Feedback-Based Dynamic Voltage and Frequency Scaling for Memory-Bound Real-Time Applications

Christian Poellabauer
Computer Science and Engineering
University of Notre Dame
Notre Dame, IN 46556
cpoellab@cse.nd.edu

Leo Singleton and Karsten Schwan
College of Computing
Georgia Institute of Technology
Atlanta, GA 30332
{lcs, schwan}@cc.gatech.edu

Abstract

Dynamic voltage and frequency scaling is increasingly being used to reduce the energy requirements of embedded and real-time applications by exploiting idle CPU resources, while still maintaining all application's real-time characteristics. Accurate predictions of task run-times are key to computing the frequencies and voltages that ensure that all tasks' real-time constraints are met. Past work has used feedback-based approaches, where applications' past CPU utilizations are used to predict future CPU requirements. Mis-predictions in these approaches can lead to missed deadlines, suboptimal energy savings, or large overheads due to frequent changes to the chosen frequency or voltage. One shortcoming of previous approaches is that they ignore other 'indicators' of future CPU requirements, such as the frequency of I/O operations, memory accesses, or interrupts. This paper addresses the energy consumptions of memory-bound real-time applications via a feedback loop approach, based on measured task run-times and cache miss rates. Using cache miss rates as indicator for memory access rates introduces a more reliable predictor of future task run-times. Even in modern processor architectures, memory latencies can only be hidden partially, therefore, cache misses can be used to improve the run-time predictions by considering potential memory latencies. The results shown in this paper indicate improvements in both the number of deadlines met and the amount of energy saved.

1. Introduction

Background. As mobile processors become more powerful, their increased energy demands can lead to shorter battery lives and greater heat dissipation. In order to reduce these demands while preserving high performance, energy-awareness has become a critical factor in the design of mobile and embedded systems. One outcome is the development of new methods for dynamic power management, such as the support of dynamic voltage and frequency scaling (DVFS) [8, 21, 13, 7] in microprocessors like Intel's XScale and StrongARM and Transmeta's Crusoe chips. DVFS takes advantage of the quadratic relationship between supply voltage and energy consumption [2], which can result in significant energy savings. However, DVFS techniques pose a difficult challenge to real-time systems: not only must DVFS be used to reduce energy consumption, but it must do so without impacting the desired quality of service offered by the system to applications and end users, e.g., previous work has introduced energy-aware CPU schedulers aiming at meeting all task deadlines while exploiting DVFS to reduce the processor's energy consumption. Recent approaches to solving this issue bases the frequency/voltage computations used by the scheduler on its predictions of future task run-times [26, 14]. However, these approaches ignore that applications' run-times also depend on I/O, memory accesses, or interrupts. Inaccuracies in run-time predictions can therefore lead to missed deadlines, suboptimal energy savings, or large overheads due to frequent recomputations and adjustments of the frequency and voltage.

This paper addresses inaccuracies in predicting task run-times, by removing one of their major causes, the utilization of memory resources. Memory-bound applications, including image and video processing and scientific applications, require frequent accesses to memory, which contributes substantially to their total run-times, particularly in systems where DVFS not only changes the CPU's clock frequency, but also the frequency of the bus linking CPU and memory (e.g., in XScale-based systems). The memory overheads experienced by these tasks also depend on their I/O activities, which can increase bus or memory loads, thereby also increasing memory access times. The key approach explored in this paper, therefore, is to capture the memory overheads experienced by memory-bound tasks and then use these overheads to better predict task run-times. The method used is to dynamically monitor a task's cache miss rate, which determines the memory access rate experienced by this task. Dynamically monitored cache miss rates are used as feedback input by the DVFS algorithm, which then computes the frequencies and voltages to be used for task execution.

In order to utilize DVFS, the CPU scheduler must cooperate closely with the DVFS algorithm. The two main questions that need to be answered by each DVFS approach are: (a) *when* to change the frequency and voltage and (b) *how* to change it. In our approach, each task receives a time slice C in every period of T time units, and tasks are scheduled according to the Earliest Deadline First (EDF) algorithm. Utilizing all tasks' scheduling attributes, the DVFS algorithm can compute the total CPU utilization, and consequently, the frequency and voltage combination that maximizes this utilization (i.e., 100%). However, inaccuracies in predicting tasks' run-times can cause system utilization to exceed 100%, thereby leading to missed deadlines and frequent frequency and voltage re-computations. Overestimating the utilization can result in inefficient energy management. One of the reasons for these inaccuracies are the frequent memory accesses of memory-bound applications. Even modern processors are able to hide these memory access latencies only partially. We therefore address this issue by monitoring the cache miss rates of all tasks to obtain an estimate of the memory latencies and to improve the frequency and voltage computations of our DVFS approach. In the evaluation section of this paper we evaluate several applications at different CPU utilizations, showing improvements in energy savings in about 27% of all cases. Our future work will extend this feedback approach to I/O-bound applications (memory and disk accesses), thereby addressing the needs of energy-aware communication-intensive applications (e.g., real-time sensor networks).

Contributions and Related Work. There has been substantial prior work on energy management for mobile multimedia systems, including low-power modes for disks and networks [5, 3], energy-aware scheduling policies [19, 10, 7, 15], and energy management techniques for wireless communication [1, 11, 16, 18, 17].

The main contribution of this work is the implementation and evaluation of a feedback-based DVFS approach for real-time systems that adjusts predictions of future system utilizations based on monitored memory accesses. The resulting approach to DVFS can lead to higher energy savings, reduced deadline misses, or a reduced number of recomputations of frequencies and voltages. Our experimental results utilize the voltage and frequency scaling capabilities of an XScale-based evaluation board. In comparison, previous work on DVFS algorithms for real-time schedulers described in [13] and [19] relies on worst-case task execution times, whereas we use cache feedback data to approximate actual execution times. There has been substantial prior work that examines the effects of DVFS on cache performance. In [9], the authors evaluate energy savings due to the reduced cache miss penalties at lower clock frequencies. Simulation results with a SimpleScalar simulator show that a substantial reduction in energy consumption can be achieved with minimal performance degradation. Compile-time techniques have exploited these stall periods, e.g., in [6], the authors implement a compiler-directed dynamic voltage scaling algorithm that achieves energy savings of up to 55%, with only a 6% performance penalty. In [22], the authors investigate the use of feedback loops to refine approaches such as PAST [24] by considering the rate of change in a system's processing requirements. In [25], the authors also distinguish between processing and memory access, however, their work is limited to frequency scaling, where each task operates at its own speed. With voltage scaling, approaches that compute frequency and voltage pairs over the entire task set are preferable due to the reduced changes in speed settings and thereby reduced overheads. The FAST approach [20] focuses on static timing analysis for simple in-order single issue pipeline. Although the authors argue that static timing analysis is preferable over dynamic approaches (because they do not provide any worst-case execution time guarantees), our results for our feedback-based approach on an out-of-order completion processor show that 'real' applications (e.g., gzip) have cache miss rates that do change, but by no means in a random way. Finally, in [4], the authors utilize cache miss feedback based on performance counters as shown in our work, however, focusing on best-effort tasks. For example, to limit the overheads of frequent voltage changes, they use a time quantum of 60ms, which prohibits rapid changes often needed in real-time systems. Our approach therefore can change voltages at any time, however, it uses a Schmitt-trigger-style function to limit the number of voltage changes and the overheads associated with these changes.

2. Dynamic Voltage and Frequency Scaling

Real-Time CPU Scheduling. The traditional UNIX scheduler has been shown to have unacceptable performance for multimedia applications [12]. In our approach, to efficiently support real-time applications, each task i has a period T_i associated, where the end of the current period is the deadline. Each task i is guaranteed C_i time units each period T_i, and the scheduler uses Earliest Deadline First (EDF) among all currently 'eligible' tasks (i.e., tasks that have not yet executed for their time slices in their respective periods). If a task misses its deadline, the scheduler violated the real-time guarantees to this task. Further, each task can run for at most its assigned service time in its period, unless it is marked as *work-conserving*, in which case it is possible to schedule this process several times within its period as long as CPU utilization allows.

Clock frequencies are typically computed such that the *rest utilization* of the CPU is exploited by slowing down task execution. We compute a new clock frequency whenever the system utilization changes, e.g., when tasks join or leave the run queue. The current utilization of all tasks is computed with:

$$U = \sum \frac{C_i}{T_i} \tag{1}$$

C_i is the service time allocated to task i at the default clock frequency and this service time increases when the clock is slowed down. For each clock frequency n, a *scaling factor* k_n can be obtained by executing a sample processing-intensive code at both the default frequency f_{max} and f_n and dividing the measured run-times: $k_n = C_n/C_{max}$. This is repeated for each available clock frequency (or core voltage) for a given processor. The goal of frequency scaling is to get as close to 100% utilization as possible, i.e.,

$$U_{100\%} = \sum \frac{C_i * k'}{T_i} \tag{2}$$

where k' is the yet unknown scaling factor. To guarantee that best-effort tasks are not starved, we can replace $U_{100\%}$ with $U_{x\%}$, e.g., $U_{95\%}$. Then k' can be determined with:

$$k' = \frac{U_{95\%}}{\sum \frac{C_i}{T_i}} \tag{3}$$

The resulting k' is compared to the previously obtained scaling factors, and the scaling factor k_n closest to k' ($k_n \leq k'$) is selected, and the clock frequency is adjusted to frequency n.
Note that other approaches exist that compute different scaling factors for each individual application, however, these approaches have higher computational overheads due to more frequent voltage and frequency changes. Nevertheless, the solution introduced in this paper could also be applied to these approaches. However, this common approach to DVFS assumes that the run-times can be predicted by monitoring previous CPU requirements, ignoring that other factors (e.g., I/O utilization) can affect the run-times, making the predictions inaccurate. This issue is addressed by the remainder of this paper.

Deficiencies and Conclusion. Intuitively, the scaling factor k' would be the ratio of the clock frequencies. However, various factors, including cache misses penalties, context switching, and I/O requests, cause a change in k'.

Figure 1 shows the performance of three applications, relative to the maximum frequency (i.e., the inverse of the execution times relative to the execution time at the maximum frequency). On the x-axis, we display the different frequencies supported by an XScale-based device: the first number depicts the core clock frequency (ranging from 99MHz to 398MHz); the second number is the bus frequency (ranging from 50MHz to 196MHz). This results in 5 different frequency settings. Note however, that the fourth and fifth settings differ only in the bus frequency. The three programs selected are dcache-miss, a C program written specifically to generate data cache misses by performing memory accesses on a large array; gzip, the popular file compression application; and cjpeg, a JPEG image compressor. The theoretical line shows the expected ratio, assuming that the execution time was linearly related

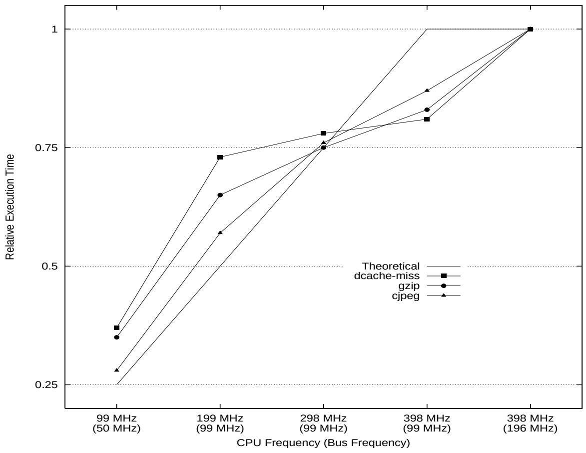

Figure 1. Relative performance of three applications at various clock frequencies on the Intel XScale PXA255.

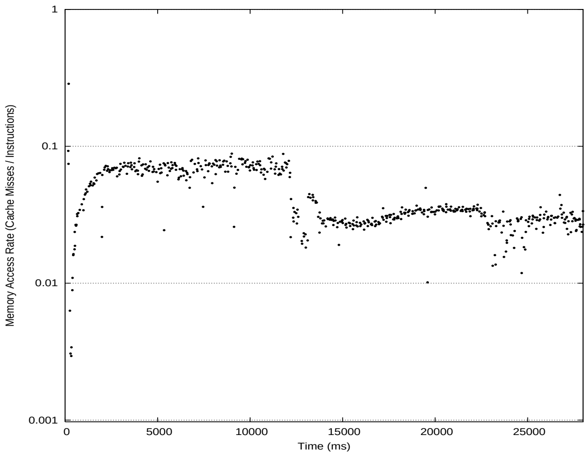

Figure 2. Data memory access rate of gzip.

to the CPU frequency. With memory-intensive programs like gzip and dcache-miss, execution times differ substantially from the theoretical line, since the execution times of these programs are more dependent on the bus frequency than the CPU frequency. The third application, cjpeg, is less memory-intensive, showing an expected ratio closer to the theoretical line compared to the two other applications. Intuitively, each process by itself or each combination of different processes could require a different k' factor, however, the approach addressed in this paper uses one k' factor for all processes currently on the run queue. If a simple linear model were used to determine the k' factors, it would underestimate the utilization at lower clock frequencies, resulting in missed deadlines, and overestimate the utilization at higher clock frequencies, resulting in wasted energy. Also, notice that the curves nearly converge at three points: 99MHz (50MHz), 199MHz (99MHz), and 398MHz (196MHz). In this example, there is roughly the same ratio between CPU and bus frequencies, so cache miss rates have little effect on the execution time. The deviations shown in Figure 1 point out the importance of considering cache miss rates in the DVFS approach to accurately predict future task run-times. Note that these measurements have been performed on an XScale processor, a modern mobile scalar processor with a 7-stage pipeline and out-of-order completion. While modern processors can hide certain memory latencies, the hundreds of cycles resulting from DRAM accesses can cause significant performance losses [23]. Therefore, we use cache misses as indicator for the performance penalties incurred by memory accesses.

The remainder of this paper will use the *Memory Access Rate (MAR)* to quantify the rate of cache misses. As opposed to the *miss rate*, MAR is defined in Equation 4 as the ratio of data cache misses to instructions executed.

$$MAR = \frac{data\ cache\ misses}{instructions\ executed} \tag{4}$$

Figure 2 shows gzip's MAR plotted over time. Applications like gzip show a behavior where overall the miss rate is rather constant, but can change dramatically as shown in the figure (e.g., different 'phases' of application execution). In this example, the MAR value can vary by a factor of over a hundred. Due to these variations, a single set of k' values would not be sufficient for DVFS over the entire execution of gzip.

3. Cache Miss Rates as Feedback for DVFS

Approach. Previous approaches to feedback-based DVFS and CPU scheduling have neglected that the utilization of resources, such as disk, network, or memory, can significantly affect the run-time of applications, making run-time predictions, and thereby frequency/voltage computations, inaccurate. These inaccuracies can result in missed deadlines for real-time applications, inefficient energy management (e.g., the voltage and frequency are chosen too high), or too frequent re-adjustments of voltage and frequency. To improve predictions of execution time, we monitor cache miss rates using a software feedback loop. This is particularly important for memory-bound applications real-time applications (e.g., image processing), where frequent memory accesses introduce additional latencies. Our DVFS approach uses performance counters to count the number of data cache misses and instructions executed, and keep a weighted average of the MAR for each process (Figure 3), i.e., the most recent reading of MAR plus the previous reading of MAR multiplied by 0.5. From this value, the CPU scheduler can pick an appropriate matrix of scaling factors, which will more accurately estimate the process's execution time under the available frequencies.

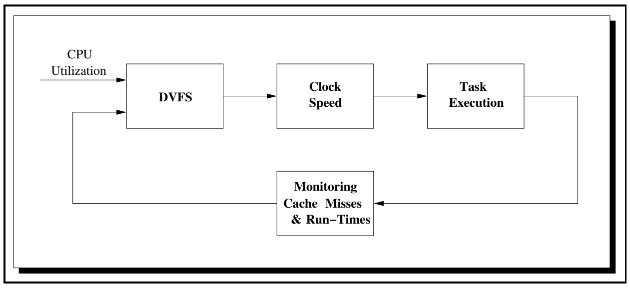

Figure 3. Feedback-based DVFS.

Our approach is implemented on an Intel Sitsang evaluation board with an XScale PXA255 processor. The XScale PXA255 supports various CPU frequencies, ranging from 99 MHz to 398 MHz and bus frequencies ranging from 50 MHz to 196 MHz. However, not all frequency combinations may be used for DVFS, since some change the LCD or SDRAM frequencies. Of the possible combinations, the 5 frequency levels supported in the Linux kernel are listed in Table 1.

CPU Frequency	Bus Frequency	Min. CPU Voltage
99 MHz	50 MHz	0.8V
199 MHz	99 MHz	1.0V
299 MHz	99 MHz	1.1V
398 MHz	99 MHz	1.3V
398 MHz	196 MHz	1.3V

Table 1. Supported CPU / Bus Frequencies.

Cache	Size	Block Size	Associativity	Policy
Instruction	32 KB	32 B	32-Way	Round-Robin
Mini-Data (L1)	2 KB	32 B	2-Way	Round-Robin
Data (L2)	32 KB	32 B	32-Way	Round-Robin

Table 2. XScale PXA255 Caches.

The Sitsang evaluation board contains 32 MB of flash memory and 64 MB of SDRAM. The XScale PXA255 processor has three caches, summarized in Table 2. For the purposes of this project, we will only be monitoring cache misses on the 32 KB data cache, future work will also include the instruction cache.

The system runs a modified Linux 2.4.19 kernel, with Sitsang patches provided by Intel and a patch implementing our replacement real-time CPU scheduler.

Performance Counters. Many processors support some form of performance counters for optimization. These hardware counters support the monitoring of numerous types of events, such as memory accesses and pipeline stalls. The Intel XScale PXA255 processor has three 32-bit performance counters: one which counts clock cycles and two general-purpose counters, which may be used to monitor any of 16 possible events, selectable in software via a configuration register[1].

For these experiments, it is important to choose performance counters which are not affected by frequency changes. Since the execution time varies with frequency, cache misses over clock cycles would vary for a given program depending on the frequency. Therefore, for the computation of MAR, instructions executed was chosen instead of clock cycles. Unfortunately, the PXA255 does not provide a performance counter for combined cache misses, only separate ones for the instruction and data caches. Due to the limitations of the processor, data cache misses was chosen. Hence, the definition of the MAR as the ratio of data cache misses to instructions executed (Equation 4).

Monitoring of Cache Miss Rates. To implement an energy-aware scheduler using cache feedback, we first must approximate the execution time of a process with a given MAR. If we take a program's execution time and break it into two components – the time spent executing instructions and the time spent stalled on memory accesses – we get the following formula:

$$t_{execution} = C_{CPU}(f_{CPU}) + (MAR * C_{bus}(f_{bus})) \quad (5)$$

1 http://www.intel.com/design/pca/prodbref/252780docs.htm

where $C_{CPU}(f_{CPU})$ is a constant dependent on CPU clock frequency, and $C_{bus}(f_{bus})$ is a constant dependent on bus frequency. To find the values for these two constants, a test program was designed to generate a specified miss rate by performing memory accesses on a large array. The test program was run with multiple MARs at each supported frequency, and using linear regression, the execution times were used to determine the constants C_{CPU} and C_{bus}. The experimental determination of the C_{bus} constants ensures that processor techniques to hide memory latencies are considered, i.e., C_{bus} only indicates the memory latencies that could not be hidden by the processor.

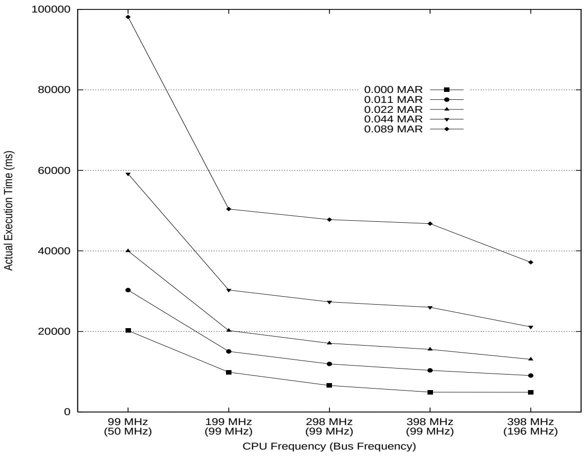

Figure 4. Actual execution time of our test program.

Figure 4 displays the actual – measured – execution times of our test program. In contrast, Figure 5 shows the predicted execution times, based on Equation 5. The results of these two figures are almost identical, underlining the importance of adding caching information to the prediction of task run-times.

The scaling factors for each frequency can be calculated using the ratio of the execution times from the equation above. Calculating these scaling factors at runtime would add extra overhead to each scheduler invocation, so twelve matrices of scaling factors were pre-calculated and compiled into the DVFS module. The twelve matrices were se-

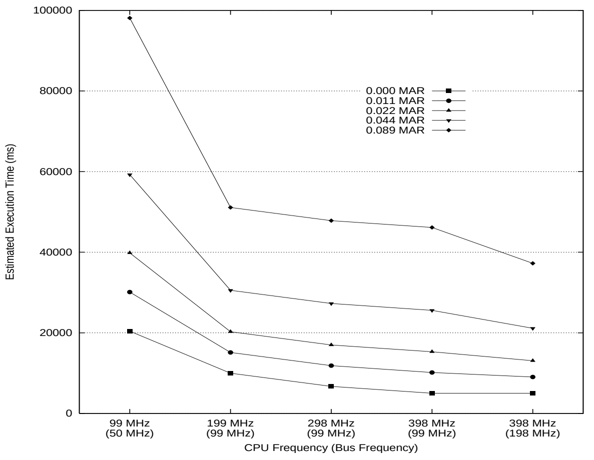

Figure 5. Estimated execution time using Equation 5.

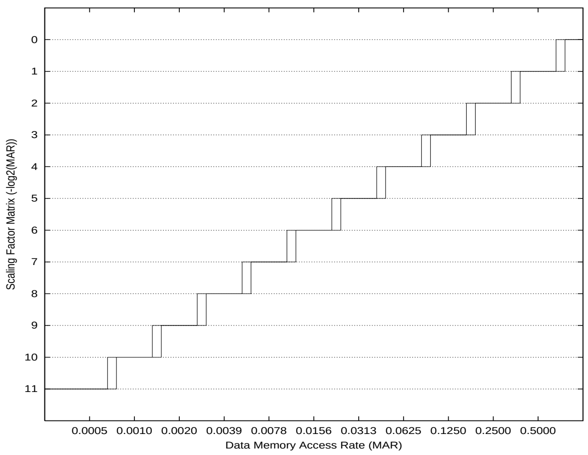

Figure 6. Schmitt-trigger-style function mapping MARs to matrices of scaling factors.

lected such that matrix M_n corresponds to the scaling factors for processes with an MAR of 2^{-n}.

A field was added to Linux's task_struct to keep a weighted average of the MAR for each process. The scheduler was modified to support DVFS using the following algorithm:

```
 1:  prev->dcache_misses=(prev->dcache_misses/2)
             +pmu->dcache_misses;
 2:  prev->instructions=(prev->instructions/2)
             +pmu->instructions;

 3:  schedule();

 4:  inverse_mar=next->instructions/next->dcache_misses;
 5:  n=schmitt_function(inverse_mar);

 6:  M_n=freq_matrix[n];

 7:  new_freq=find_minimum_freq(M_n, cur_freq, utilization);

 8:  if (cur_freq != new_freq)
 9:      set_cpu_freq(new_freq);

10:  reset_pmu_counters();
```

Lines 1-2 update the weighted average of the previous process's MAR. Line 3 invokes the CPU scheduler, selecting the next process. Once the next task is selected, we can compute n, the $-log_2$ of the next process's MAR, using the Schmitt-trigger-style function in Figure 6 (lines 4-5). This function avoids frequent changes in voltage and frequency, thereby avoiding the large overheads associated with these changes; for the test programs used in this paper we measured at most 3 voltage/frequency changes for the entire application run-time. The closest of the 12 pre-computed frequency matrices, M_n is selected from the lookup table (line 6). Line 7 uses the matrix of scaling factors (see Section 2) to determine the minimum frequency needed to meet the process's deadline (i.e., a simple matrix lookup), and if this frequency is different than the current frequency, the processor frequency is changed (lines 8-9). Finally, the XScale's PMU counters are reset before the next process begins execution (line 10).

4. Experimental Evaluation

This section verifies the modified DVFS approach based on cache information with multiple applications. The first part of this section addresses the overheads associated with our approach and the second part tests the effectiveness of the cache feedback loop. The scaling factors are computed to bring the system's utilization as close as possible to 100%.

Microbenchmarks and Overheads. The following list summarizes the noteworthy overheads associated with DVFS in general, and with our approach in particular.

- According to the PXA255 User's Guide[2], the worst-case time for a frequency change is $500\mu s$, which underlines the importance of limiting the number of voltage/frequency changes (e.g., using the Schmitt-trigger-style function described above).
- The PXA255 performance counters are mapped into memory. Reading the three performance counters requires only three memory reads per context switch. Resetting the counters at each context switch requires an additional read and write to the control register.

2 http://www.intel.com/design/pca/prodbref/252780docs.htm

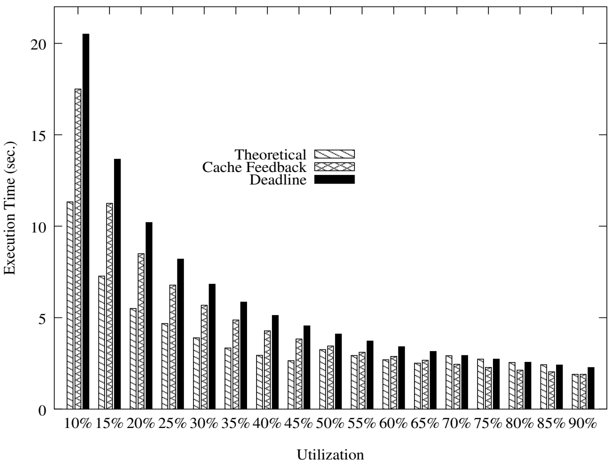

Figure 7. Execution times of dcache-miss.

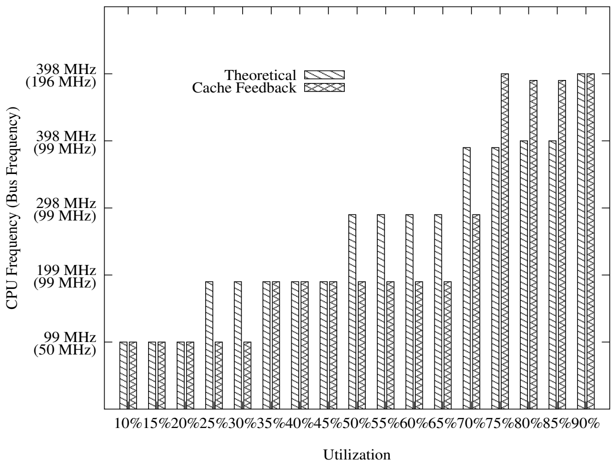

Figure 8. CPU/bus frequencies of dcache-miss.

- The 12 pre-calculated frequency matrices occupy 1,200 bytes of memory. The additional overhead of selecting the appropriate matrix adds less than 75 lines of code using only 32-bit integer arithmetic and only one 32-bit integer division to calculate the ratio of cache misses to instructions.
- The execution time of the CPU scheduler was measured to range from 10 to 150μs. The addition of the cache feedback loop added an additional 1 to 5μs overhead. These numbers vary greatly, since scheduler execution time is dependent on both the clock frequency and the number of processes in the run queue.

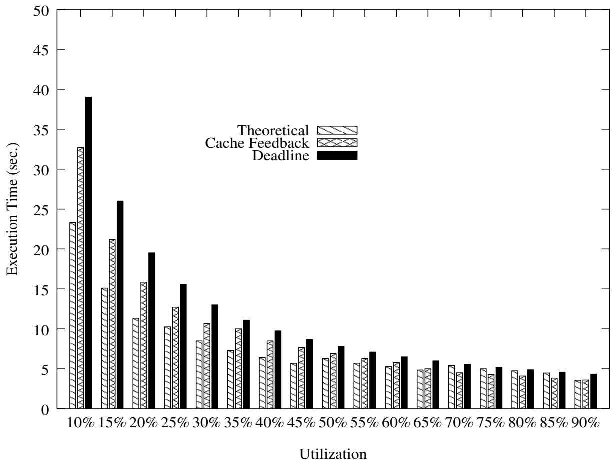

Figure 9. Execution times of gzip.

Evaluation. To test the effectiveness of the cache feedback loop, six different programs are run under various service constraints. Of the six test programs, two of the programs chosen are special test programs, designed to generate data cache hits and misses, respectively. The other four programs, madplay, djpeg, cjpeg, and gzip, are chosen to be representative of common memory-bound applications or real-time multimedia applications.

Test Program	MAR (%)
dcache-hit	0.05
cjpeg	1.26
madplay	3.03
djpeg	3.39
gzip	4.06
dcache-miss	11.13

Table 3. MARs of test programs.

For the dcache-hit and cjpeg, test cases with low MARs (expressed in percentage of instructions executed in Table 3), there is little difference between the results of the theoretical frequency scaling and the feedback loop. This is to be expected, since the single scaling factors used by the theoretical algorithm were calculated using a process with a low MAR.

On processes with a higher MAR, such as dcache-miss, gzip, djpeg, and madplay, the performance exhibits the S-shaped curve in Figure 1 as frequency is scaled. In these cases, the feedback loop helps conserve energy at lower utilizations and also helps avoid missed deadlines at higher utilizations. Figures 7 through 12 show the measured results for dcache-miss, gzip, madplay, and djpeg.

Figures 7, 9, 11, and 13 show the execution times of our

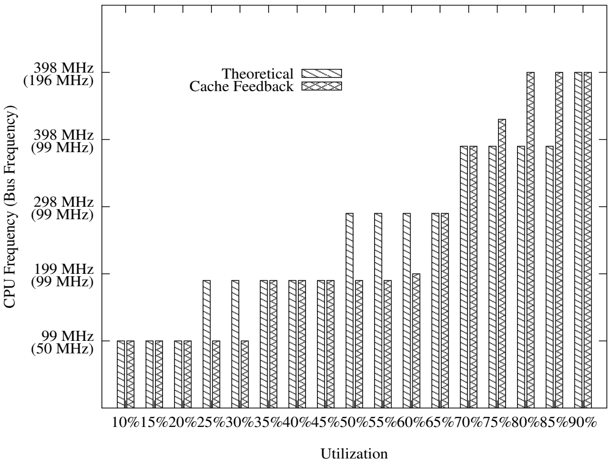

Figure 10. CPU/bus frequencies of gzip.

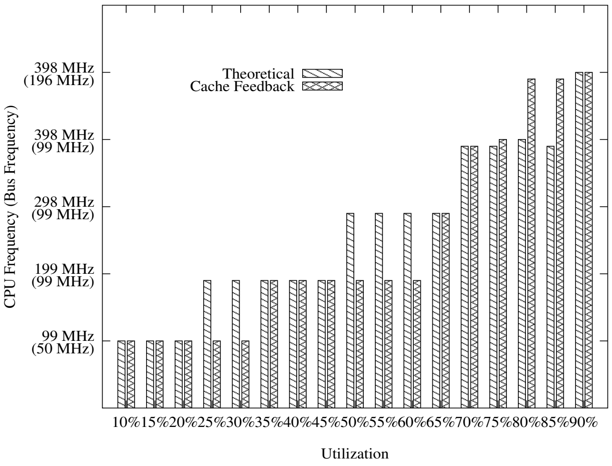

Figure 12. CPU/bus frequencies of madplay.

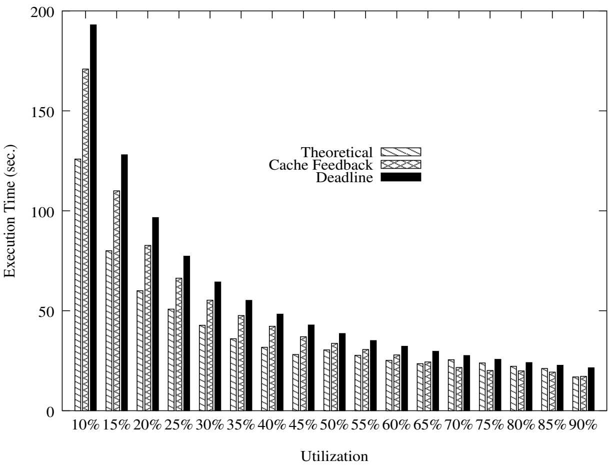

Figure 11. Execution times of madplay.

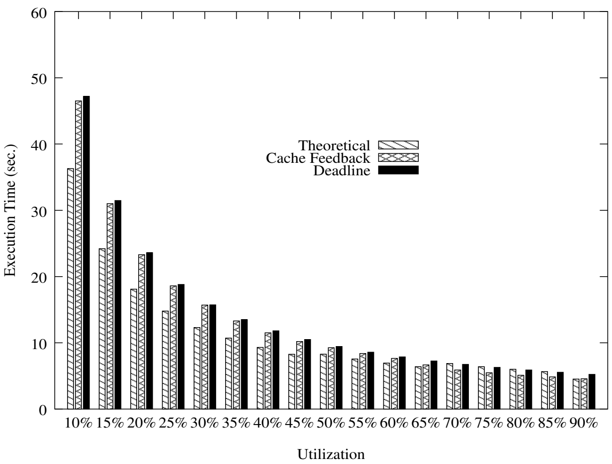

Figure 13. Execution times of djpeg.

test cases. In most cases, the execution times of both the theoretical algorithm and the feedback loop are less than the deadline, indicating that the process met its deadline. Remember that in the scheduler used in this paper, the end of the current period for each task is considered the task's current deadline. For example, in the case of dcache-miss (Figure 7), the feedback loop is able to save more energy in 7 cases, compared to higher energy consumptions in 3 cases, and equal energy consumptions in another 7 cases.

Figures 8, 10, 12, and 10 show the average frequencies of the same applications. In many cases, the feedback loop reduces energy consumption by selecting a lower frequency than the theoretical algorithm. In these cases, a bus speed of 99 MHz is necessary to support the memory references. Since the MAR of the test processes is high, the lowest CPU speed supporting a 99 MHz bus will meet the deadlines in most of the test cases. By detecting the high MAR, the feedback loop selects the lowest CPU frequency supporting the necessary bus speed, resulting in a lower energy consumption over the theoretical algorithm.

However, in a few test cases, the feedback loop is overaggressive when minimizing clock frequencies, causing it to perform worse than the theoretical algorithm. In other test cases, both the feedback loop and the theoretical algorithm miss their deadlines. In these cases, outside factors such as I/O response time and context switching likely increase the execution time of the process more than expected, resulting in a missed deadline. In further research, we intend to implement a multi-resource feedback algorithm, measuring and taking these factors into consideration when choosing optimal frequencies.

Table 4 summarizes the results of the test cases. In the majority of cases, there is no difference between the execution under the theoretical algorithm and the feedback loop.

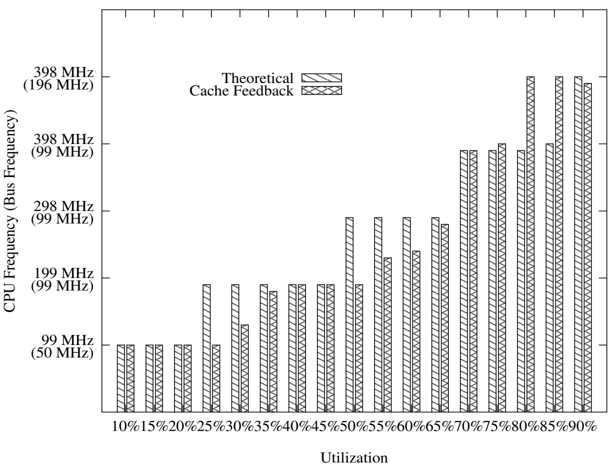

Figure 14. CPU/bus frequencies of djpeg.

This is due to the XScale PXA255's limited number of frequencies. Since there are only five frequency combinations available, the theoretical algorithm usually picks the optimum frequency. However, particularly at high utilizations and/or high memory access rates, the feedback loop is more accurate in the determination of the optimum frequency.

Out of the 102 test cases, the feedback loop results in fewer missed deadlines in 6% of the cases tested. In another 27% of the cases, the feedback loop results in a lower operating frequency. Overall, these savings lead to an average frequency 8.1% lower than the theoretical DVFS algorithm.

Finally, to show that our approach achieves good results when multiple applications compete for the CPU, we run one CPU-bound task (gzip) and one memory-bound task (dcache-miss) simultaneously. Here, the feedback-based approach meets all task deadlines for all CPU utilizations (10%-90%), while the theoretical approach misses the deadline in 22% of the test cases. In addition, the feedback-based approach manages to achieve a lower energy consumption in 44% of the test cases.

5. Summary and Conclusion

Dynamic voltage and frequency scaling allows for a reduction in energy consumption and in heat dissipation for mobile devices. However, most DVFS algorithms are not well-suited to real-time applications, since they will either miss deadlines or waste energy if the execution time of the process is miscalculated. Utilizing the processor's performance counters to monitor memory accesses, we propose to better predict process execution time using a feedback loop. Here, the DVFS approach considers the recent number of cache misses, deriving the overheads for accessing memory, and thereby better predicting an application's runtime, resulting in more precise voltage and frequency computations.

When tested with six applications under various utilizations, the cache feedback method results in an energy savings in 27% of the cases tested. At high utilizations and high memory access rates, the feedback loop raised the CPU frequency, avoiding missed deadlines in 6% of the cases tested.

In future research, we plan to monitor additional resources to use in our feedback loop, including I/O requests and context switching overheads. We intend to more accurately predict process execution times by incorporating more variables in our calculations. Further, our ongoing work addresses the linking of DVFS with the energy management techniques found on other resources, e.g., the sleep modes of I/O devices.

References

[1] S. Agrawal and S. Singh. An Experimental Study of TCP's Energy Consumption over a Wireless Link. In *Proc. of the 4th European Personal Mobile Communications Conference*, February 2001.

[2] T. Burd and R. Brodersen. Energy Efficient CMOS Microprocessor Design. In *Proceedings of the 28th Annual Hawaii International Conference on System Sciences. Volume 1: Architecture*, pages 288–297, January 1995.

[3] S. Chandra and A. Vahdat. Application-Specific Network Management for Energy-Aware Streaming of Popular Multimedia Formats. In *Proc. of the USENIX Annual Technical Conference*, 2002.

[4] K. Choi, R. Soma, and M. Pedram. Dynamic Voltage and Frequency Scaling based on Workload Decomposition. In *Proc. of the International Symposium on Low Power Electronics and Design*, August 2004.

[5] D. Helmbold, D. Long, and B. Sherrod. A Dynamic Disk Spin-down Technique for Mobile Computing. In *Proc. of the Intl. Conference on Mobile Computing and Networking*, 1996.

[6] C. Hsu and U. Kremer. Compiler-Directed Dynamic Voltage Scaling for Memory-Bound Applications. In *Proceedings of the Workshop on Power-Aware Computer Systems, in conjunction with ASPLOS-IX*, November 2000.

[7] W. Kim, D. Shin, H. Yun, J. Kim, and S. Min. Performance Comparison of Dynamic Voltage Scaling Algorithms for Hard Real-Time Systems. In *Proc. of the 8th IEEE Real-Time and Embedded Technology and Applications Symposium*, September 2002.

[8] J. Lorch and A. Smith. Improving Dynamic Voltage Scaling Algorithms with PACE. In *Proc. of the ACM SIGMETRICS Conference*, 2001.

[9] D. Marculescu. On the Use of Microarchitecture-Driven Dynamic Voltage Scaling. In *Proceedings of the Workshop on Complexity-Efective Design, in conjunction with ISCA-27*, June 2000.

	Utilization (%)																
Test Program	10	15	20	25	30	35	40	45	50	55	60	65	70	75	80	85	90
dcache-miss	-	-	-	$	$	-	-	-	$	$	$	$	$	+	-	+	-
gzip	-	-	-	$	$	-	-	-	$	$	$	-	-	-	#	#	-
madplay	-	-	-	$	$	-	-	-	$	$	$	-	-	-	#	#	-
djpeg	-	-	-	$	$	-	-	-	$	$	$	-	+	+	+	+	-
cjpeg	-	-	-	$	$	-	-	-	$	$	$	-	-	-	-	#	-
dcache-hit	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

Legend:

$ Both algorithms meet the deadline, but the feedback loop conserved energy by running at a lower frequency than the theoretical algorithm.

Both algorithms meet the deadline, but the feedback loop conserved less energy than the theoretical algorithm.

+ The theoretical algorithm misses its deadline, but the feedback loop avoids a missed deadline by raising the frequency.

- Both the theoretical algorithm and feedback loop execute at the same frequency.

Table 4. Test processes.

[10] P. Mejia-Alvarez, E. Levner, and D. Mosse. Power-Optimized Scheduling Server for Real-Time Tasks. In *Proc. of the 8th IEEE Real-Time and Embedded Technology and Applications Symposium*, September 2002.

[11] J. Monks, V. Bharghavan, and W. Hwu. A Power Controlled Multiple Access Protocol for Wireless Packet Networks. In *Proc. of IEEE Infocom*, April 2001.

[12] J. Nieh, J. Hanko, J. Northcutt, and G. Wall. SVR4 UNIX Scheduler Unacceptable for Multimedia Applications. In *Proc. of the 4th Int. Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV '93), Lancaster, UK*, November 1993.

[13] P. Pillai and K. Shin. Real-Time Dynamic Voltage Scaling for Low-Power Embedded Operating Systems. In *Proc. of the 18th ACM Symposium on Operating Systems Principles*, 2001.

[14] P. Pillai and K. G. Shin. Real-time Dynamic Voltage Scaling for Low-power Embedded Operating Systems. In *Proc. of the 18th Symposium on Operating Systems Principles*, October 2001.

[15] C. Poellabauer and K. Schwan. Power-Aware Video Decoding using Real-Time Event Handlers. In *Proc. of the 5th International Workshop on Wireless Mobile Multimedia, Atlanta, GA*, September 2002.

[16] C. Poellabauer and K. Schwan. Energy-Aware Media Transcoding in Wireless Systems. In *Proc. of the Second IEEE Intl. Conference on Pervasive Computing and Communications (PerCom 2004)*, March 2004.

[17] C. Poellabauer and K. Schwan. Energy-Aware Traffic Shaping for Wireless Real-Time Applications. In *Proc. of the IEEE Real-Time and Embedded Technology and Applications Symposium*, May 2004.

[18] D. Qiao, S. Choi, A. Jain, and K. Shin. MiSer: An Optimal Low-Energy Transmission Strategy for IEEE 801.11a/h. In *Proc. of the ACM/IEEE Intl. Conference on Mobile Computing and Networking*, September 2003.

[19] S. Saewong and R. Rajkumar. Practical Voltage-Scaling for Fixed-Priority RT-Systems. In *Proc. of the 9th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, May 2003.

[20] K. Seth, A. Anantaraman, F. Mueller, and E. Rotenberg. FAST: Frequency-Aware Static Timing Analysis. In *Proc. of the Real-Time Systems Symposium*, December 2003.

[21] T. Simunic, L. Benini, A. Acquaviva, P. Glynn, and G. D. Micheli. Dynamic Voltage Scaling and Power Management for Portable Systems. In *Proc. of Design Automation Conference*, 2001.

[22] A. Varma, B. Ganesh, M. Sen, S. R. Choudhury, L. Srinivasan, and B. Jacob. A Control-Theoretic Approach to Dynamic Voltage Scheduling. In *Proc. of the International Conference on Compilers, Architectures, and Synthesis for Embedded Systems (CASES)*, October 2003.

[23] Z. Wang, D. Burger, K. S. McKinley, S. K. Reinhardt, and C. C. Weems. Guided Region Prefetching: A Cooperative Hardware/Software Approach. *ACM SIGARCH Computer Architecture News*, 2003.

[24] M. Weiser, B. Welch, A. Demers, and S. Shenker. Scheduling for Reduced CPU Energy. In *Proc. of the 1st Symposium on Operating Systems Design and Implementation (OSDI)*, 1994.

[25] A. Weissel and F. Bellosa. Process Cruise Control. In *Proc. of the International Conference on Compilers, Architectures and Synthesis for Embedded Systems*, October 2002.

[26] Y. Zhu and F. Mueller. Feedback EDF Scheduling Exploiting Dynamic Voltage Scaling. In *Proc. of the IEEE Real-Time and Embedded Technology and Applications Symposium*, May 2004.