Feedback EDF Scheduling Exploiting Hardware-Assisted Asynchronous Dynamic Voltage Scaling

Yifan Zhu Frank Mueller

Department of Computer Science/Center for Embedded Systems Research
North Carolina State University, Raleigh, NC 27695-7534
mueller@cs.ncsu.edu, phone: +1.919.515.7889, fax: +1.919.515.7925

Abstract

Recent processor support for dynamic frequency and voltage scaling (DVS) allows software to affect power consumption by varying execution frequency and supply voltage on the fly. However, processors generally enter a sleep state while transitioning between frequencies/voltages. In this paper, we examine the merits of hardware/software co-design for a feedback DVS algorithm and a novel processor capable of executing instructions during frequency/voltage transitions. We study several power-aware feedback schemes based on earliest-deadline-first (EDF) scheduling that adjust the system behavior dynamically for different workload characteristics. An infrastructure for investigating several hard real-time DVS schemes, including our feedback DVS algorithm, is implemented on an IBM PowerPC 405LP embedded board. Architecture and algorithm overhead is assessed for different DVS schemes. Measurements on the experimentation board provide a quantitative assessment of the potential of energy savings for DVS algorithms as opposed to prior simulation work that could only provide trends. Energy consumption, measured through a data acquisition board, indicates a considerable potential for real-time DVS scheduling algorithms to lower energy up to 64% over the naïve DVS scheme. Our feedback DVS algorithm saves at least as much and often considerably more energy than previous DVS algorithms with peak savings of an additional 24% energy reduction. 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.

LCTES'05, June 15–17, 2005, Chicago, Illinois, USA.

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

Keywords Real-Time Systems, Scheduling, Dynamic Voltage Scaling, Feedback Control

1. Introduction

Energy consumption has become a vital design constraint in embedded systems. The demand for efficient energy management is increasing in hand-held and embedded devices, where battery service life is usually critical to system performance. For many non-battery powered systems, energy consumption is also an important cost factor due to environment issues. CPU is one of the most power-consuming devices of a computer. In order to reduce the CPU energy consumption, dynamic voltage scaling (DVS) technology is widely supported in recent processor products for extending battery life. DVS dynamically scales the processor core voltage depending on the computational demand of the system. Reducing the supply voltage results in a lower transistor switching speed, which also allows a lower clock frequency. Assuming that voltage and frequency are linearly related, scaling voltage *and* frequency results in cubic reduction of power consumption ($P \propto V^2 \times f$) [4]. While useful for simulation, the formula ignores architectural details studied in this paper.

DVS algorithms have been intensively studied for both non real-time and real-time systems [20, 1, 13, 6, 8, 19, 24]. In the case of real-time systems, the DVS algorithm calculates a safe frequency that provides just enough processing resources to finish a given task before its deadline. The goal is to save the maximum possible amount of energy and still guarantee the schedulability of hard real-time systems where all tasks are required to meet their deadlines.

In this work, we develop several power-aware feedback schemes for our feedback DVS algorithm based on earliest-deadline-first (EDF) scheduling, which adjusts a real-time

system dynamically according to different workload characteristics. A feedback DVS framework has been presented and evaluated in simulation experiments in our previous work [5, 25]. We refine those algorithms in this paper and develop several feedback schemes considering practical design and implementation issues on a real embedded architecture. We focus on the performance of the DVS algorithm in an embedded environment where the overhead and the actual energy consumption can be measured quantitatively. The DVS-enhanced real-time scheduler may itself execute at several different CPU frequencies, which also requires accurate modeling of the entire system. 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. A unique DVS feature supported by the test board is that frequency switching can be synchronous or asynchronous, both of which we evaluate experimentally for different DVS algorithms. We show the strength of our feedback DVS algorithm by comparing its energy consumption with that of other DVS algorithms on the test board.

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. EDF Scheduling with DVS Support

In this work, we consider the inter-task DVS scheduling problem in hard real-time systems with the earliest deadline first (EDF) policy. In order to assess DVS algorithms for their suitability and energy saving performance, we regard the entire system as consisting of two components: (1) an EDF scheduler, and (2) a DVS scheduler. The EDF scheduler always assigns the task with the earliest deadline the highest scheduling priority. The DVS scheduler then determines the processor voltage and frequency during the execution of a particular task. These two components are independent of each other so that the EDF scheduler is capable of working with different DVS algorithms. Our DVS scheduler is based on feedback control that incrementally adjusts system behavior in order to reduce energy consumption. EDF is especially attractive to DVS algorithms because of its dynamic assignment of task priority, which allows the DVS scheduler to maximally exploit slack for each task.

A periodic, fully preemptive and independent task model is used in the 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 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}. The hyperperiod H of the task set is the least common multiplier (LCM) among the tasks' periods.

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

3. Feedback DVS Algorithm

Our feedback DVS algorithm anticipates an actual execution time of each task instance based on the feedback information from previous invocations. It splits the execution budget of a task into two sub-tasks T_A and T_B, as depicted in Figure 1. Under the maximal frequency, the worst-case execution time of these two subtasks under the maximal frequency are represented as C_A and C_B ($C_B = WCET - C_A$), respectively. Feedback DVS tries to scale T_A at the lowest possible

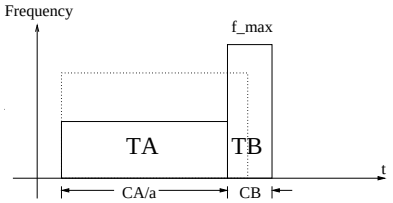

Figure 1. Task Splitting

frequency while T_B is always scaled at the maximum frequency to meet the deadline requirements of the real-time task. $\alpha = \frac{C_A}{C_A + s_k}$ is the scaling factor which determines the corresponding processor frequency and voltage for the T_A subtask. s_k is the available amount of slack to the task derived from the worst-case execution profile. Feedback DVS keeps the total system utilization below 100% even with reduced processor frequency and voltage. This guarantees the schedulability of the hard real-time task set (for algorithmic details, see [25]). The algorithm is capable of capturing variations in actual execution times through the feedback scheme. Due to the even more greedy approach than any of the previous schemes, the algorithm was reported to exhibit additional energy savings in simulation experiments, particularly for medium utilization systems, which are quite common [5]. Even more substantial savings have been observed for fluctuating execution times where PID-feedback provides new opportunities for aggressive scaling. In the implementation of the algorithm for the 405LP embedded board, we refined the feedback scheme proposed in [25] and developed the following feedback mechanisms.

3.1 Simple Feedback

If a periodic real-time workload exhibits a relatively stable behavior during a certain interval of time, the actual execution time of different jobs remains nearly constant or 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 described in the next section, because a simple feedback usually provides sufficiently accurate predictions. 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 later discuss in the context of Table 3.

In this simple feedback mechanism, we choose the value of C_A as the controlled variable. 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 is reserved at the maximum frequency. The maximum frequency portion guarantees the deadline requirements, 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. 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:

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

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 during some interval. When the workload's behavior keeps changing dynamically with highly fluctuating execution times, simple feedback becomes not enough to yield the best energy savings. In those cases, a more sophisticated feedback mechanism is required, as detailed in the next section.

3.2 PID Feedback

The original PID-feedback DVS mechanism, as presented in [25], requires the DVS scheduler to create and maintain multiple independent feedback controllers for each of the tasks in the workload. Multiple inputs and multiple outputs need to be manipulated simultaneously by the DVS scheduler. Such a PID-feedback mechanism, albeit its potential for energy savings shown in our previous simulation experiments, results in substantial execution overhead on an embedded architecture. Given the difficulty of precisely characterizing the behavior of a multiple-input multiple-output control system, it also adds complexity to the theoretical analysis of the algorithm. Therefore, we refine the original PID-feedback DVS mechanism by the following simplified design.

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:

$$r_j = \frac{1}{n}\sum_{i=1}^{n}\frac{C_{Aij} - c_{ij}}{c_{ij}} \tag{2}$$

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

$$\epsilon(r_j) = r_j - 0. \tag{3}$$

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:

$$\begin{aligned} \Delta r_j &= K_p\epsilon(r_j) + \tfrac{1}{K_i}\textstyle\sum_{IW}\epsilon(r_j) + K_d\tfrac{\epsilon(r_j)-\epsilon(r_{j-DW})}{DW} \\ r_{j+1} &= r_j + \Delta r_j \end{aligned} \tag{4}$$

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.

In order to analyze the performance of such a feedback control system, we compute its transfer function in the Laplace domain. The transfer function of the PID controller is defined as:

$$G_{PID}(s) = K_p + \frac{K_i}{s} + K_d s \tag{5}$$

The transfer function between r_j and C_A can be derived by taking derivative of both sides of the equation 2:

$$G_r(s) = Ms \tag{6}$$

where $M = \frac{1}{n}\sum_{i=1}^{n}\frac{1}{c_i}$. Therefore, the transfer function of the entire closed-loop feedback system can be computed as :

$$\frac{G_{PID}(s)G_r(s)}{1 + G_{PID}(s)G_r(s)} = \frac{MK_p s + MK_i + MK_d s^2}{1 + MK_p s + MK_i + MK_d s^2} \tag{7}$$

According to control theory, a system is stable if and only if all the poles (the denominator of its transfer function) are in the negative half-plane of the s-domain. From Equation 7, we infer the poles of our system as

$$\frac{-MK_p \pm \sqrt{MK_p^2 - 4MK_d(MK_i + 1)}}{2MK_d} \tag{8}$$

Note that $-MK_p + \sqrt{MK_p^2 - 4MK_d(MK_i + 1)}$ is always less than 0 when $MK_p^2 - 4MK_d(MK_i + 1) > 0$. Hence,

all the poles are in the negative half-plane of the s-domain. Therefore, the stability of the above system is ensured.

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. But it also has its drawback, *i.e.*, it does not provide direct feedback information of the C_A value for each individual task. When r equals zero, one cannot infer that every task's C_A has approximated its actual execution time. It is an imprecise description of the original scheduling objective and may take longer to get the system into a stable state. Nonetheless, our experiment shows significant energy savings of this feedback DVS mechanism compared to other DVS algorithms. In the next section, we present the details of our experimental results.

4. Experimental Evaluation

By evaluating our feedback DVS algorithm on a real embedded architecture, we assess the true potential of our algorithm for energy savings in an actual system as opposed to a simulation environment. Also, we compare the overhead and energy consumption between our algorithm and several other DVS algorithms, namely static DVS, cycle-conserving DVS, look-ahead-1/2 DVS (all by Pillai and Shin [20]) as well as DR-OTE and AGR-2 (by Aydin *et al.* [1]). Look-ahead-1 and look-ahead-2 are the original and a modified version of the look-ahead DVS algorithm in [20], respectively. Look-ahead-1 updates each task's absolute deadline immediately when a task instance completes. Look-ahead-2 delays such update until the next task instance is released, which results in additional energy savings. AGR-2 follows the most aggressive scheme presented in [1] with an aggressiveness parameter k of 0.9. In these experiments, 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. We use our PID feedback approach unless explicitly stated that we use the simple scheme from Section 3.1.

4.1 Platform and Methodology

The embedded platform used in our experiment is a PowerPC 495LP 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 [18, 3, 9]. 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 [20] and our feedback DVS) were exposed to the DVS capabilities of the 405LP board. In our experiments, the frequency and voltage pairs depicted in Table 1 were chosen.

Table 1. 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 [3] 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 consumption 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 application 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.

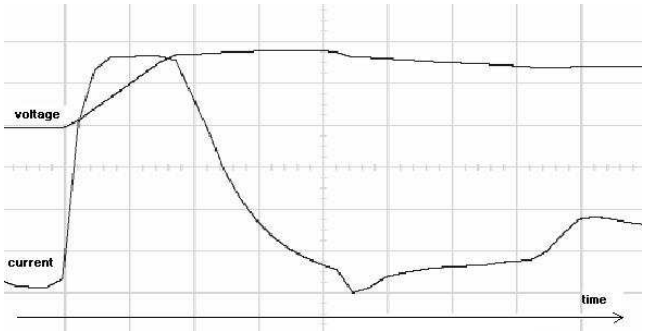

Figure 2. Current/Volt. for Async. Freq. Switch

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 2 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 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 (using 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.

Table 2. 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

4.3 DVS Scheduler Overhead

We compared the 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 3. 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. It reflects the overall scheduling overhead when different DVS algorithms are integrated into the scheduler.

Table 3. Overhead of DVS-EDF Scheduler

	DVS scheduling overhead[μsec]			
CPU freq.	static	cc	look-ahead	PID-feedback
33 MHz	217	487	2296	3612
44 MHz	170	366	1714	2943
66 MHz	100	232	1112	1728
133 MHz	52	120	546	801
266 MHz	36	76	229	472

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 selects a unified frequency and voltage setting for the entire task set. No dynamic voltage and frequency modulation takes place during task context switches. But the static scheme also falls 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 trade-off between overhead and performance always needs to be examined carefully.

Next, we 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 4 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.

The first task set in Table 4 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 5 depicts the energy consumption of different DVS algorithms in mWatt-hours. The naïve DVS algorithm serves as a base of comparison for each of the subsequent DVS algorithms. The absolute energy consumption value, as well as the percentage of energy savings over the naïve DVS,

Table 4. 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

Table 5. Energy $[mW - hrs]$ consumption per RT-DVS algorithm

algorithm	naïve	static(savings)	cycle-cons.(savings)	look-ahead(savings)	our feedback(savings)
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 savings	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 savings	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 savings	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%

is presented for each 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 for each algorithm are lower for task set two peaking at about 23% for 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 5) while energy consumption decreases.

Another noteworthy result is the comparison between synchronous and asynchronous DVS switching depicted in the last row labeled "sync/async savings" for each task set in Table 5. For each of the scheduling algorithms, we see additional savings of 1-5% on asynchronous switching 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 time settings used in our experimental environment are still large compared to the execution time of a synchronous or asynchronous switching. If we only save about 100 μsec at each frequency switch (as has been shown in Table 2) 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 very last row of Table 5 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.

4.4 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 periodic tasks 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 3) 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.

Altogether, four synthesized execution patterns were created. 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 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. Asynchronous switching was used in the experiment. We executed multiple runs for each setting and report the average energy consumption.

Figures 3 and 4 present the energy consumption of our feedback-DVS as well as four other dynamic DVS algorithms. The number of tasks in the task set varies between 3 and 30 tasks, and the tasks' actual execution times follow the dynamic execution pattern 2. All energy values are normalized to the naïve DVS results. AGR-2 dynamically reclaims unused slack up to the next arrival time of any task instance (NTA), hence saving about 50% extra energy than naïve DVS. AGR-2 is not as good as Look-ahead-1/2 DVS for 3 tasks since it considers slack only up to the next task instance's deadline, while Look-ahead DVS collects slack up to the largest deadline among all tasks. But AGR-2 benefits from smaller task granularity in 30-task sets and outperforms Look-ahead-1 and Look-ahead-2 in some utilization 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. Feedback-DVS saves another 5%-20% energy over look-ahead DVS and AGR-2 due to the algorithm's self-adaptation to jobs' actual execution times. In cases of extremely low utilization, feedback-DVS, Look-ahead DVS and AGR-2 are observed to result in virtually the same energy savings because every task has enough slack to run at the minimum speed, resulting in the same frequencies for a schedule irrespective of the DVS algorithm. Since look-ahead-2 DVS results in the lowest energy consumption on average among all other algorithms, we now focus on the comparison of our feedback-DVS algorithm with look-ahead-2 DVS for task sets which contain three tasks. We want to evaluate in detail the behavior of our algorithm with different execution time patterns. Figure 5 shows the average energy savings of our algorithm among the four patterns as well as the maximum and the minimum savings, which are represented as the upper bound and the lower bound on each error bar. When the variation of tasks' actual execution time follows the four different patterns, feedback-DVS still exhibits stable energy saving performance in all utilization cases. It saves up to 21% and 65% more energy than look-ahead-2 and naïve DVS. The largest savings again occur in median utilization cases where there is considerable

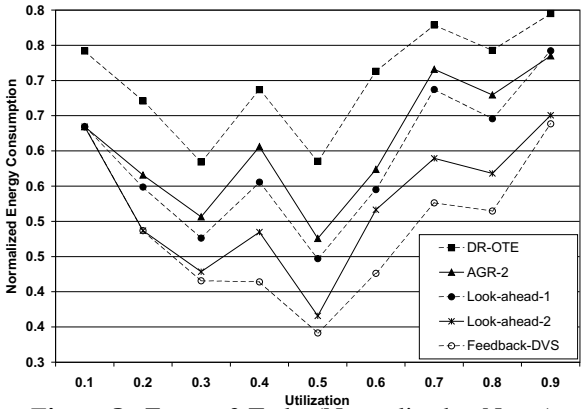

Figure 3. Energy 3 Tasks (Normalized to Naïve)

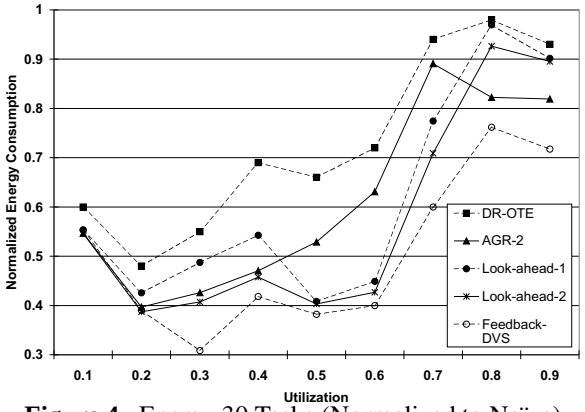

Figure 4. Energy 30 Tasks (Normalized to Naïve)

dynamic slack for speed reduction. Variations of energy savings never exceed 10% of the average savings among different execution time patterns.

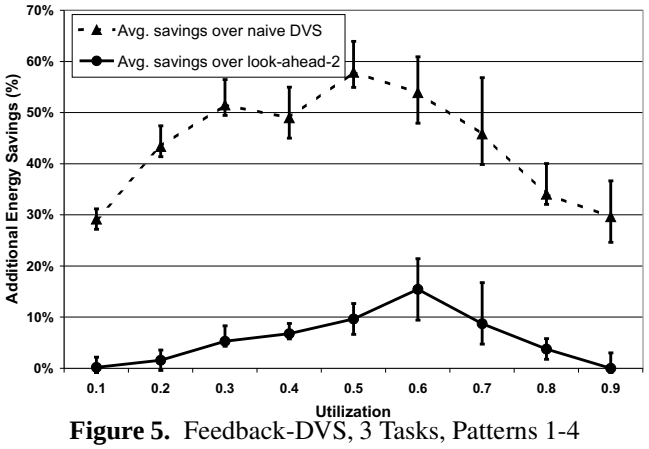

Figure 5. Feedback-DVS, 3 Tasks, Patterns 1-4

To better assess the scalability of our feedback-DVS algorithm, we further fixed the execution time pattern of the task set, while varying the baseline of its actual execution times. Figure 6 shows the energy consumption on a task set with execution time pattern 4, whose average execution time changes from 0.75WCET to 0.3WCET. All energy values are normalized to the naïve DVS values. We see from the figure that our algorithm 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 DVS. Our PID-feedback mechanism shows even better strength for median execution times than the loose or tight ones. In this range, there is enough slack to distinguish itself from the other algorithms.

Figure 7 depicts 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. 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-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 further reduce power consumption occasionally as sufficient slack exists to be recovered by the algorithms of the previous schemes. Dynamic slack is recovered in increasing levels by the latter three schemes.

4.5 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 [25], we clearly 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 show a consistent trend under both simulation and the actual embedded platform. But 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 report as 29% in simulation while the best result we measured from the test board is around 24%. It is non-trivial to model the actual power/energy consumption in simulation without considering actual hardware details. This is also the case when evaluating the overhead. Since 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 measurement from the test board show only 3%-6% savings for these cases.

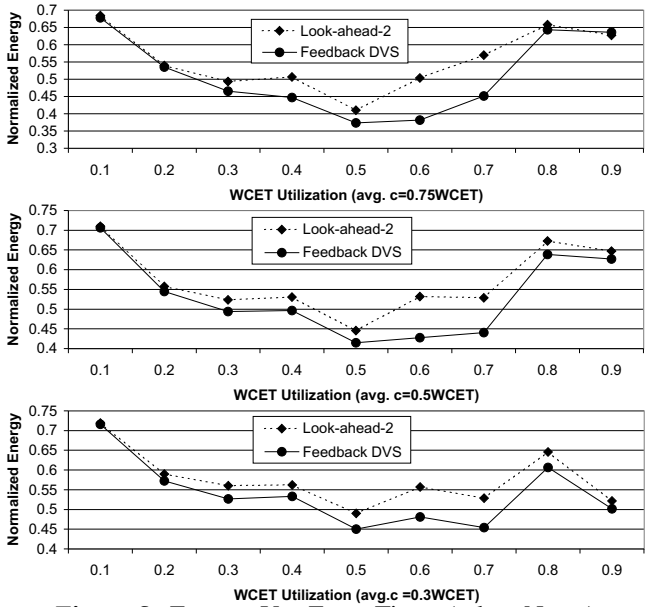

Figure 6. Energy: Var. Exec. Times (rel. to Naïve)

Overall, our experiments on the embedded platform quantitatively show the potential of our feedback DVS algorithm and its ability to scale power even more aggressively than previous DVS algorithms.

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 [20]. 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 [1]. Energy-aware scheduling of hybrid workloads, including both periodic and aperiodic tasks,

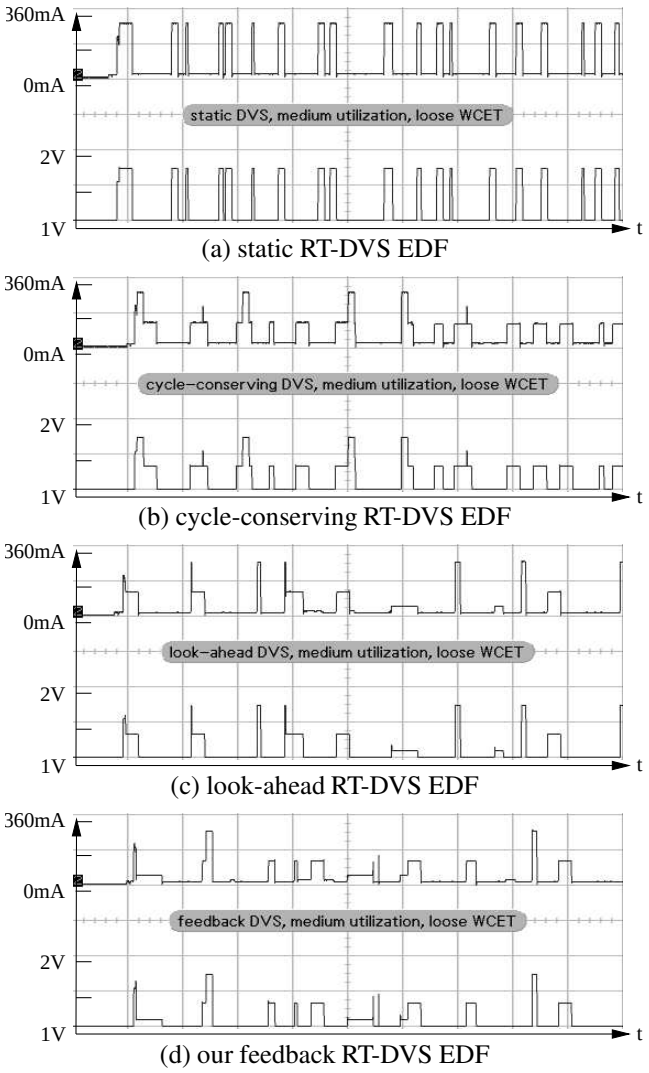

Figure 7. Voltage/Current Oscilloscope Shot, loose WCET= 2× ActualExecTime, U=0.5

are further investigated by Aydin and Yang in [2]. Gruian analyzes a dual-speed DVS schedule based on stochastic data derived from past task execution traces [7]. Jejurikar and Gupta investigate static and dynamic slowdown factors for periodic tasks [11] and combine it with procrastination scheduling [12] and preemption threshold scheduling [10] for DVS. Several of these algorithms were compared in a unified simulation environment, SimDVS [21]. 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.* [22]. Real-time system performance specifications are analyzed systematically through a control-theoretical methodology by Lu *et al.* [14]. 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 [15]. Our work extends feedback to power-aware EDF scheduling.

Feedback control was also proposed for energy-aware computing in previous work. Varma *et al.* 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 [23]. 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.* describe a formal feedback-control algorithm combined with dynamic voltage/frequency scaling technologies [16]. A general energy management scheme with feedback control is proposed by Minerick *et al.* [17]. Average energy usage is achieved by continuously adjusting the voltage/frequency of a processor to meet the energy consumption goal. While Varma and Lu's work targets soft 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

In this paper, we presented feedback DVS algorithm considering practical design and implementation issues. We evaluated it as well as several other real-time DVS algorithms on an IBM 405LP embedded platform. A unique DVS feature of this platform is asynchronous frequency switching, which supports continued execution during voltage/frequency transitions. We have shown up to 5% energy savings of asynchronous switching for fast DVS modulation without entering sleep modes as opposed to traditional synchronous switching. We assessed the benefits of our feedback DVS algorithm by measuring the energy consumption over the hyperperiod of real-time tasks. Energy consumption as well as scheduling overhead between different DVS schemes were compared with each other. The experimental results indicate that our aggressive feedback DVS scheduling algorithm achieves up to 24% savings in energy consumption best competitors, *i.e.*, over the look-ahead DVS and AGR-2 algorithms, and up to 64% energy savings over the naïve DVS scheme when considering scheduling overheads. 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

A. Dudani contributed to early work of the Feedback-DVS scheme [5]. A. Anantaraman, A. Mahmoud and R. Venkatesan designed and implemented an early version of the DVS environment. Bishop Brock (IBM) provided most valuable technical details for the PPC 405LP board donated by IBM Research (Austin). This work was supported in part by NSF grants CCR-0208581, CCR-0310860 and CCR-0312695.

References

[1] H. Aydin, R. Melhem, D. Mosse, and P. Mejia-Alvarez. Power-aware scheduling for periodic real-time tasks. *IEEE Trans. Comput.*, 53(5):584–600, 2004.

[2] H. Aydin and Q. Yang. Energy-responsiveness tradeoffs for real-time systems with mixed workload. In *Proceedings of the 11th IEEE Real-Time and Embedded Technology and Applications Symposium*, May 2004.

[3] B. Brock and K. Rajamani. Dynamic power management for embedded systems. In *IEEE International SOC Conference*, Sept. 2003.

[4] A. Chandrakasan, S. Sheng, and R. W. Brodersen. Low-power cmos digital design. In *IEEE Journal of Solid-State Circuits, Vol. 27, pp. 473-484.*, April, 1992.

[5] A. Dudani, F. Mueller, and Y. Zhu. 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)*, pages 213–222, June 2002.

[6] K. Govil, E. Chan, and H. Wasserman. Comparing algorithms for dynamic speed-setting of a low-power cpu. In *1st Int'l Conference on Mobile Computing and Networking*, Nov 1995.

[7] F. Gruian. 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*, Aug 2001.

[8] D. Grunwald, P. Levis, C. M. III, M. Neufeld, and K. Farkas. Policies for dynamic clock scheduling. In *Symp. on Operating Systems Design and Implementation*, Oct 2000.

[9] IBM and MontaVisa Software. Dynamic power management for embedded systems. white paper.

[10] R. Jejurikar and R. Gupta. 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)*, 25-27 Aug 2004.

[11] R. Jejurikar and R. Gupta. Optimized slowdown in real-time task systems. In *Proceedings of the 16th Euromicro Conference on Real-Time Systems (ECRTS '04)*, Jun30-Jul2 2004.

[12] R. Jejurikar and R. Gupta. Procrastination scheduling in fixed priority real-time systems. In *Proceedings of the Language Compilers and Tools for Embedded Systems*, Jun 2004.

[13] D. Kang, S. Crago, and J. Suh. A fast resource synthesis technique for energy-efficient real-time systems. In *IEEE Real-Time Systems Symposium*, Dec. 2002.

[14] C. Lu, J. A. Stankovic, T. F. Abdelzaher, G. Tao, S. H. Son, and M. Marley. Performance specifications and metrics for adaptive real-time systems. In *Proceedings of the IEEE Real-Time Systems Symposium*, December 2000. 2000.

[15] C. Lu, J. A. Stankovic, G. Tao, and S. H. Son. Feedback control real-time scheduling: Framework, modeling, and algorithms. *Real-Time Syst.*, 23:85–126, 2002.

[16] Z. Lu, J. Hein, M. Humphrey, M. Stan, J. Lach, and K. Skadron. Control-theoretic dynamic frequency and voltage scaling for multimedia workloads. In *Conference on Compilers, Architecture and Synthesis for Embedded Systems*, pages 156–63, 2002.

[17] R. Minerick, V. W. Freeh, and P. M. Kogge. Dynamic power management using feedback. In *Proceedings of Workshop on Compilers and Operating Systems for Low Power*, 2002.

[18] K. Nowka, G. Carpenter, and B. Brock. The design and application of the powerpc 405lp energy-efficient system on chip. *IBM Journal of Research and Development*, 47(5/6), September/November 2003.

[19] T. Pering, T. Burd, and R. Brodersen. The simulation of dynamic voltage scaling algorithms. In *Symp. on Low Power Electronics*, 1995.

[20] P. Pillai and K. Shin. Real-time dynamic voltage scaling for low-power embedded operating systems. In *Symposium on Operating Systems Principles*, 2001.

[21] 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 *Workshop on Power-Aware Computer Systems*, Feb. 2002.

[22] J. A. Stankovic, C. Lu, S. H. Son, and G. Tao. The case for feedback control real-time scheduling. In *Proceedings of the EuroMicro Conference on Real-Time Systems*, June 1999.

[23] A. Varma, B. Ganesh, M. Sen, S. R. Choudhury, L. Srinivasan, and J. Bruce. A control-theoretic approach to dynamic voltage scheduling. In *Proceedings of the 2003 international conference on Compilers, architectures and synthesis for embedded systems*, pages 255–266. ACM Press, 2003.

[24] M. Weiser, B. Welch, A. Demers, and S. Shenker. Scheduling for reduced cpu energy. In *1st Symp. on Operating Systems Design and Implementation*, Nov 1994.

[25] Y. Zhu and F. Mueller. Feedback edf scheduling exploiting dynamic voltage scaling. In *IEEE Real-Time Embedded Technology and Applications Symposium*, pages 84–93, May 2004.