Quantitative Analysis of Dynamic Voltage Scaling Algorithms for Hard Real-Time Systems

Woonseok Kim Jihong Kim Sang Lyul Min

School of Computer Science and Engineering
Seoul National University ENG4190, Seoul, Korea, 151-742
Tel : 02-880-1831 Fax : 02-885-7296
wskim@archi.snu.ac.kr, jihong@davinci.snu.ac.kr, symin@dandelion.snu.ac.kr

Abstract

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 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.

I. 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 a 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 [4, 5, 14, 1, 9, 12, 2, 6, 7]. Since lowering the supply voltage also decreases the maximum achievable clock speed [11], 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 [13]. In order to better observe the impact of DVS algorithms on system behaviors, 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 [8]).

II. 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 [12, 2] adjust the voltage within an individual task boundary, while the inter-task DVS algorithms determine the voltage on a task-by-task basis at each scheduling point. The main difference between 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. Table 1 summarizes representative techniques used in existing DVS algorithms.

Table 2 summarizes the DVS algorithms selected for the comparative study. Nine InterDVS algorithms are chosen,

Table 1. Classification of DVS techniques.

	Voltage Scaling Methods	Scaling Decision
IntraDVS	(1) Path-based method	Off-Line
	(2) Stochastic method	
InterDVS	(3) Maximum constant speed	
	(4) Stretching to NTA	On-Line
	(5) Priority-based slack-stealing	
	(6) Utilization updating	
	(7) Short-term work-demand analysis	

Table 2. Target DVS algorithms.

Category	Scheduling Policy	DVS Policy	Used Methods[†]
InterDVS	EDF	`lppsEDF` [14]	(3)+(4)
		`ccEDF` [9]	(6)
		`laEDF` [9]	$(6)^*$
		`DRA` [1]	(3)+(4)+(5)
		`AGR` [1]	$(4)^*$+(5)
		`lpSHE` [6]	(3)+(4)+$(5)^*$
	RM	`lppsRM` [14]	(3)+(4)
		`ccRM` [9]	(3)+$(4)^*$
		`lpWDA` [7]	(4)+(7)

[†] Numbers indicate corresponding techniques in Table 1.

$(n)^*$ indicates an improved version of n.

three [14, 9, 7] of which are based on the RM scheduling policy, while the other six algorithms [14, 9, 1, 6] are based on the EDF scheduling policy. The "used methods" column of Table 2 shows the DVS techniques employed by each target DVS algorithm. For example, in `lppsEDF` and `lppsRM` which were proposed by Shin *et al.* in [14], a slack time of a task is estimated using the maximum constant speed and stretching-to-NTA methods.

III. 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.

SimDVS is a software simulator designed for performance evaluation of hard real-time DVS algorithms. It is useful in estimating the energy efficiencies 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 implement all the DVS algorithms listed in Table 2.

Two evaluation tools have different pros and cons each other. 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.

On the other hand, DEW is slower than SimDVS (because DEW runs actual applications) and less flexible for experimental studies (because DEW represents a single machine specification). However, it allows to monitor real system behaviors under DVS.

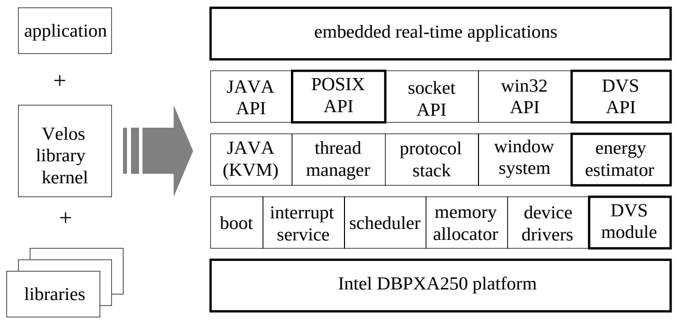


Figure 1. Overview of DEW.

Figure 1 shows the overview of DEW. (For the detailed description of SimDVS, refer [13].) DEW is based on an XScale evaluation board, Intel Board DBPXA250. Intel Board DBPXA250 includes Intel PXA250 microprocessor which supports dynamic voltage scaling. In DEW, tasks run on top of a POSIX-compliant embedded real-time operating system, VELOS [3].

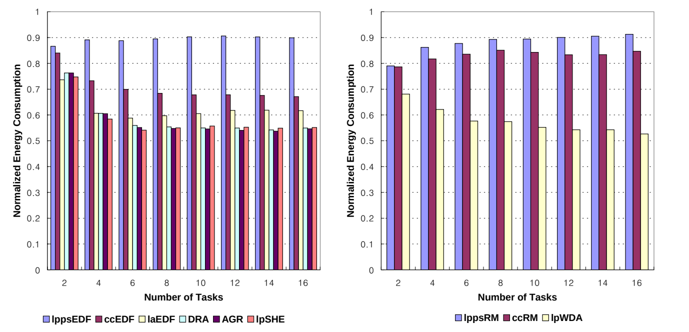
IV. Experimental Results

A. 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.

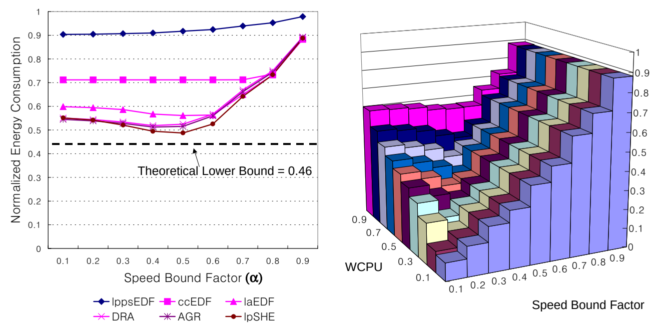
Number of Tasks

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, ..., 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. 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



(a) EDF InterDVS (b) RM InterDVS

Figure 2. Impact of the number of tasks.



(a) Under WCPU=1.0 and ACPU=0.55 (b) Normalized energy consumption of `laEDF`

Figure 3. Impact of speed bound.

0.55.

Figure 2 shows the impact of the number of tasks on the energy consumption [1]. 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. 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.

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 Inter-DVS algorithms such as laEDF, AGR and lpSHE, slack times may be distributed unevenly among task instances. When the current task instance exhausts its assigned slack time by 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.

Figure 3(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 with a step size of 1 MHz where = 100 MHz and is the speed bound factor. As becomes larger, the task instances is scheduled with lowered clock speed less aggressively because the clock scaling is restricted by. When 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 from 0.1 to 0.9. In Figure 3(a), the x-axis indicates the speed bound factor. The energy efficiency of InterDVS algorithms (except for lppsEDF and ccEDF) is generally higher when values are between 0.3 and 0.5. For example, when the speed bound factor is 0.5 in Figure 3(a), an improvement of 6 11% was achieved over when the greedy policy is used.

In Figure 3(a), it is shown that the energy efficiency of AGR and lpSHE is very close to the theoretical lower bound [2] 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.

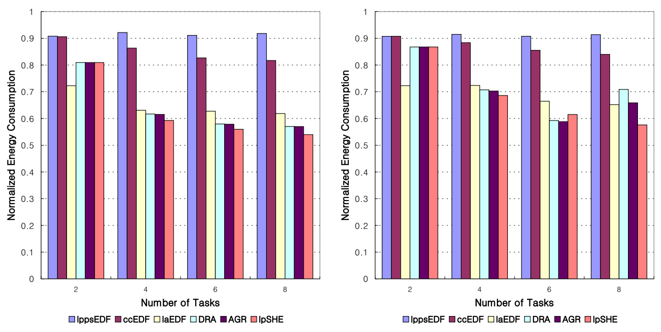
To show the relationship between the speed bound and ACPU, extensive experiments were performed for various task sets while varying ACPU and scaling bound. Figure 3(b) shows the results. (Due to the lack of space, only the results for laEDF (an example of aggressive InterDVSs) are shown. The results for AGR and lpSHE are very similar to that of laEDF.) 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.

B. Real Platform Evaluation Results

Although SimDVS is a useful tool to experiment with various scenarios under different machine configurations/task specifications, 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

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

[2] For EDF scheduling, the theoretical lower bound is computed with the complete execution trace information using Yao's algorithm [15]. For RM scheduling, the theoretical lower bound also can be computed using Quan's algorithm [10].



(a) Energy consumption in SimDVS

(b) Energy consumption in DEW

Figure 4. Evaluation results using SimDVS and DEW.

using DEW.

Figures 4(a) and 4(b) show the normalized energy consumption for the same task sets using SimDVS and DEW, respectively. In these experiments, the same machine specification and same energy consumption model were used. Each task set consists of 2 8 tasks, and its WCET and ACET are set to be 1.0 and 0.5, respectively. 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 operations. The loop body consists of a 16-KB single basic block. (Since 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 4(a) and 4(b), the overall trend on relative energy efficiencies among various DVS algorithms is similar in both SimDVS and DEW, partially demonstrating the validity of SimDVS as a research tool. 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.

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 4. Figures 5(a), 5(b), and 5(c) show the changes of system overhead when the task execution frequencies increase by 2 times, 4 times, and 40 times, respectively. 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 represents the ratio of time delay caused by the clock/voltage scaling hardware, the middle part indicates the ratio of extra execution times caused by the slack computation in a DVS algorithm, and the bottom part represents the ratio of the rest of the system overhead such as context switching and timer service. (PM in Figure 5 indicates a power-down only system.)

As illustrated in Figure 5, when a DVS algorithm is used, the system overhead increases as the number of tasks 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 5(b) and 5(c), and parts increase quickly. (Note that, the scale of y-axis in Figure 5(c) is 10 times greater than that of the others.)

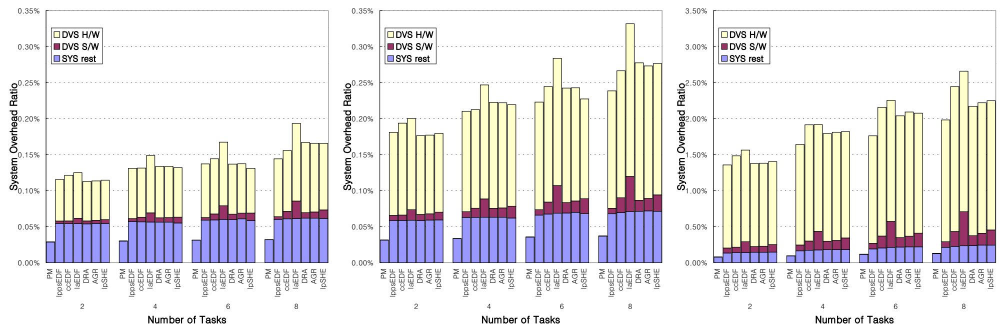
It is interesting to observe that the parts are relatively larger in ccEDF and laEDF than in other algorithms. This is because ccEDF and laEDF perform the voltage scaling step more frequently. In ccEDF and laEDF, voltage scaling steps are executed additionally when each task is activated.

Figures 6(a), 6(b), and 6(c) show the changes in the energy efficiencies 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 efficiencies are less sensitive to the increased execution frequency.

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. 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 re-



(a) Long-period task set (b) Medium-period task set (c) Short-period task set

Figure 5. System overhead variations in DVS algorithms.

maining 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.

Figure 6(a) and 6(c) also illustrate the impact of timing resolution on the energy efficiencies. When the ratio of the timing tick interval to the tasks' WCETs is relatively small as in Figure 6(c)[3], DRA, AGR and lpSHE perform worse than ccEDF and laEDF. This is an opposite to result to the SimDVS result. As shown in Figure 6(a), ccEDF and laEDF perform poorly than DRA, AGR and lpSHE in SimDVS.

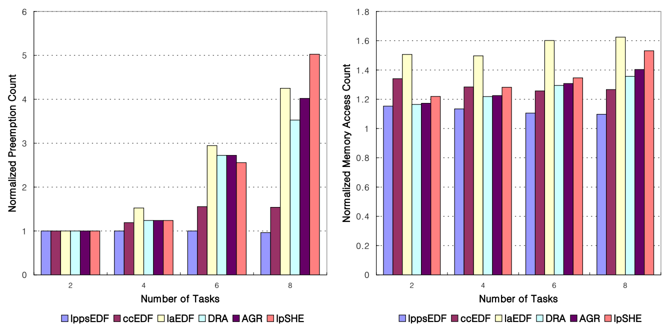
Impact of Memory Behavior

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 4(b). Figures 7(a) and 7(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 7(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 5 times compared to that of PM.

We also measured the memory access count using the hardware performance monitoring counters available on PXA250. As shown in Figure 7(b), all the DVS algorithms require more

[3]In the 8-task set of Figure 6(c), the range of tasks' WCET is [20,90]ms where the tick interval used is 10ms.

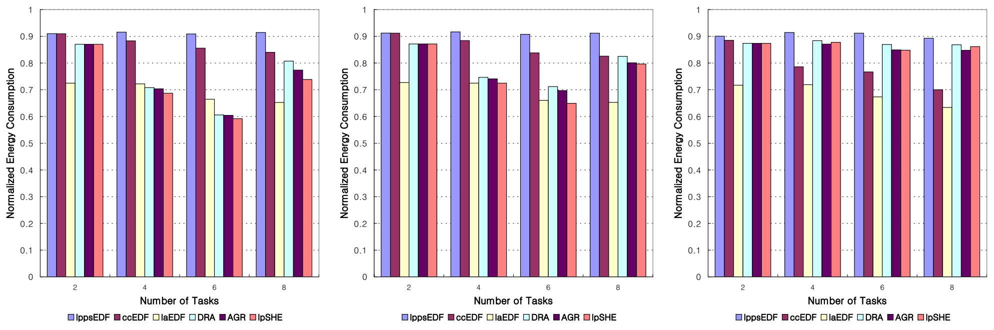


(a) Normalized preemption count (b) Memory access ratio

Figure 7. Changes in memory system behaviors.

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 7(a) and Figure 7(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 along with the number of tasks.

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. 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.



(a) Long-period task set (b) Medium-period task set (c) Short-period task set

Figure 6. Energy efficiency variations of DVS algorithms.

V. Conclusion

We have compared the energy efficiencies of recent DVS algorithms for hard real-time periodic tasks, and analyzed the impact of these algorithms on system behaviors. Our comparative study shows that the existing EDF InterDVS algorithms such as laEDF, AGR, and lpSHE are theoretically close to optimal. We also evaluated the performance of the DVS algorithms based on an XScale-based research platform. Our analysis results based on actual measurements show that a DVS algorithm may negatively influence the system overheads as well as the energy consumption in the memory system.

Our study is the first comprehensive performance evaluation work of DVS algorithms for hard real-time systems, covering both the simulation-based analysis and the real platform-based analysis. Based on the findings of our evaluation, the existing DVS algorithms can be further improved as well. 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 on the behavior of the memory system.

References

[1] H. Aydin, R. Melhem, D. Mosse, and P. M. Alvarez. Dynamic and Aggressive Scheduling Techniques for Power-Aware Real-Time Systems. In *Proceedings of IEEE Real-Time Systems Symposium*, pages 95–105, December 2001.

[2] F. Gruian. Hard Real-Time Scheduling Using Stochastic Data and DVS Processors. In *Proceedings of the International Symposium on Low Power Electronics and Design*, pages 46–51, August 2001.

[3] Hankook MDS Corporation. VELOS: POSIX-Compliant Embedded Real-Time Operating System. *http://www.hkmds.com*, June 2003.

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

[5] T. Ishihara and H. Yasuura. Voltage Scheduling Problem for Dynamically variable voltage processors. In *Proceedings of the International Symposium on Low Power Electronics and Design*, pages 197–202, August 1998.

[6] 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 *Proceedings of Design, Automation and Test in Europe*, pages 788–794, March 2002.

[7] W. Kim, J. Kim, and S. L. Min. Dynamic Voltage Scaling Algorithm for Fixed-Priority Real-Time Systems Using Work-Demand Analysis, *to appear in Proceedings of the International Symposium on Low Power Electronics and Design*. pages 396–401, August 2003.

[8] W.-S. Liu. *Real-Time Systems*. Prentice Hall, Englewood Cliffs, NJ, June 2000.

[9] P. Pillai and K. G. Shin. Real-Time Dynamic Voltage Scaling for Low-Power Embedded Operating Systems. In *Proceedings of 18th ACM Symposium on Operating Systems Principles*, pages 89–102, October 2001.

[10] G. Quan and X. S. Hu. An Optimal Voltage Schedule for Real-Time Systems on a Variable Voltage Processor. In *Proceedings of the Design, Automation and Test in Europe*, pages 782–787, March 2002.

[11] T. Sakurai and A. Newton. Alpha-power Law MOSFET Model and Its Application to CMOS Inverter Delay and Other Formulas. *IEEE Journal of Solid State Circuits*, 25(2):584–594, 1990.

[12] D. Shin, J. Kim, and S. Lee. Intra-Task Voltage Scheduling for Low-Energy Hard Real-Time Applications. *IEEE Design and Test of Computers*, 18(2):20–30, March 2001.

[13] 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. In *Proceedings of Workshop on Power-Aware Computer Systems*, February 2002.

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

[15] F. Yao, A. Demers, and A. Shenker. A Scheduling Model for Reduced CPU Energy. In *Proceedings of the IEEE Foundations of Computer Science*, pages 374–382, 1995.