Exploiting Synchronous and Asynchronous DVS for Feedback EDF Scheduling on an Embedded Platform

YIFAN ZHU and FRANK MUELLER, North Carolina State University

Contemporary processors support dynamic voltage scaling (DVS) to reduce power consumption by varying processor voltage/frequency dynamically. We develop power-aware feedback-DVS algorithms for hard real-time systems that adapt to dynamically changing workloads. The algorithms lower execution speed while guaranteeing timing constraints.

We study energy consumption for synchronous and asynchronous DVS switching on a PowerPC board. Energy, measured via data acquisition, is reduced up to 70% over naïve DVS for our feedback scheme with 24% peak savings over previous algorithms. These results, albeit differing in quantity, confirm trends observed under simulation. They are the first of their kind on an embedded board.

Categories and Subject Descriptors: D.4.1 [**Operating Systems**]: Process Management—*scheduling*; D.4.7 [**Operating Systems**]: Organization and Design—*real-time systems and embedded systems*

General Terms: Algorithms, Experimentation

Additional Key Words and Phrases: Real-Time Systems, Scheduling, Dynamic Voltage Scaling, Feedback Control

1. INTRODUCTION

Energy consumption has always been a critical issue for embedded and mobile systems where the system service time largely depends on the limited battery capacity. Software mechanisms have been widely exploited to reduce the energy consumption of those embedded systems, especially the energy costs from processors and I/O devices. Some coarse-grained power saving solutions have been implemented in operating systems, which turn off the CPU or I/O devices whenever the system idle time exceeds a certain threshold. These solutions restrict the system functionality when the CPU or I/O devices are shut down, usually resulting in much longer response times for external requests. Therefore, fine-grained power saving mechanisms have been studied in recent years for a better trade-

This work was supported in part by NSF grants CCR-0208581, CCR-0310860 and CCR-0312695.

Authors' address: Y. Zhu, F. Mueller, Department of Computer Science and Center for Embedded Systems Research, North Carolina State University, Raleigh, NC 27695-7534, e-mail: mueller@cs.ncsu.edu, phone: +1.919.515.7889

Preliminary versions of this paper, at different stages, appeared in the internal IBM Pa=c^2 Conference [Anantaraman et al. 2004] and the ACM SIGPLAN Conference on Language, Compiler, and Tool Support for Embedded Systems, 2005 [Zhu and Mueller 2005].

off between system performance and energy consumption. The philosophy behind those mechanisms is to keep processors or I/O devices running without heavy-weight shutdown operations. Processors or I/O devices dynamically switch between low power states and high power states with little overhead. The degradation of quality of service is usually transparent to users so that low power consumption is achieved while providing continuous system services and short interrupt response times.

Dynamic voltage scaling (DVS) is one of the fine-grained power saving mechanisms proposed in recent years for microprocessors. It is based on the relationship of

$$P \propto V^2 \times f \tag{1}$$

where P denotes the power dissipation of CMOS-based chips, V is the supply voltage and f the clock frequency the processor [Chandrakasan et al. 1992]. Because of the quadratic relationship between the supply voltage and the power consumption, lowering the voltage results in significant power reductions for CMOS circuits. Since multiple voltage and frequency levels are supported in contemporary microprocessors and supporting chipsets, DVS can leverage power consumption by dynamically scaling the processor supply voltage up or down depending on the computational demand of the workload. For any concrete processor, a higher clock frequency mandates a higher supply voltage. Hence, DVS controls both voltage and frequency simultaneously. In the following, we use the term dynamic voltage scaling to refer to combined voltage and frequency switching.

DVS algorithms for general-purpose systems often use various heuristics to reduce processor voltage or frequency according to observed system workload [Weiser et al. 1994; Pering et al. 1995; Govil et al. 1995]. DVS for hard real-time systems, in contrast, requires more subtle control to not only consider the voltage and frequency but also the timing constraints of a task set. Each task in a real-time system has a certain deadline associated with it. Since the execution of the task has to finish before that deadline, reducing the processor frequency, *i.e.*, slowing down the speed of execution, may not always feasible in a real-time context. Different schemes have been proposed in previous work to integrate DVS algorithms into real-time systems [Pillai and Shin 2001; Aydin et al. 2001; Grunwald et al. 2000]. The objective is to derive a safe clock frequency considering timing constraints so that just enough processing capability is provided to complete a given task before its deadline. Th primary objective is to meet deadlines, *i.e.*, to guarantee schedulability of hard real-time tasks while the secondary objective is to safe as much energy as possible under the timing constraints.

In this work, we focus on DVS algorithms for hard real-time systems with earliest-deadline-first (EDF) scheduling. Hard real-time systems are mission-critical control systems where a single deadline miss may have catastrophic effects, *i.e.*, the highest mandate is to meet all deadlines without compromise. We develop several power-aware schemes for feedback DVS algorithms under EDF scheduling. The feedback DVS algorithms adjust real-time tasks' speed dynamically according to feedback of past execution workload characteristics and guarantee that deadlines can still be met at the adjusted speeds.

Our previous work presented a *simulation framework* for DVS scheduling algorithms and reported results from simulations with different task sets [Dudani et al. 2002; Zhu and Mueller 2004]. In this paper, we refine this framework and further develop feedback schemes considering practical design and implementation issues *on a real embedded platform*. Specifically, our feedback DVS algorithm targets real-time tasks with dynamic workloads. It aggressively reclaims unused or under-utilized CPU resources according to

the dynamic variation of task workloads. Different feedback schemes are compared with each other and the actual energy consumption is measured on an embedded platform. Although the relationship of $P \propto V^2 \times f$ is useful for simulation, the actual power savings also depend on other architectural and operating system factors. DVS algorithm overhead, such as the voltage and frequency switching overhead, and the DVS scheduling overhead, are measured and evaluated quantitatively in an actual embedded environment.

Traditionally, the processor voltage and frequency switching operation, although much of lighter weight than the overheads associated with sleep modes, still takes a small amount of time to complete. All other tasks have to be stooped during that interval. A unique DVS feature supported by our embedded platform is asynchronous switching, which allows applications to continue execution even during the frequency and voltage transitions. We study this asynchronous switching feature with our feedback DVS algorithm and compare it with synchronous switching operations.

Another factor that needs to be considered is the DVS scheduling overhead. Since DVS schemes are integrated into the operating system scheduler, the scheduler itself may execute at different processor speeds, which impacts timing properties of the entire system and requires accurate modeling. We examine all these issues by implementing our feedback DVS algorithm as well as several other DVS algorithms on an IBM PowerPC 405LP embedded board, which was specially modified for power management research. We show the strength of our feedback DVS algorithms by comparing their energy consumption with other DVS algorithms on the embedded platform.

This paper is organized as follows. Section 2 gives a brief introduction of the DVS scheduling framework and task model. Section 3 discusses our DVS algorithm and two feedback mechanisms proposed for the practical environment. Detailed experimental results are presented in Section 4. Section 5 discusses some of the related work. Conclusions are given in Section 6.

2. TASK MODEL

We use a periodic, fully preemptive and independent task model in our framework. Each task T_i is defined by a triple (P_i, D_i, C_i), where P_i is the period of T_i, D_i is the relative deadline of T_i, and C_i is the worst-case execution time (WCET) of T_i, measured at the maximal processor frequency. We always assume D_i=P_i in our model. The periodically released instances of a task are called jobs. T_{ij} is used to denote the j^{th} job of task T_i. Its release time is $P_i * (j-1)$ and its deadline is $P_i * j$. We use c_{ij} to represent the actual execution time of job T_{ij}. Different instances of a task T_i usually has different actual execution times, which are always bounded by that task's worst case execution time C_i. The hyperperiod H of the task set is the least common multiplier (LCM) among the tasks' periods.

With the above task model, we examine the DVS scheduling problem in hard real-time systems with the earliest deadline first (EDF) policy. The EDF scheduler always assigns the highest priority to the task with the earliest deadline and schedules tasks preemptively. EDF is especially attractive to DVS algorithms because of the dynamic priority for each task, which allows the DVS algorithm to reclaim un-used slack as much as possible. We integrate our feedback DVS algorithm into the operating system scheduler. Feedback DVS algorithm incrementally adjusts system behavior according to feedback information from previous workload characteristics to achieve a low energy consumption for the task set.

In order to assess the algorithms for their suitability and energy saving performance, we regard the entire system as consisting of the following two components: (1) an EDF scheduler, (2) a DVS scheduler. The EDF scheduler always chooses the highest priority task in the task set while The DVS scheduler assigns a particular processor voltage and frequency setting to that task. These two components are independent of each other so that the EDF scheduler is capable of working with different DVS algorithms.

In the following, we describe in detail the feedback DVS scheduler and several feedback schemes used in the framework.

3. FEEDBACK DVS ALGORITHM

The feedback DVS algorithm is based on the task-splitting scheme, as depicted in Figure 1. Instead of running the entire task at a uniform speed, each task's worst case execution

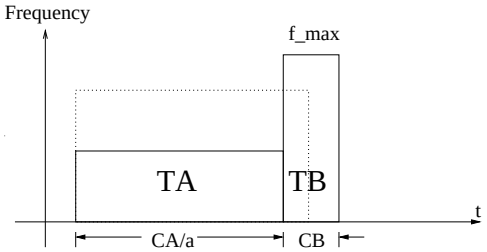

Fig. 1. Task Splitting

image is divided into two sub-tasks: T_A and T_B. These two subtasks are allowed to execute at different frequency and voltage levels. T_B always executes at the maximum frequency level f_{max}, which allows T_A capable to execute at a lower frequency level than it could without task splitting. We expect that the task can finish its actual execution within T_A while reserving enough time in T_B to meet the deadline if it requires its full WCET. An optimal case is achieved if the entire task completes within the T_A portion. With this scheme, we can safely scale the frequency of T_A using available slack while T_B executes at maximum frequency following a last-chance approach [Chetto and Chetto 1989]. Let C_A and C_B be the worst-case execution time of the two subtasks($C_B = WCET - C_A$) under the maximal frequency, and s_k be the slack available when the task is scheduled. α is the speed scaling factor which can be derived from $\alpha = \frac{C_A}{C_A + s_k}$. By splitting each task into at most two subtasks, we incur at most one speed change for every task, and keep the impact of voltage and frequency switching overhead to a minimum.

In order to let a task complete within its T_A portion most of the time to remove the high voltage and frequency portion of T_B, we need to let C_A approximate the task's actual execution time. In real-time applications, the actual execution time of a task often experiences fluctuations over different intervals. Different instances of a task also present variant execution time behavior. We use feedback control to capture the dynamic workload behavior, which is one of the fundamental mechanisms for dynamic systems to achieve equilibrium. In a feedback system, some variables, *i.e.*, controlled variables, are monitored and measured by the feedback controller and compared to their desired values, the so-called set points. The differences (errors) between the controlled variables and the set points are

fed back to the controller for further actions. Corresponding system states are usually adjusted according to the differences to let the system variables approximate the set points as closely as possible. Our feedback DVS algorithm assigns C_A according to the feedback information collected from the execution time of previous instances. As long as the actual execution time of a task is less than or equal to C_A, the entire task can therefore execute at a low frequency and voltage level. If, on the contrary, a task's actual execution time is longer than C_A, the additional task (C_B) runs at the maximum frequency to meet the deadline requirements of the real-time task. The algorithm still keeps the total system utilization below 100%, even when reducing processor frequency and voltage, which guarantees the schedulability of hard real-time task sets.

The feedback scheme plays an important role in our DVS algorithm. In the following, we propose several feedback schemes that were refined during the implementation of our feedback DVS algorithm on the IBM 405LP embedded board. Practical design and implementation issues are considered here to exploit different opportunities of energy saving potentials.

3.1 Simple Feedback

A periodic real-time workload may exhibit a relatively stable behavior during a certain interval of time. Thus, the actual execution time of different jobs remains nearly constant or only varies within a very small range. For such workloads, we use a very simple feedback mechanism by computing the moving average of previous jobs' actual execution times and feed it back to the DVS scheduler. We try to avoid the overhead of more complicated feedback mechanisms, such as the PID-feedback controller proposed in the next section, because the moving average values usually provide sufficiently accurate predictions in this case. The quantitative comparison of the overhead between our PID-feedback DVS algorithm and several other DVS algorithms also shows that a complicated feedback DVS scheme can degrade the energy saving potential to some extent, as discussed in the context of Table III.

We choose the value of C_A as the controlled variable for our simple feedback mechanism. Each job T_{ij}'s actual execution time c_{ij} is chosen as the set point. C_A is assigned to be 50% WCET for the first job of each task, which means half of the job's execution is budgeted at a low frequency, and half of it (C_B) is reserved at the maximum frequency. This is only a heuristic setting we have chosen for the initial job. The C_A value of following jobs is determined by the feedback scheme. The maximum frequency setting for C_B guarantees the deadline requirements of real-time tasks, even if the worst-case execution time is exhibited. Each time a job completes, its actual execution time is fed back and aggregated to anticipate the next job's C_A using previous jobs' moving average. Let C_{Aij} denote the C_A value for T_{ij}. The $(j+1)^{th}$ job of the task is assigned a C_A value according to:

$$C_{Ai(j+1)} = (C_{Aij} \times N + c_{ij} - c_{i(j-N)})/N \quad (2)$$

where N is a constant representing the number of items used in the moving average calculation. Our experiments show significant energy savings for such a simple feedback mechanism with very low scheduling overhead as long as the workload's actual execution time exhibits a stable behavior for a certain period of time. When the workload's behavior keeps changing dynamically with highly fluctuating execution times, simple feedback does not necessarily minimize power consumption. In those cases, a more sophisticated feedback mechanism is required, as detailed in the next section.

3.2 Multi-Input PID Feedback

The simple feedback control described in the previous section follows a proportional adjustment relative to average execution times. In practice, real-time embedded systems, such as audio and video playback or image processing systems, often experience fluctuating execution times among multiple task instances. The fluctuations may result in tendencies leading to higher processing demands up to some point and receding demands after this peak point. In order to devise a DVS algorithm adaptive to such a dynamic environment, we design a more sophisticated feedback scheme presented as a multiple-input PID feedback control system. PID-feedback control is a continuous feedback controller capable of providing sophisticated control response. The controlled variable can usually reach its set point and stabilize within a short period. A PID controller consists of three different elements, namely, proportional control, integral control, and derivative control. Proportional control influences the speed of the system adapting to errors, which is defined as the difference between the controlled variable and the set point, denoted by a pure proportional gain item. Integral control is used to adjust the accuracy of the system through the introduction of an integrator on past error histories. Derivative control usually increases the stability of the system through the introduction of a derivative of the errors.

For every task T_i in the system, its C_A value is chosen as the controlled variable while its actual execution time c_{ij} is chosen as the set point. The system error is defined as the difference between the controlled variable and the set point, *i.e.*,

$$\epsilon_{ij} = c_{ij} - C_{Aij}. \tag{3}$$

The error is measured periodically by the controller. Its output is fed back to the feedback-DVS scheduler to adjust the value for C_A. For n tasks in the task set, there are altogether n feedback inputs (ϵ_{ij}, i=1...n) and n system outputs (C_{Ai}, i=1...n). For each task T_i, let C_{Aij} be the estimated C_A value for its j^{th} job. The following discrete PID control formula is used in our feedback-DVS scheduler:

$$\begin{aligned} \Delta C_{Aij} &= K_p * \epsilon_{ij} + \tfrac{1}{K_i} \textstyle\sum_{IW} \epsilon_{ij} + K_d \tfrac{\epsilon_{ij} - \epsilon_i(t-DW)}{DW} \\ C_{Ai(j+1)} &= C_{Aij} + \Delta C_{Aij} \end{aligned} \tag{4}$$

where K_p, K_i and D_d are proportional, integral, and derivative parameters, respectively. ϵ_{ij} is the monitored error. The output ΔC_{Aij} is fed back to the scheduler and is used to regulate the next anticipated value for C_{Ai}. IW and DW are tunable window sizes such that only the errors from the last IW (DW) task jobs will be considered in the integral (derivative) term. We use $DW = 1$ to limit the history, which ensures that multiple feedback corrections do not affect one another. The three control parameters K_p, K_i and K_d adjust the control response amplitude and dynamic behavior with great versatility. The process of tuning the control parameters is a compromise among different system performance metrics. For example, the system may be tuned to have either a stable but slow control response, or an instable but dynamic control response. What is preferred in our system is a sufficiently rapid and stable control output during the entire scheduling process. We chose to tune those PID parameters by trial and error in our experiments.

This multi-input model achieved significant energy savings in our previous simulation experiments [Zhu and Mueller 2004], but it also exhibited some drawbacks when we implemented it on real embedded platforms. The multi-input control structure increases the total memory requirements of the system, since the DVS scheduler needs to create an indi-

vidual feedback controller for every task in the task set. Each feedback controller maintains a queue structure in order to store the execution time history of previous jobs, which requires additional memory spaces proportional to the length of the queue as well as the total number of tasks. Such per-task memory requirements limit the maximal number of tasks an embedded system can sustain. Furthermore, the multi-input model manipulates multiple inputs and multiple outputs simultaneously, which increases the complexity of the scheduler design and implementation. Given the difficulty of precisely characterizing the behavior of a control system, it also adds complexity to the theoretical analysis of the system.

In order to address these drawbacks, we transform the above multi-input control problem into a single-input control model in the following.

3.3 Single-Input PID Feedback

Instead of using $C_{Ai}(i = 1...n)$ as the controlled variable for each task T_i and creating *n* different feedback controller for *n* different tasks, we now define a single variable r as the controlled variable for the entire system as:

$$r_j = \frac{1}{n}\sum_{i=1}^{n}\frac{C_{Aij} - c_{ij}}{c_{ij}} \tag{5}$$

where j is the index of the latest job of task T_i before the sampling point. r_j describes the average difference between tasks' actual execution times and their corresponding C_A values. Our objective is to make r approximate 0 (*i.e.*, the set point). The system error becomes

$$\epsilon(r_j) = r_j - 0. \tag{6}$$

where $\epsilon(r_j)$ reflects the error of the entire task set and is not a function of a particular task T_i any more. $\epsilon(r_j)$ is further fed back to the PID scheduler to regulate the controlled variable r. The PID feedback controller is now defined as:

$$\begin{aligned} \Delta r_j &= K_p\epsilon(r_j) + \tfrac{1}{K_i}\textstyle\sum_{IW}\epsilon(r_j) + K_d\tfrac{\epsilon(r_j)-\epsilon(r_{j-DW})}{DW} \\ r_{j+1} &= r_j + \Delta r_j \end{aligned} \tag{7}$$

where K_p,K_i and K_d are the PID parameters. IW and DW are the integral and derivative window sizes.

When job T_{ij} completes, we adjust the C_A value for $T_{i(j+1)}$ by $C_{Ai(j+1)} = r_j c_{ij} + c_{ij}$, which is used by the DVS scheduler to calculate the scaling factor α and to determine a processor frequency and voltage for the next job. Such a single controller mechanism is easy to implement because one feedback controller suffices for the entire system, which reduces the complexity and overhead of the feedback DVS algorithm. It reduces the memory requirement of the system since only one global feedback queue needs to be created, instead of n different queues for n different tasks in the multi-input feedback scheme. The stability analysis of this control model is omitted here as it does not deviate from the standard proof for PID control systems [Franklin et al. 2002].

4. EXPERIMENTAL EVALUATION

In order to evaluate the energy saving potential of our algorithm in an actual system as opposed to a simulation environment, we implemented our feedback-DVS algorithm as well as several other DVS algorithms, namely static DVS, cycle-conserving DVS, look-ahead-1/2 DVS (all by Pillai and Shin [Pillai and Shin 2001]), DR-OTE and AGR-2 (by

Aydin *et al.* [Aydin et al. 2001]). Look-ahead-1 and look-ahead-2 are the original and a modified version of the look-ahead DVS algorithm in [Pillai and Shin 2001], respectively. Look-ahead-1 updates each task's absolute deadline immediately when a task instance completes. Look-ahead-2 delays such updates till the next task instance is released, which results in additional energy savings. AGR-2 follows the most aggressive scheme with an aggressiveness parameter k of 0.9. In these experiments, we use simple feedback on constant workloads and single-input PID feedback for dynamic fluctuating workloads, if not stated explicitly. We compared the energy consumption as well as DVS overhead of different algorithms. We also wanted to determine if the lower frequencies and voltages chosen by our feedback scheme outweigh the higher computational overhead required to make scheduling decisions.

4.1 Platform and Methodology

The embedded platform used in our experiment is a PowerPC 405LP embedded board running on a diskless MontaVista Embedded Linux variant, which is based on the 2.4.21 stock kernel but has been patched to support DVS on the PPC 405LP. This board provides the hardware support required for DVS and allows software to scale voltage and frequency via user-defined operation points ranging from a high end of 266 MHz at 1.8V to a low end of 33 MHz at 1V [Nowka et al. 2003; Brock and Rajamani 2003; IBM and Software]. The board has also been modified for 50% reduced capacitance, which allows DVS switches to occur more rapidly, *i.e.*, switches are bounded by at most a 200-microseconds duration from 1V to 1.8V. The DVS algorithms (static, cycle-conserving, look-ahead [Pillai and Shin 2001] and our feedback-DVS) were exposed to the DVS capabilities of the 405LP board. The scheduling algorithms can choose any frequency/voltage pair from the set depicted in Table I.

Table I. Valid Frequency/Voltage Pairs

Setting	0	1	2	3	4
CPU freq. (MHz)	33	44	66	133	266
bus freq. (MHz)	33	44	66	133	133
CPU voltage (Volts)	1.0	1.0	1.1	1.3	1.7

This set of pairs was constrained by a need to have a common phase lock loop (PLL) multiplier of 16 relative to the 33MHz base clock, and a divider of two or any multiple of 4. Changing the multiplier incurs additional overhead for switching, which we wanted to eliminate in this study. A dynamic power management (DPM) facility [Brock and Rajamani 2003] is developed as an enhancement to the Linux kernel to support DVS features. DPM *operating point* defines stable frequency/voltage pairs (as well as related system parameters), which we experimentally determined.

In order to assess power consumption, we need to monitor processor core voltage and current at a high rate. Hence, we used a high-frequency analog data acquisition board to gather data for (a) the processor core voltage and (b) the processor current. The latter was measured as a voltage level over a resistor with a 1V drop per 360mA. Power consumption was computed by multiplying the CPU voltage with its current. The data acquisition board allowed us to experiment with longer-running applications to assess the energy consump-

tion of the processor, which is the integration of power over time. We also employed an oscilloscope for visualizing the voltages and currents with high precision in readings.

We implemented an EDF scheduler as a user-level thread library under Linux on the 405LP board. A user-level library was chosen over a kernel-level solution because of the simplicity of its design and the fact that the operating system background activity is minimal on the embedded board infrastructure. Different DVS scheduling schemes were integrated into the EDF scheduler as independent modules.

4.2 Synchronous vs. Asynchronous Switch

We first assessed the overhead of different DVS techniques supported by the test board and the dynamic power management extensions of the operating system.

A unique DVS feature supported by the IBM PPC 405LP embedded board is that frequency switching can be done either synchronously or asynchronously. Synchronous switching is the traditional approach for processor frequency/voltage transitions, where applications have to stop execution during the transitional interval. Asynchronous switching, on the contrary, allows applications to continue execution during the frequency/voltage transitions. Figure 2 depicts the changes in current (lower curve) and voltage (upper curve) of the PPC 405LP processor core during an asynchronous switch.

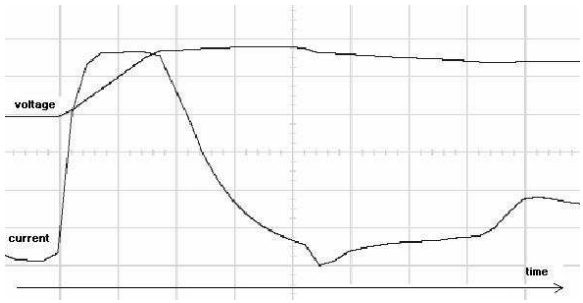

Fig. 2. Current and Voltage Transition During Asynchronous Frequency Switching

This unique feature of asynchronous switching is achieved by a system call that, when switching to a higher voltage/frequency, first reprograms the voltage to ramp up towards the maximum as fast as possible (the 30 degree voltage ramp on the upper curve of Figure 2). Meanwhile, the time to reach a voltage level at least as high as required by the new frequency is estimated. A high-resolution timer is programmed to interrupt when this duration expires, prior to which the application can still continue execution. Once the timer interrupt triggers its handler (at the peak after the 30 degree ramp on the upper curve), the power management unit is reprogrammed to settle at the target voltage level, and the new processor frequency is activated before returning from the handler. The voltage then settles (in case it overshot) in a controlled manner to the new operating point. The current also settles in a controlled manner depending on processing activity.

Table II reports the overhead for synchronous and asynchronous switching in a time range bounded by two extremes: (a) Switching between adjacent frequency/voltage levels and (b) switching between the lowest and highest frequency/voltage levels. Furthermore, the overhead of the subsequent signal handler associated with each asynchronous switch

Table II. Frequency/Voltage Switch Overhead

sync. switch	async. switch	signal handler	syscall
117-162 μsec	8-20 μsec	0.07-0.6 μsec	3-8 μsec

is also measured for a range of the highest and the lowest processor frequencies. In order to make a comparison, the execution time of a system call getpid() is also measured. The results indicate that a synchronous DVS switch has about an order of a magnitude higher overhead than an asynchronous switch. In contrast, the asynchronous DVS switch is almost as efficient as a null system call. The timer interrupt handler triggered at each asynchronous switch has a negligibly small impact on the DVS switching operation. Overall, triggering an asynchronous DVS switch only has the cost of a light-weight system call.

4.3 DVS Scheduler Overhead

We compared the timing overhead of our feedback-DVS algorithm with several other dynamic DVS algorithms. We first measured the execution time of these DVS scheduling algorithms under different frequencies on the embedded board, as depicted in Table III. The overhead was obtained by measuring the amount of time when a task issues a yield() system call till another task was dispatched by the scheduler. The table shows that static DVS has the lowest overhead among the four while our PID-feedback DVS has the highest one. This is not surprising since static DVS uses a very simple strategy to select the frequency and voltage falling short in finding the best energy saving opportunities. Cycle-conserving DVS, look-ahead DVS and our PID-feedback DVS use more sophisticated and aggressive algorithms for lower energy consumption, albeit at higher overheads. The single-input feedback scheme and the multi-input feedback scheme have almost the same timing overhead at high frequencies, since they require constant time to update the feedback information. But the single-input scheme imposes slightly less overhead than the multi-input scheme scheme at low frequency cases.

Table III. Overhead of DVS-EDF Scheduler

	DVS scheduling overhead[μsec]					
CPU freq.	static	cc	la	feedback		
				simple	PID	
					SI	MI
33 MHz	217	487	2296	3207	3612	3633
44 MHz	170	366	1714	2433	2943	3012
66 MHz	100	232	1112	1568	1728	1739
133 MHz	52	120	546	725	801	796
266 MHz	36	76	229	413	472	477

4.4 Synchronous vs. Asynchronous DVS

We also assessed if our feedback-DVS algorithm, although incurring the largest overhead among the four, gives the best energy saving results in the real embedded environment. We measured the actual energy consumption of these DVS algorithms when executing three medium utilization task sets depicted in Table IV using both synchronous and asynchronous DVS switchings. As a baseline for comparison, we also implemented a naïve

DVS scheme where the maximum frequency is always chosen whenever a task is scheduled, and the minimum frequency is always chosen whenever the system is idle. Since we isolate power measurements at the CPU level, excluding the memory and I/O subsystems, we experimented with a synthetic task set that does not affect the measurements of DVS at the CPU level due to their off-chip usage of resources.

Table IV. Task Set, times in msec

	Task Set 1		Task Set 2		Task Set 3	
task	Period (P_i)	WCET (C_i)	Period (P_i)	WCET (C_i)	Period (P_i)	WCET (C_i)
1	2,400	400	600	80	90	12
2	2,400	600	320	120	48	18
3	1,200	200	400	40	60	6

The first task set in Table IV is harmonic, *i.e.*, all periods are integer multiples of the smallest period, which facilitates scheduling. This often allows scheduling algorithms to exhibit an extreme behavior, typically outperforming any other choice of periods. The second and third task sets are non-harmonic with longer and shorter periods, respectively. Actual execution times were half that of the WCET for each task for this experiment.

Table V depicts the energy consumption in a unit of mWatt-hours. The naïve DVS algorithm serves as a base of comparisons for each of the subsequent DVS algorithms. For task set one, static DVS reduces energy consumption by about 29% over the naïve scheme. Cycle-conserving DVS saves 47% energy. Look-ahead RT-DVS saves over 50%, and our feedback method saves about 54% energy compared to naïve DVS. This clearly shows the tremendous potential in energy savings for real-time scheduling. The savings

Table V. Energy [$mW - hrs$] consumption per RT-DVS algorithm

algorithm	naïve	static(saved)	cycle-cons.(saved)	look-ahead(saved)	**feedback (saved)**
Task Set 1					
synchronous	4.47	3.2 (**28.41%**)	2.38 (**46.61%**)	2.21 (**50.56%**)	2.04 (**54.21%**)
asynchronous	4.43	3.13 (**29.35%**)	2.327 (**47.51%**)	2.12 (**52.07%**)	2.00 (**54.70%**)
sync/async saved	0.89%	2.19%	2.51%	3.92%	1.95%
Task Set 2					
synchronous	0.544	0.5056 (**7.06%**)	0.4713 (**13.36%**)	0.424 (**22.06%**)	0.4089 (**24.83%**)
asynchronous	0.5276	0.5025 (**4.76%**)	0.4622 (**12.40%**)	0.4218 (**20.05%**)	0.4064 (**22.97%**)
sync/async saved	3.01%	0.61%	1.93%	0.52%	0.61%
Task Set 3					
synchronous	0.595	0.5616 (**5.61%**)	0.4799 (**19.34%**)	0.4043 (**32.05%**)	0.3708 (**37.68%**)
asynchronous	0.5802	0.5496 (**5.27%**)	0.4547 (**21.63%**)	0.3912 (**32.57%**)	0.3671 (**36.73%**)
sync/async saved	2.49%	2.14%	5.25%	3.24%	1.00%
Task Set 2 vs. Task Set 3					
change	9.07%	8.57%	-1.65%	-7.82%	-10.71%

of each algorithm are lower for task set two peaking at 23% in our feedback scheme. As mentioned before, task set one is harmonic, which typically results in the best scheduling

(and energy) results since execution is more predictable. Task set three lies in between the other two with peak savings of 37% for our feedback scheme. The results also demonstrate that the overhead for calculations inherent to scheduling algorithms is outweighed by the potential for energy savings. This is underlined by the increasing overhead in execution time for each of the scheduling algorithms (from left to right in Table V) while energy consumption decreases.

Another noteworthy result is the comparison between synchronous and asynchronous DVS switching depicted in the last row for each task set in Table V. For each of the scheduling algorithms, we see additional savings of 1-5% on asynchronous switch due to the ability to commence with a task's execution during frequency and voltage transitions. We also ran experiments with task sets that had an order of a magnitude smaller periods and execution times. Surprisingly, the synchronous *vs.* asynchronous savings remained approximately the same, even though DVS switches occur ten times as often. We believe that the periods and execution times used in our experiments are still large compared to the execution time of a synchronous or asynchronous switch. If we only save about 100 μsec at each frequency switch (as has been shown in Table II) but later on spend more then 10-100 msec in running a task, the benefit of the asynchronous DVS switching becomes insignificant. These results seem to indicate that the benefit of continuous execution during DVS switching, although not negligible, is secondary to trying to minimize the overhead of DVS scheduling itself.

We also compared task sets two and three in terms of their absolute energy readings, which is valid since they executed for the same amount of time (ten seconds), the same actual to worst-case execution time ration and the same utilization, albeit at seven times more context switches. This change is depicted in the last row of Table V for the asynchronous case. Not surprisingly, the energy with naïve DVS is about 9% higher for task set three than for set two due to the higher context switch overhead of the latter. Quite interestingly, this overhead turns into a reduction in energy as DVS schemes become more aggressive.

4.5 Impact of Different Workloads

We now examine the behavior of our DVS algorithm on different workloads in more detail. A suite of task sets with synthetic CPU workloads was created. Each task set contains three independent tasks with different periods whose worst-case execution time varies from 0.1 to 0.9 with an increment of 0.1. The actual execution time of a task is determined by timing the body of each task plus the scheduler overhead (see Table III) of the corresponding DVS algorithm under the lowest CPU frequency. We dynamically changed the number of instructions inside each task body among different invocations (jobs) to approximate the workload fluctuation behavior of actual real-time applications. The fluctuation of one task is independent of that in another and, due to different periods, intentionally out-of-sync. Altogether, four synthesized execution patterns were created, as shown in Figure 3.

In the first pattern, a task's actual execution time is always 50% WCET. In the second pattern, the actual execution time of a task drops exponentially from a peak value c_m to 50%WCET among its consecutive jobs, modeled as $c_i = 1/2^{(t-c_m)}$. The peak value c_m is randomly generated for each spike from a uniform distribution between 50% of WCET and 100% of WCET. This pattern simulates event-triggered activities that result in sudden, yet short-term computational demands due to complex inputs often observed in interrupt-driven systems. The third pattern is similar to the second one except that it drops more gradually, modeled as $c_i = c_m sin(t + \pi/2)$. This pattern simulates events resulting in

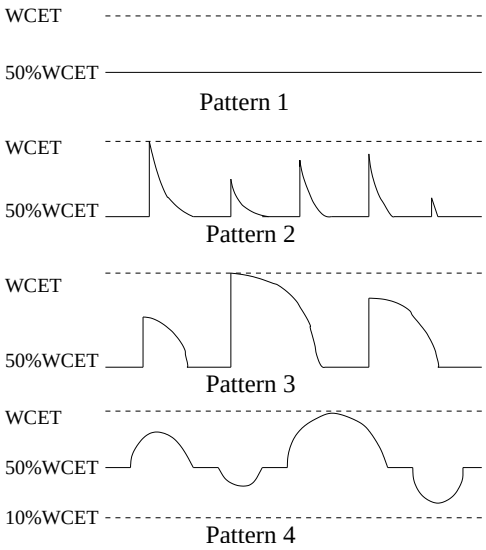

Fig. 3. Task Actual Execution Time Pattern

computational demands in a phase of subsequent complex inputs with a decaying tendency. In the fourth pattern, the actual execution time of a task increases and decreases gradually around 50% WCET with either a positive or negative amplitude, modeled as $c_i = c_m sin(t)$ and $c_i = -c_m sin(t)$. This pattern represents periodically fluctuating activities with gradually increasing and decreasing computational needs around peaks. We used simple feedback on pattern 1 because of its nearly constant execution time pattern among different jobs. The number of items to compute the moving average was set as $N = 10$. PID-feedback was used on patterns 2, 3, and 4 to exploit fluctuating execution time characteristics. The PID parameters were chosen by manual tuning as $K_p = 0.9$, $K_i = 0.08$, $K_d = 0.1$. The derivative and integral window size were 1 and 10, respectively. We used asynchronous switching in the experiment.

Figures 4 - 7 present the energy consumption of our feedback-DVS algorithm, as well as four other dynamic DVS algorithms under the four dynamic execution time patterns. Each task set contains three tasks. For pattern 1, we compare our simple feedback scheme with the multi-input feedback scheme. For dynamic pattern 2, 3 and 4, we compared our single-input feedback scheme with the multi-input feedback scheme. All energy values are normalized to the naïve DVS results under corresponding task set configurations. DR-OTE and AGR-2 dynamically reclaim unused slack up to the next arrival time of any task instance, thereby saving about 50% extra energy than naïve DVS. AGR-2 is not as good as Look-ahead-1/2 DVS in pattern 1 and 3, but beats Look-ahead-1/2 in pattern 2 and 4 for some cases. Look-ahead-1/2 is aggressive in frequency scaling, but it has to overcome the fact that the frequency is occasionally lowered too aggressively so that it has to be subsequently raised to a high level. We avoid such behavior in our algorithm *via* feedback.

In Figure 4, our simple feedback scheme performs almost as well as the multi-input feedback scheme. The difference of normalized energy between our algorithm and others ranges from an additional 5% to 15% energy savings over the best scheme published previously. Considering the low overhead of the simple feedback scheme, it is a good choice

for tasks whose execution time does not vary over multiple instances.

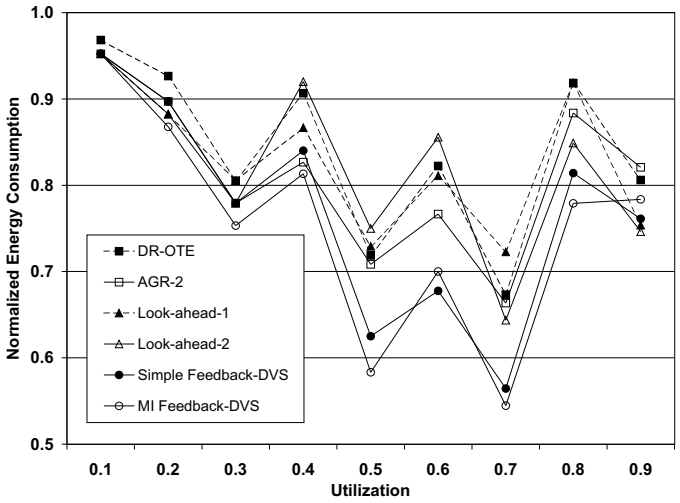

Fig. 4. Energy Consumption for Set of 3 Tasks, Pattern 1

In Figures 5, 6 and 7, our single-input and multi-input feedback schemes save an additional 5%-18% energy over other schemes due to the algorithm's self-adaptation to a job's actual execution time. Single-input feedback performs slightly worse than the multi-input feedback scheme, because its modeling method is not as precise as the multi-input feedback scheme . But the energy consumption differs by about 4% for all cases between these two schemes. There are even cases, *e.g.*, at 0.6 utilization in Figure 6, where single-input feedback outperforms the multi-input feedback. In extremely low or extremely high task utilization cases, our feedback-DVS, Look-ahead DVS and AGR algorithm result in comparable energy consumption. In these cases, tasks either have enough slack to always run at the minimum speed, or they do not have slack at all preventing them to lower their speed. This results in virtually identical frequency choices irrespective of the DVS algorithm.

To better assess the scalability of our feedback-DVS algorithm, we further ran two experiments. One experiment increases the number of tasks in the task set from 3 to 30, as shown in Figure 8. AGR-2 benefits from such a smaller task granularity in 30-task sets and outperforms Look-ahead-1 and Look-ahead-2 in some utilization cases. Small task granularity also reduces the gap between our feedback-DVS algorithms and the other algorithms. But we still save around 3% to 8% additional energy than others.

The second experiment fixes the execution time pattern of the task set, while varying the baseline (the average execution time) of different task instances, as shown in Figure 9. The average execution time in Figure 9 is set as 75%WCET, 50%WCET and 30%WCET, respectively. All energy values are normalized to the naïve DVS values. We see from these figures that our single-input feedback scheme scales equally well for loose (0.3WCET case) and tight (0.75WCET case) actual execution-times. In all three cases, 14% to 24% additional energy is saved over look-ahead-2 DVS. The feedback schemes show larger improvements for median execution times than the loose or tight ones. In this range, there is enough slack to distinguish itself from the other algorithms. When comparing our feedback

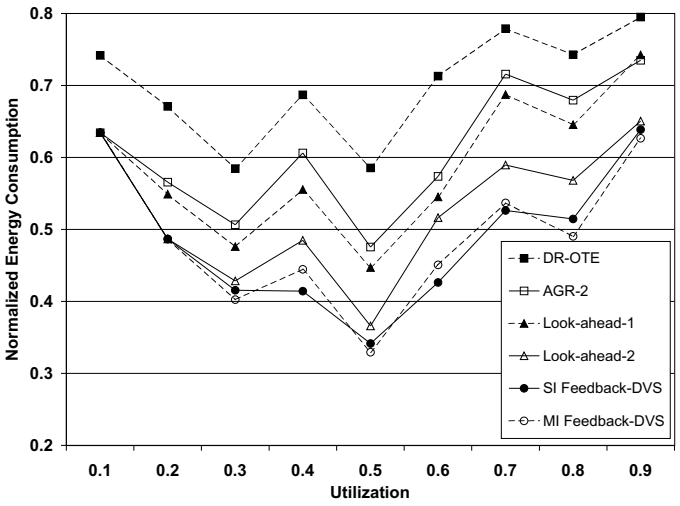

Fig. 5. Energy Consumption for Set of 3 Tasks, Pattern 2

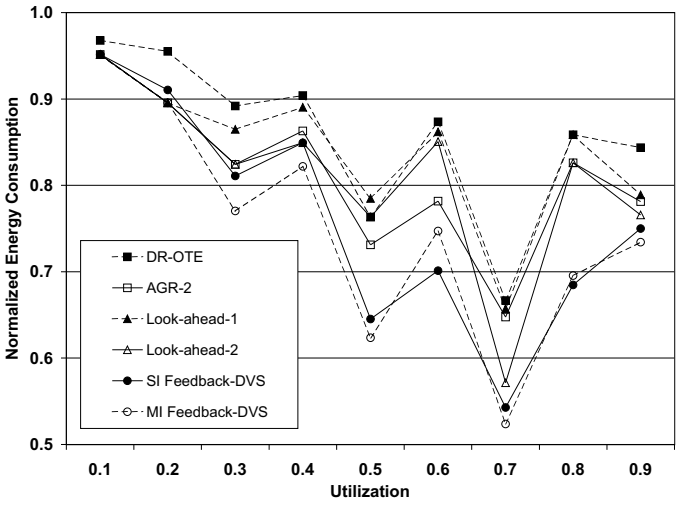

Fig. 6. Energy Consumption for Set of 3 Tasks, Pattern 3

algorithm with the naïve DVS scheme, we observe even more significant energy savings. The largest saving is shown in Figure 8, where up to 70% additional savings are achieved by our algorithm over the naïve one at the 0.3 utilization case.

We also visualized voltage and current switches using an oscilloscope. Figures 10 and 11 depict the screen-shots of voltage and current obtained from the oscilloscope for the phase just after a simultaneous release of all tasks at the beginning of a hyperperiod. In Figure 10, every task has a loose WCET, which is two times of its actual execution time.

In Figure 11, a tight WCET equal to a task's actual execution time is used. Static DVS shows two levels of voltages (busy/idle time) whereas cycle-conserving DVS differentiates three levels on a dynamic base. Even lower voltage and current readings are given by look-

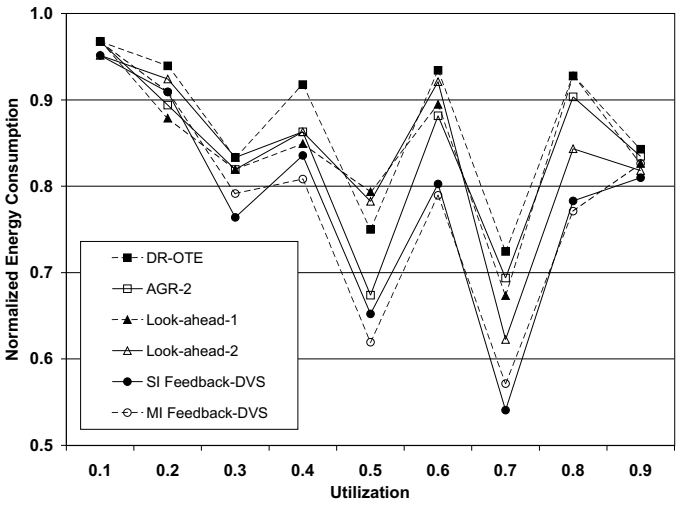

Fig. 7. Energy Consumption for Set of 3 Tasks, Pattern 4

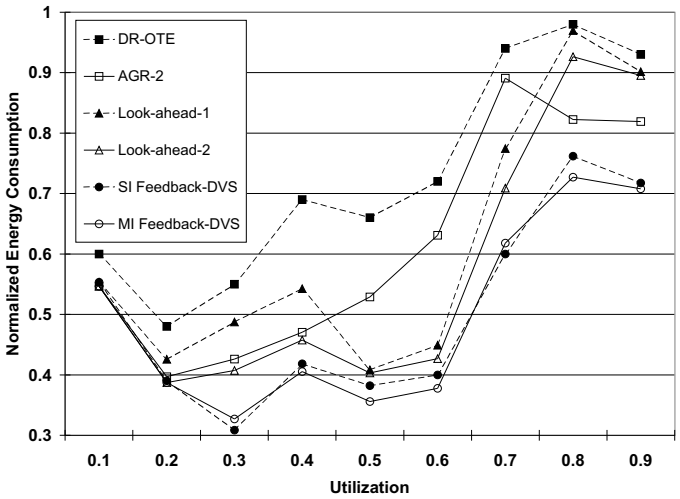

Fig. 8. Energy Consumption for Set of 30 Tasks, Pattern 2

ahead DVS, which not only distinguishes more levels but also exhibits much lower power levels during load. The lowest results were obtained by our feedback DVS, which defers execution even more aggressively than any of the other methods. However, our feedback scheme can only reduce power consumption occasionally as sufficient static or dynamic slack exists to be reclaimed. Dynamic slack is recovered in increasing orders by the latter three schemes.

Our feedback-DVS scheme obtains lower energy consumption than previous non-feedback approaches because the feedback controller lets most of the tasks complete within the T_A subtask without getting into the high-frequency T_B portion. In order to substantiate this claim, for the varying execution time patterns 2, 3 and 4, we measured the percentage

of jobs that get into the high frequency T_B subtask, as well as the percentage of energy consumed in the T_B portion. The results, as depicted in Figures 12, 13 and 14, show that the number of jobs that get into the high-frequency T_B portion is constrained to be less than 31% of the total number of released jobs. The amount of energy consumed in those T_B portions is even less, from 1% to 9%, compared to the total energy consumption. These results clearly show that the feedback scheme can approximate the optimal energy case by removing the T_B subtask during the actual task execution as much as possible.

Besides the execution time patterns listed in Figure 3, we also investigated task sets with random execution characteristics, *i.e.*, tasks' actual execution times are derived from a random uniform distribution. We performed this experiment in order to assess the behavior of our algorithm for task sets with highly fluctuating execution time patterns that cannot be predicted with PID feedback. Our feedback-DVS scheme still resulted in similar energy savings as other DVS algorithms. Random execution times do not give additional benefits to our algorithm because the algorithm cannot supply any useful history information to the feedback controller. This is a limitation of feedback schemes in general. Nonetheless, even in this worst case, our feedback-DVS algorithm behaves no worse than previous non-feedback-based DVS algorithms.

4.6 Comparison with Simulation Results

When we compare the energy saving results obtained from the IBM 405LP embedded board with our previous simulation results presented in [Zhu and Mueller 2004], we clearly

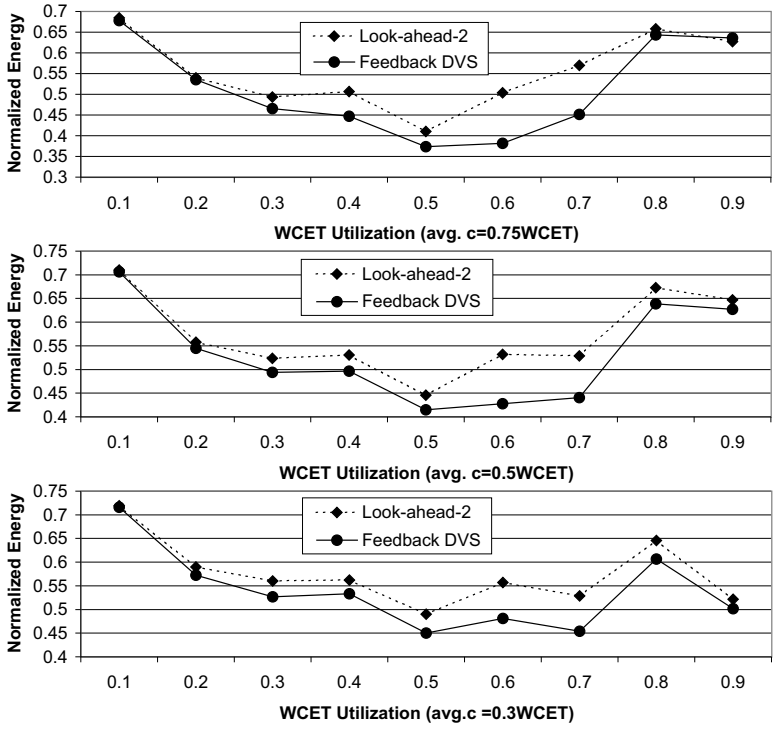

Fig. 9. Energy Consumption for Set of 3 Tasks, Pattern 4, with Varying Baseline

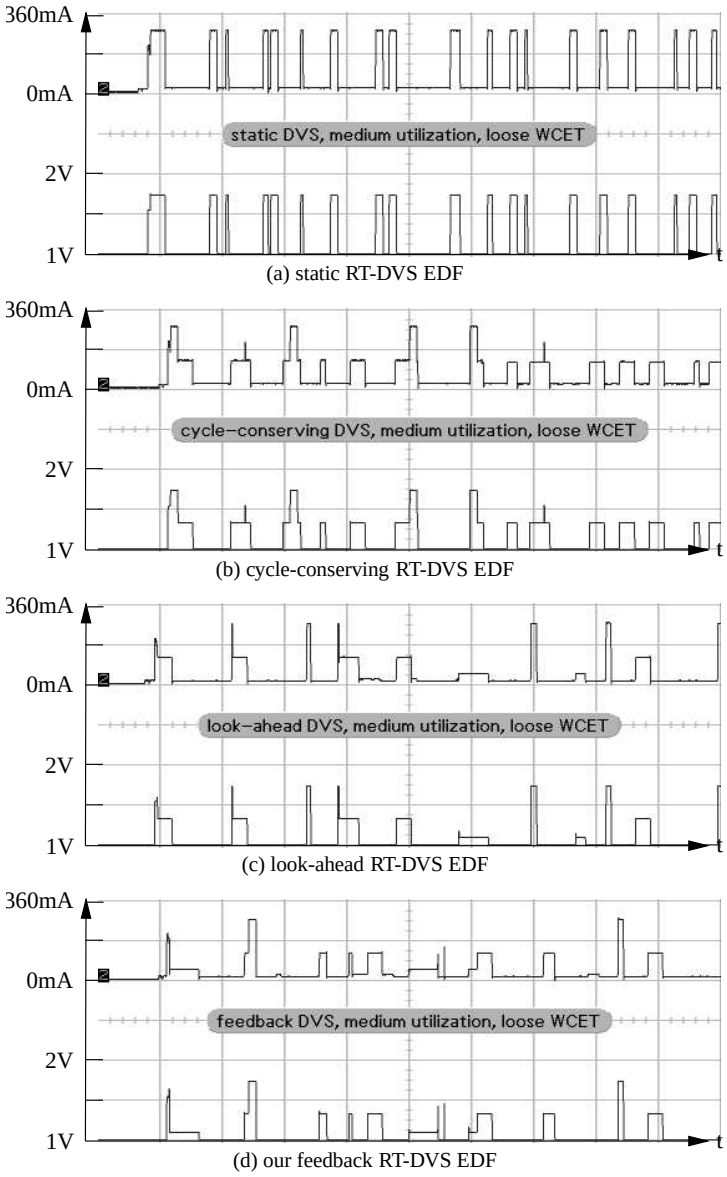

Fig. 10. Voltage/Current Oscilloscope Shot, Loose WCET= 2× ActualExecTime, U=0.5

see the advantage and disadvantage of simulation for power-aware studies. The advantage of simulation lies in its ease of implementation and predictability of performance trends. The energy consumption of different DVS algorithms shows a consistent trend under both simulation and the actual embedded platform. But the quantitative results differ. Our previous simulation results reported 5%-10% higher savings on average. For example, the best energy saving of our feedback-DVS over look-ahead DVS was reported as 29% in simulation while the best result we measured from the test board is around 24%. The differences

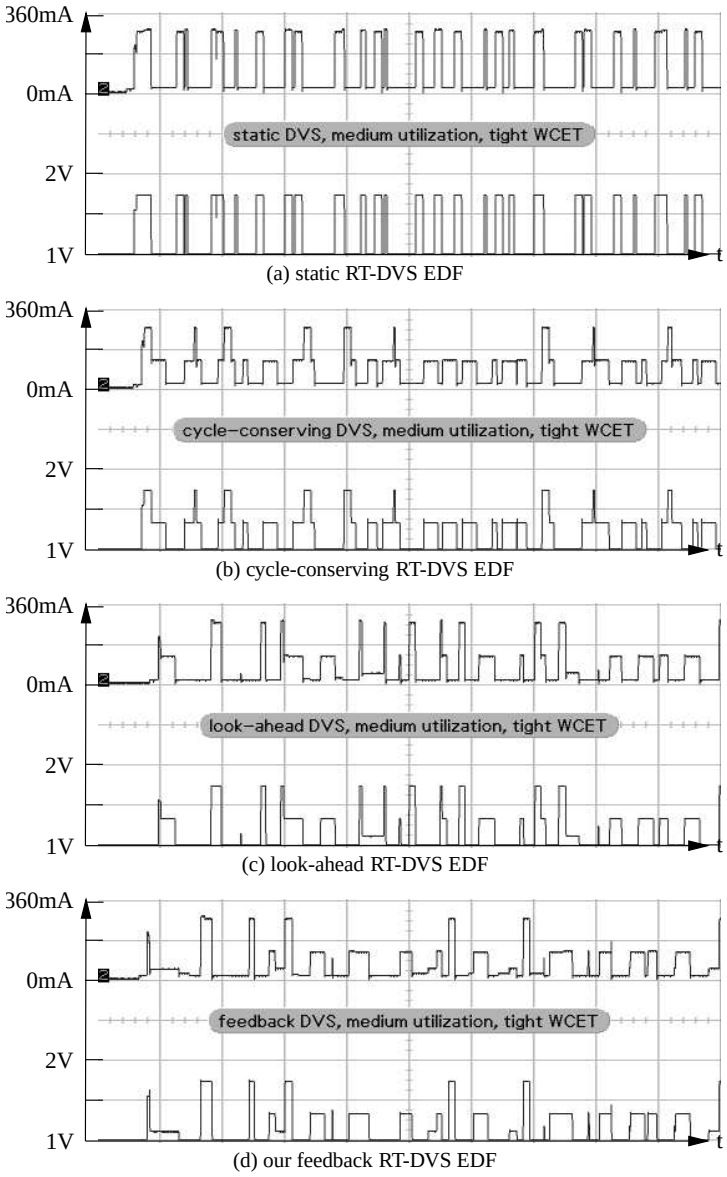

Fig. 11. Voltage/Current Oscilloscope Shot, Tight WCET= ActualExecTime, U=0.5

are likely due to discrepancies in the simulation model for actual power/energy consumption since it does not consider specific hardware details, such as fabrication sizes, number of transistors etc. [Jejurikar et al. 2004]. We also used different task set parameters, such as periods and execution times.[1] This is also the case when evaluating the overhead. Since

[1]Our objective was to allow a direct comparison with look-ahead DVS, which constrained us to four frequency levels at 25% intervals. While more frequency levels are support by our hardware platform, they do not match

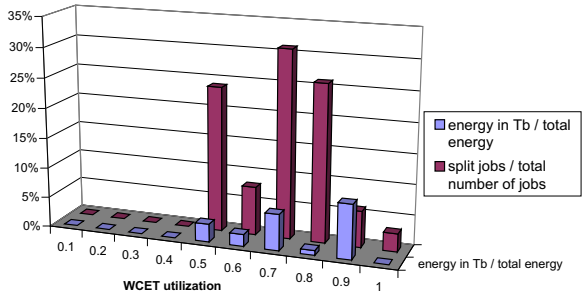

Fig. 12. Pattern 1, Percentage of subtask(energy) in T_B

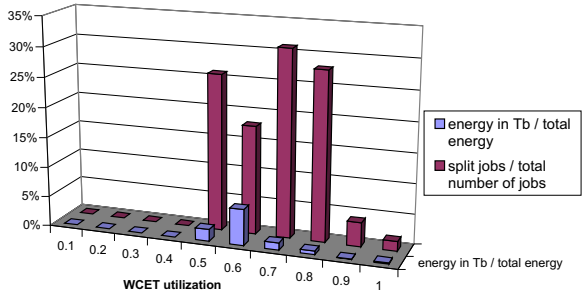

Fig. 13. Pattern 2, Percentage of subtask(energy) in T_B

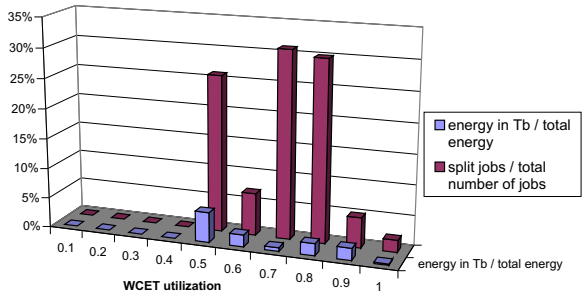

Fig. 14. Pattern 3, Percentage of subtask(energy) in T_B

the overhead of DVS algorithms was not included in our previous simulation experiment, we still observed 7%-10% energy savings over look-ahead DVS even at high utilization cases. But the actual energy measurements from the test board show only 3%-6% savings for these cases.

Overall, our experiments on the embedded platform quantitatively show the potential of our feedback-DVS algorithm. Using the feedback-based DVS scheme, processor frequencies are possible to be scaled even more aggressively than previous DVS algorithms.

the 25% intervals, *e.g.*, a 75% frequency level does not exist.

5. RELATED WORK

Dynamic voltage scaling for real-time systems has received considerable attention in recent years. Pillai and Shin present a suite of DVS algorithms integrated with hard real-time EDF and RM scheduling [Pillai and Shin 2001]. Processor speed for each task is adjusted dynamically while the schedulability of the system is still reserved. Look-ahead DVS is the most aggressive DVS scheme among the suite of algorithms proposed. Aydin *et al.* discuss a series of algorithms, which dynamically reclaim unused computation time of real-time tasks to reduce the processor speed [Aydin et al. 2004]. Energy-aware scheduling of hybrid workloads, including both periodic and aperiodic tasks, are further investigated by Aydin and Yang in [Aydin and Yang 2004]. Gruian analyzes dual-speed and multi-step stochastic intra-task DVS using a hybrid scheme with static and dynamic frequency scaling [Gruian 2001]. He observes that a dual-speed approach would be best but, short of knowledge about the actual execution time, promotes the stochastic approach whose data is derived from past execution traces. In contrast, our PID controller approximates the actual execution time closely so that the dual-speed approach becomes feasible. PID feedback is superior to a stochastic approach in that it can dynamically adjust to changing execution patterns. Also, our approach switches to the maximum frequency in the second part, not just a higher one. Furthermore, this switch only takes place when the predicted actual execution time is exceeded while Gruian's approach allowed earlier switches, which would result in additional switch overhead on average. Jejurikar and Gupta investigate static and dynamic slowdown factors for periodic tasks [Jejurikar and Gupta 2004b] and combine it with procrastination scheduling [Jejurikar and Gupta 2004c] and preemption threshold scheduling [Jejurikar and Gupta 2004a] for DVS. Several of these algorithms were compared in a unified simulation environment, SimDVS [Shin et al. 2002]. In contrast, we measure power consumption on a concrete micro-architecture for several EDF-based algorithms.

Feedback control for real-time scheduling was first investigated by Stankovic *et al.* [Stankovic et al. 1999]. Real-time system performance specifications are analyzed systematically through a control-theoretical methodology by Lu *et al.* [Lu et al. 2000]. A feedback-control real-time scheduling framework for unpredictable dynamic real-time systems is further proposed by Lu *et al.* where execution times diverge from their worst case [Lu et al. 2002]. Dynamic models of real-time systems are developed to identify different categories of real-time applications with different feedback control algorithms. Our work extends feedback to power-aware EDF scheduling.

Feedback control was also proposed for energy-aware computing in previous work. Varma *et al.*[Varma et al. 2003] present a feedback-control algorithm where the previous workload execution history is used to predict the future workload behavior by a discrete-time PID function. The combination of the proportional, integral and derivative part of the PID function provides good estimation across different applications insensitive of the change of their parameters. Lu *et al.*[Lu et al. 2002] describe a formal feedback-control algorithm combined with dynamic voltage/frequency scaling technologies for multimedia systems. Both continuous and discrete DVS settings are exploited in a scheme to reduce energy consumption while still guaranteeing real-time requirements. An adaptive set-point is used to achieve fast responses with a stable multimedia throughput. Poellabauer *et al.* [Poellabauer et al. 2005] apply a feedback loop on cache miss rates to make more reliable prediction of future task execution times. A general energy management scheme with

feedback control is proposed by Minerick *et al.* [Minerick et al. 2002]. Average energy usage is achieved by continuously adjusting the voltage/frequency of a processor to meet the energy consumption goal. A PI (proportional and integral) feedback controller is used to adapt the proper power setting based on previous energy consumptions without the prediction of future system workloads. While Varma, Lu and Poellabauer's work target software real-time systems and Minerick's work targets general purpose systems, our feedback DVS scheme focuses on hard real-time systems where timing constraints must not be violated.

6. CONCLUSION

This paper presents different feedback schemes for DVS algorithms considering practical hardware and software design and implementation issues. We evaluated our feedback DVS algorithms as well as several other real-time DVS algorithms on an IBM 405LP embedded platform. Real-time task sets with different varying workloads were assessed under different feedback schemes. We measured the actual energy consumption of our feedback DVS algorithm as well as several other real-time dynamic DVS algorithms on the embedded platform. Algorithm performance and its adaptability to dynamic workloads were evaluated. Asynchronous switching, a unique feature provided by the IBM 405LP embedded test board, and synchronous switching were also assessed with the real-time task sets. Our experiments show up to 5% additional energy savings with asynchronous switching, as opposed to traditional synchronous switching operations. The experimental results indicates a considerable potential for real-time DVS scheduling algorithms with up to 70% energy savings over a naïve DVS scheme. The strengths of our feedback DVS algorithms were also shown in experiments with a 24% peak energy savings over previous real-time DVS algorithms. These results differ in quantity from prior simulations, but they confirm the trends of savings observed under simulation. To the best of our knowledge, this is the first comparative study of real-time DVS algorithms on a concrete micro-architecture and the first evaluation of asynchronous DVS switching.

Acknowledgments

Ajay Dudani contributed to early work of the Feedback-DVS scheme [Dudani et al. 2002]. Aravindh Anantaraman, Ali Mahmoud and Ravi Venkatesan designed and implemented an early version of the DVS experimentation environment. Bishop Brock from IBM provided most valuable technical details for the PPC 405LP board, which was donated by IBM Research (Austin).

REFERENCES

ANANTARAMAN, A., MAHMOUD, A., VENKATESAN, R., ZHU, Y., AND MUELLER, F. 2004. Edf-dvs scheduling on the ibm embedded powerpc 405lp. In *Proceedings of the IBM Pa=c^2 Conference*.

AYDIN, H., MELHEM, R., MOSSE, D., AND MEJIA-ALVAREZ, P. 2001. Dynamic and agressive scheduling techniques for power-aware real-time systems. In *IEEE Real-Time Systems Symposium*.

AYDIN, H., MELHEM, R., MOSSE, D., AND MEJIA-ALVAREZ, P. 2004. Power-aware scheduling for periodic real-time tasks. *IEEE Trans. Comput. 53*, 5, 584–600.

AYDIN, H. AND YANG, Q. 2004. Energy-responsiveness tradeoffs for real-time systems with mixed workload. In *Proceedings of the 11th IEEE Real-Time and Embedded Technology and Applications Symposium*.

BROCK, B. AND RAJAMANI, K. 2003. Dynamic power management for embedded systems. In *IEEE International SOC Conference*.

CHANDRAKASAN, A., SHENG, S., AND BRODERSEN, R. W. April, 1992. Low-power cmos digital design. In *IEEE Journal of Solid-State Circuits, Vol. 27, pp. 473-484.*

CHETTO, H. AND CHETTO, M. 1989. Some results of the earliest deadline scheduling algorithm. *IEEE Transactions on Software Engineering 15*, 10 (Oct.), 1261–1269.

DUDANI, A., MUELLER, F., AND ZHU, Y. 2002. Energy-conserving feedback edf scheduling for embedded systems with real-time constraints. In *ACM SIGPLAN Joint Conference Languages, Compilers, and Tools for Embedded Systems (LCTES'02) and Software and Compilers for Embedded Systems (SCOPES'02)*. 213–222.

FRANKLIN, G., POWELL, J. D., AND EMAMI-NAEINI, A. 2002. *Feedback Control of Dynamic Systems*, 4 ed. Prentice-Hall.

GOVIL, K., CHAN, E., AND WASSERMAN, H. 1995. Comparing algorithms for dynamic speed-setting of a low-power cpu. In *1st Int'l Conference on Mobile Computing and Networking*.

GRUIAN, F. 2001. Hard real-time scheduling for low energy using stochastic data and dvs processors. In *Proceedings of the International Symposium on Low-Power Electronics and Design ISLPED'01*.

GRUNWALD, D., LEVIS, P., III, C. M., NEUFELD, M., AND FARKAS, K. 2000. Policies for dynamic clock scheduling. In *Symp. on Operating Systems Design and Implementation*.

IBM AND SOFTWARE, M. Dynamic power management for embedded systems. white paper.

JEJURIKAR, R. AND GUPTA, R. 2004a. Integrating preemption threshold scheduling and dynamic voltage scaling for energy efficient real-time systems. In *Proceedings of the 10th International Conference on Real-Time and Embedded Computing Systems and Applications (RTCSA '04)*.

JEJURIKAR, R. AND GUPTA, R. 2004b. Optimized slowdown in real-time task systems. In *Proceedings of the 16th Euromicro Conference on Real-Time Systems (ECRTS '04)*.

JEJURIKAR, R. AND GUPTA, R. 2004c. Procrastination scheduling in fixed priority real-time systems. In *Proceedings of the Language Compilers and Tools for Embedded Systems*.

JEJURIKAR, R., PEREIRA, C., AND GUPTA, R. 2004. Leakage aware dynamic voltage scaling for real-time embedded systems. In *Design Automation Conference*.

LU, C., STANKOVIC, J. A., ABDELZAHER, T. F., TAO, G., SON, S. H., AND MARLEY, M. 2000. Performance specifications and metrics for adaptive real-time systems. In *Proceedings of the IEEE Real-Time Systems Symposium*.

LU, C., STANKOVIC, J. A., TAO, G., AND SON, S. H. 2002. Feedback control real-time scheduling: Framework, modeling, and algorithms. *Real-Time Syst. 23*, 85–126.

LU, Z., HEIN, J., HUMPHREY, M., STAN, M., LACH, J., AND SKADRON, K. 2002. Control-theoretic dynamic frequency and voltage scaling for multimedia workloads. In *Conference on Compilers, Architecture and Synthesis for Embedded Systems*. 156–63.

MINERICK, R., FREEH, V. W., AND KOGGE, P. M. 2002. Dynamic power management using feedback. In *Proceedings of Workshop on Compilers and Operating Systems for Low Power*.

NOWKA, K., CARPENTER, G., AND BROCK, B. 2003. The design and application of the powerpc 405lp energy-efficient system on chip. *IBM Journal of Research and Development 47*, 5/6 (September/November).

PERING, T., BURD, T., AND BRODERSEN, R. 1995. The simulation of dynamic voltage scaling algorithms. In *Symp. on Low Power Electronics*.

PILLAI, P. AND SHIN, K. 2001. Real-time dynamic voltage scaling for low-power embedded operating systems. In *Symposium on Operating Systems Principles*.

POELLABAUER, C., SINGLETON, L., AND SCHWAN, K. 2005. Feedback-based dynamic frequency scaling for memory-bound real-time applications. In *Proceedings of the 11th IEEE Real-Time and Embedded Technology and Applications Symposium*.

SHIN, D., KIM, W., JEON, J., KIM, J., AND MIN, S. L. 2002. Simdvs: An integrated simulation environment for performance evaluation of dynamic voltage scaling algorithms. In *Workshop on Power-Aware Computer Systems*.

STANKOVIC, J. A., LU, C., SON, S. H., AND TAO, G. 1999. The case for feedback control real-time scheduling. In *Proceedings of the EuroMicro Conference on Real-Time Systems*.

VARMA, A., GANESH, B., SEN, M., CHOUDHURY, S. R., SRINIVASAN, L., AND BRUCE, J. 2003. A control-theoretic approach to dynamic voltage scheduling. In *Proceedings of the 2003 international conference on Compilers, architectures and synthesis for embedded systems*. ACM Press, 255–266.

WEISER, M., WELCH, B., DEMERS, A., AND SHENKER, S. 1994. Scheduling for reduced cpu energy. In *1st Symp. on Operating Systems Design and Implementation*.

ZHU, Y. AND MUELLER, F. 2004. Feedback edf scheduling exploiting dynamic voltage scaling. In *IEEE Real-Time Embedded Technology and Applications Symposium*. 84–93.

ZHU, Y. AND MUELLER, F. 2005. Feedback edf scheduling exploiting hardware-assisted asynchronous dynamic voltage scaling. In *ACM SIGPLAN Conference on Language, Compiler, and Tool Support for Embedded Systems*. 203–212.

Received August 2005; revised March 2006; accepted September 2006