency is less sensitive to the increased execution frequency.

4.2.2. *Impact of Timing Resolution*

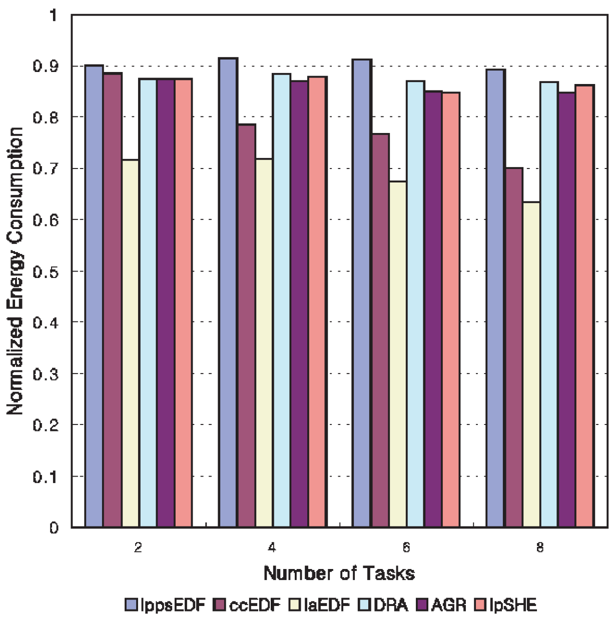
One of the major differences between SimDVS and DEW is the timing resolution. While DEW is based on a discrete time model, SimDVS assumes a continuous time model.



(a) Long-period task set



(b) Medium-period task set



(c) Short-period task set

Fig. 5. Energy efficiency variations of DVS algorithms.

In the kernel of DEW, the global clock count increases at every 10 ms, as with all other timing services. Therefore, the execution times and periods of tasks are specified in the unit of 10 ms. Although a discrete timing resolution does not affect the overall schedule of tasks significantly, it can change the accuracy of slack computation in a DVS algorithm, thus influencing the algorithm's energy efficiency.

In DRA, AGR, and lpSHE, slack times are computed based on the remaining WCETs of activated task instances and unused times of completed task instances. Since the remaining WCETs and unused times of task instances are expressed in the number of timing tick intervals, there can be a discrepancy between the estimated slack value and the theoretically available slack time. For example, even if a task executed 15 ms, its remaining WCET is decreased by only 10 ms (instead of 15 ms) because of the 10 ms tick interval.

On the contrary, in ccEDF and laEDF, slack times are estimated based on the system's local utilization, which is computed based on a real number (i.e., not an integer value). Thus, even though the timing tick is 10 ms, a slack can be computed accurately.

Figures 5(a) and 5(c) also illustrate the impact of timing resolution on the energy efficiency. When the ratio of the timing tick interval to the tasks' WCETs is relatively small as in Figure 5(c),[g] DRA, AGR, and lpSHE perform worse than ccEDF and laEDF. This is an opposite to result to the SimDVS result. As shown in Figure 1(a), ccEDF and laEDF usually perform worse than DRA, AGR, and lpSHE in SimDVS.

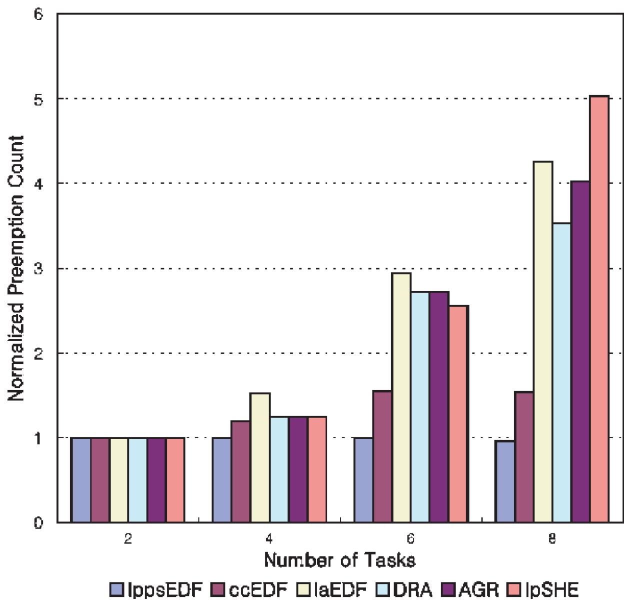
4.2.3. Impact of Memory Behavior

Since a DVS algorithm generally lowers the task execution speed, the execution time of the task will be increased under a DVS-enabled RTOS. Although the lowered execution speed is desirable for reducing the energy consumption, it can introduce negative side effects as well. One such a side effect is an increase in the number of task preemptions, which, in turn, increases the number of memory accesses.

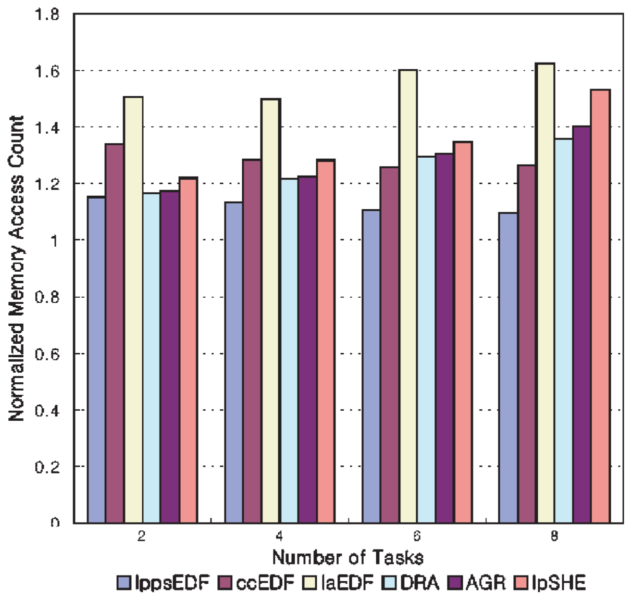
In order to see the impact of a DVS algorithm on the context switching frequency and memory energy consumption, we measured the preemption count and memory access count for the same task sets used in Figure 3(b). Figures 6(a) and 6(b) show the results. In both figures, the preemption count and memory access count are normalized to that of a power-down only system. As shown in Figure 6(a), the preemption count increases with increasing number of tasks. Especially, in aggressive algorithms such as laEDF, DRA, AGR, and lpSHE, the number of preemptions increases more rapidly than the others. For example, in lpSHE, the preemption count increases roughly to 5 times of that of PM.

We also measured the memory access count using the hardware performance monitoring counters available on the PXA250. As shown in Figure 6(b), all the DVS algorithms require more memory accesses than PM. In ccEDF and laEDF, the increases in memory accesses can be attributed to two sources: (1) the increase in the number of preemptions and (2) the increase in memory accesses from the algorithm itself. The 2-task set of Figure 6(a) and Figure 6(b) show that the latter source is also significant. Since ccEDF and laEDF perform the voltage scaling step more frequently, they require more memory accesses. In DRA, AGR, and lpSHE, memory access counts increase as their preemption counts increase. In worst case, memory access counts increase up to 55%.

The increase in memory accesses will result in the increase in the energy consumed in memory system. For example, if DRAMs were used as the memory system (where the energy consumption is proportional to the number of accesses), the memory energy consumption may increase up to 55% due to DVS (in the worst case).



(a) Normalized preemption count



(b) Memory access ratio

Fig. 6. Changes in memory system behavior.

Our measurements show that the memory system behavior should be carefully considered if a DVS algorithm can be an effective low-power technique. For example, depending on the characteristics of the memory system, it might be better to use the simple DVS algorithm such as lppsEDF or ccEDF than more aggressive ones for overall system energy savings.

The aggressive DVS algorithms may increase the energy consumption not only in memory subsystem but also in other peripherals. For example, the lengthened task lifetime due to lowered clock speed may increase the leakage energy consumption in peripheral devices (related to the task). For these preemption and memory related issues,[9]

[g]In the 8-task set of Figure 5(c), the range of tasks' WCET is [20, 90] ms where the tick interval used is 10 ms.

gives simple heuristics that can reduce such side effects of DVS.

5. CONCLUSION

We have compared the energy efficiency of recent DVS algorithms for hard real-time periodic tasks, and analyzed the impact of these algorithms on system behavior. Our comparative study (using software simulation) shows the existing DVS algorithms such as laEDF, AGR, and lpSHE are theoretically close to optimal. However, as shown in the experiments using real platform, when these algorithms incur too much system overhead, their energy efficiency can be degraded, further increasing the energy consumption of other components in the system.

Our study is the first detailed performance evaluation work of DVS algorithms for hard real-time systems, covering both the simulation-based analysis and the real platform-based analysis. Our experiments does not cover all the possible cases for various task execution time distributions and hardware platforms (e.g., multi-processor platform). However, we believe that, based on the findings of our evaluation, the existing DVS algorithms can be further improved. For example, since DVS algorithms are shown to interact with memory systems (often in a negative fashion), it will be an interesting future work to make the DVS algorithms more dynamically adaptive to the behavior of the memory system.

Acknowledgments: This research was supported by the MIC (Ministry of Information and Communication), Korea, under the ITRC (Information Technology Research Center) support program supervised by the IITA (Institute of Information Technology Assessment) (IITA-2005-C1090-0502-0031), and by the Ubiquitous Autonomic Computing and Network Project, 21st Century Frontier R&D Program in Korea. All the correspondence should be sent to a corresponding author, Jihong Kim (jihong@davinci.snu.ac.kr).

References

1. H. Aydin, R. Melhem, D. Mosse, and P. M. Alvarez, Dynamic and aggressive scheduling techniques for power-aware real-time systems. *Proceedings of IEEE Real-Time Systems Symposium* (**2001**), pp. 95–105.
2. A. Dudani, F. Mueller, and Y. Zhu, Energy-conserving feedback EDF scheduling for embedded systems with real-time constraints. *Proceedings of ACM SIGPLAN Joint Conference Languages, Compilers, and Tools for Embedded Systems (LCTES'02) and Software and Compilers for Embedded Systems (SCOPES'02)* (**2002**), pp. 213–222.
3. F. Gruian, Hard real-time scheduling using stochastic data and DVS processors. *Proceedings of the International Symposium on Low Power Electronics and Design* (**2001**), pp. 46–51.
4. I. Hong, M. Potkonjak, and M. Srivastava, On-line scheduling of hard real-time tasks on variable voltage processor. *Proceedings of the IEEE/ACM International Conference on Computer Aided Design* (**1998**).
5. I. Hong, G. Qu, M. Potkonjak, and M. B. Srivastava, Synthesis techniques for low-power hard real-time systems on variable voltage processor. *Proceedings of the IEEE Real-Time Systems Symposium* (**1998**), pp. 178–187.
6. D. Kang, S. Crago, and J. Suh, A zast resource synthesis technique for energy-efficient real-time systems. *Proceedings of IEEE Real-Time Systems Symposium* (**2002**), pp. 225–234.
7. W. Kim, J. Kim, and S. L. Min, A dynamic voltage scaling algorithm for dynamic-priority hard real-time systems using slack time analysis. *Proceedings of Design, Automation and Test in Europe* (**2002**), pp. 788–794.
8. W. Kim, J. Kim, and S. L. Min, Dynamic voltage scaling algorithm for fixed-priority real-time systems using work-demand analysis. *Proceedings of the International Symposium on Low Power Electronics and Design* (**2003**), pp. 396–401.
9. W. Kim, J. Kim, and S. L. Min, Preemption-aware dynamic voltage scaling in hard real-time systems. *Proceedings of the International Symposium on Low Power Electronics and Design (ISLPED)* (**2004**).
10. W. Kim, D. Shin, J. Jeon, J. Kim, and S. L. Min, Performance comparison of dynamic voltage scaling algorithms for hard real-time systems. *Proceedings of Real-Time and Embedded Technology and Applications Symposium* (**2002**), pp. 219–228.
11. Y.-H. Lee and C. M. Krishna, Voltage-clock scaling for low energy consumption in real-time embedded systems. *Proceedings of the Real-Time Computing Systems and Applications* (**1999**), pp. 272–279.
12. P. Pillai and K. G. Shin, Real-time dynamic voltage scaling for low-power embedded operating systems. *Proceedings of 18th ACM Symposium on Operating Systems Principles* (**2001**), pp. 89–102.
13. G. Quan and X. S. Hu, An optimal voltage schedule for real-time systems on a variable voltage processor. *Proceedings of the Design, Automation and Test in Europe* (**2002**), pp. 782–787.
14. T. Sakurai and A. Newton, Alpha-power law MOSFET model and its application to CMOS inverter delay and other formulas. *IEEE Journal of Solid State Circuits* (**1990**), Vol. 25, pp. 584–594.
15. D. Shin, J. Kim, and S. Lee, Intra-task voltage scheduling for low-energy hard real-time applications. *IEEE Design and Test of Computers* (**2001**), Vol. 18, pp. 20–30.
16. D. Shin, W. Kim, J. Jeon, J. Kim, and S. L. Min, SimDVS: An integrated simulation environment for performance evaluation of dynamic voltage scaling algorithms. *Proceedings of Workshop on Power-Aware Computer Systems* (**2002**).
17. Y. Shin, K. Choi, and T. Sakurai, Power optimization of real-time embedded systems on variable speed processors. *Proceedings of the International Conference on Computer-Aided Design* (**2000**), pp. 365–368.
18. F. Yao, A. Demers, and A. Shenker, A scheduling model for reduced CPU energy. *Proceedings of the IEEE Foundations of Computer Science* (**1995**), pp. 374–382.
19. Y. Zhu and F. Mueller, Feedback EDF scheduling exploiting dynamic voltage scaling. *Proceedings of the Real-Time and Embedded Technology and Applications Symposium* (**2004**), pp. 84–93.

Woonseok Kim

Woonseok Kim *received B.S. degree in Computer Engineering from the Hongik University, Seoul, Korea, and received M.S degree and Ph.D. degree in Computer Science and Engineering from the Seoul National University, Seoul, Korea. He is a Senior Engineer at Samsung Electronics Co., Seoul, Korea. His research interests include low-power systems and embedded real-time systems.*

Dongkun Shin

Dongkun Shin *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, all from the Seoul National University, Seoul, Korea. He is a Senior Engineer at Samsung Electronics Co., Seoul, Korea. His research interests include low-power systems, computer architecture, and embedded and real-time systems.*

Han-Saem Yun

Han-Saem Yun *received the B.S. degree in Electrical Engineering, the M.S. degree in Computer Science, and the Ph.D. degree in Computer Science and Engineering, all from the Seoul National University, Seoul, Korea. His research interests include low-power systems, computer architecture, and embedded and real-time systems.*

Jihong Kim

Jihong Kim *received his B.S. degree in Computer Science and Statistics from Seoul National University in 1986, and M.S. and Ph.D. degree in Computer Science and Engineering from the University of Washington in 1988 and 1995, respectively. He is an associate professor in the School of Computer Science and Engineering, Seoul National University. His research interests include low-power systems, networked embedded systems, computer architecture, image/multimedia systems and real-time systems. He is a member of the IEEE Computer Society and ACM.*

Sang Lyul Min

Sang Lyul Min *received a B.S. degree from the Dept. of Computer Engineering, Seoul National University, in 1983; an M.S. degree from the Dept. of Computer Engineering, Seoul National University, in 1985; and a Ph.D. degree from the Dept. of Computer Science and Engineering, the University of Washington, in 1989. He is a Professor in the School of Computer Science and Engineering at Seoul National University. His current research interests are in embedded systems, real-time systems, low-power systems, computer architecture, operating systems, and computer performance evaluations.*