Voltage Scheduling in the lpARM Microprocessor System

Trevor Pering, Thomas Burd, and Robert Brodersen
Berkeley Wireless Research Center
University of California, Berkeley
2108 Allston Way, Berkeley, CA 94704
+1-510-666-3100
{pering, burd, rb}@eecs.berkeley.edu

Abstract
Microprocessors represent a significant portion of the energy consumed in portable electronic devices. *Dynamic Voltage Scaling (DVS)* allows a device to reduce energy consumption by lowering its processor speed at run-time, allowing a corresponding reduction in processor voltage and energy. A *voltage scheduler* determines the appropriate operating voltage by analyzing application constraints and requirements. A complete software implementation, including both applications and the underlying operating system, shows that DVS is effective at reducing the energy consumed without requiring extensive software modification.

Keywords
Low-power, energy-efficient, RTOS, operating systems.

1. INTRODUCTION
The importance of energy-efficient microprocessors is increasing along with the popularity of portable electronic devices, such as laptops and PDAs, which devote a significant portion of their power budgets to general purpose processing. The energy consumed by a microprocessor system can be reduced by dynamically adjusting the operating voltage using a technique called *Dynamic Voltage Scaling (DVS)*. Lowering the voltage also requires a reduction in the clock frequency, exposing a fundamental energy/speed trade-off.

Dynamically adjusting the voltage and clock frequency at run-time is the responsibility of the *voltage scheduler,* an entity which runs as part of the operating system. Application requirements, in terms of execution cycles and completion deadline, are combined to form an estimate of the optimal operating speed. Since processor speed is a global resource, the operating system is responsible for combining multiple application requirements to determine the appropriate speed and voltage.

This paper describes the simulation of a thread-based voltage scheduler designed to run on the DVS-capable Low-Power ARM (lpARM) processor. The lpARM processor itself, described in detail elsewhere [3], is an implementation of the ARM8 architecture using a 0.6 µm CMOS process. The effectiveness of DVS scheduling varies depending on the application, with energy being reduced to less than 20% of fixed-voltage in some cases.

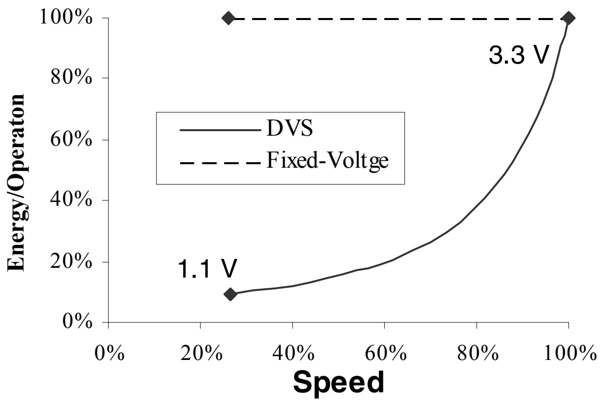

Figure 1: Energy/Speed Trade-Off

1.1 Voltage Scaling Fundamentals
The energy, E, consumed by a CMOS microprocessor can be calculated using the equation $E \propto N_{ops}CV^2$, where N_{ops} is the number of operations executed, C is the capacitance per operation, and V is the operating supply voltage [5]. Architectural energy-reduction techniques [14], such as clock-gating, focus on reducing C while traditional compiler optimizations minimize N_{ops}. Our technique, DVS, focuses on reducing V.

Static-voltage reduction is a well-established energy-reduction technique since the squared relationship between energy and voltage causes even small reductions in the voltage to affect relatively large reductions in energy. Reducing the operating voltage also reduces the maximum speed at which a device can operate, f_{max}, approximated by the relationship $f_{max} \propto (V - c)/V$, where c is a process-dependent constant.

These two relationships expose the trade-off between energy and speed which can be exploited by varying the voltage, graphically depicted in Figure 1. Reducing the clock speed of a device without simultaneously lowering the operating voltage does not reduce the energy consumed by a given task. Given a fixed number of instructions that the processor must execute, it is imperative that the operating voltage also be reduced [2].

The simulated lpARM processor is based on the ARM8 core [1] and designed to operate between 1.1V and 3.3V, resulting in speeds between 10 MHz and 100 MHz, respectively. While executing, the core and cache together consume between 1.8 mW and

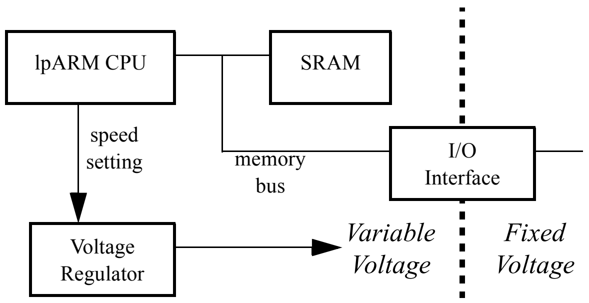

Figure 2: lpARM System Block Diagram

220 mW, depending on the processor speed. When idle, the processor consumes less than 500 µW, with a one cycle start-up cost.

Clock frequency transitions take approximately 25 µs for a complete 10 MHz to 100 MHz transition: about 1250 cycles, on the order of a context-switch. The system can continue operation while the speed and voltage are changing; it is assumed that there is no energy cost associated with changing the operating voltage. The external processor bus operates at half the frequency of the processor core.

1.2 System Overview

The lpARM CPU directly controls the operating voltage of the processor subsystem, as shown in Figure 2. The voltage of the entire microprocessor subsystem is scaled together, preventing one component from dominating while the others operate at a reduced voltage. The I/O interface provides a level-converting bridge to interact with the I/O subsystems, which are implemented using standard non-DVS components. The measurements and analysis in this paper concentrate on the variable voltage portion only.

The operating system is based on a standard run-time kernel that has been extended with a *voltage scheduler,* responsible for examining the state of the system and communicating the desired processor speed to the hardware. The processing requirements of individual threads, overall system loading, and resource utilization all effect the scheduling decision. The effectiveness of voltage scheduling is shown by comparing the energy consumed by three different benchmark programs simulated with and without voltage scaling.

2. VOLTAGE SCHEDULING

The *voltage scheduler* is responsible for determining the speed and voltage at which the processor runs. System-level information, such as recent processor utilization and predicted future behavior, is combined with expectations of and estimates provided by individual threads to determine the desired operating speed. The voltage scheduler is reevaluated each time a task is added or removed from the system, and periodically during task execution.

The voltage scaling hardware/software interface is defined in terms of speed, not voltage. The software specifies the target operating speed to the hardware, which is then responsible for delivering the requested performance. The software is not aware of the operating voltage used by the hardware; the term *voltage scheduling* is used rather than *speed scheduling* to remind us that the voltage is being adjusted along with the clock. Throughout this paper, a change in clock frequency implies an associated change in voltage, unless otherwise noted.

Applications can specify a deadline, which is an indication of when its current frame should be completed. The voltage scheduler runs independently of the conventional *temporal scheduler*, responsible for deciding which task runs when. An earliest-deadline-first (EDF) policy is used for temporal scheduling, which is optimal for fixed-speed systems in the sense that all deadlines will be met if possible [11]. However, since the processor is often running at a reduced speed, it is quite probable that a task will miss its stated deadline. Tasks in the lpARM system, therefore, must be designed to gracefully handle a missed deadlines, nominally accomplished by buffering one frame of output data.

2.1 Hardware Interface

The voltage scheduler requires the support of the processor hardware to effectively accomplish its task. The support mechanisms can often be found in existing embedded microprocessors, although they are not ubiquitous. The speed control register is unique in DVS systems in that it also adjusts the operating voltage. Four types of support are used:

- **Speed-Control Register -** sets the target operating speed of the processor and changes the system operating voltage. A read from this register returns the current physical processor speed, which can be used to test system operation.
- **Processor Cycle Counters -** measures the number of cycles the processor spends executing, stalled, waiting for memory, etc... The count used includes all stall and execution cycles, incorporating the 'work' performed by the memory system.
- **Wall-clock Time -** maintains a consistent measure of time, independent of the processor speed and sleep mode.
- **System Sleep Control -** controls the system's low-power shutdown mode, which causes the processor to consume virtually no power, used where there are no active threads available. The system is awakened by either an external interrupt or internal wall-clock alarm timer.

2.2 Scheduling Framework

Three application-level execution models are supported by the system: high-priority, rate-based, and deadline-based. Internally, high-priority and rate-based tasks are represented as deadline-based with automatically generated deadlines. Most system-level threads run at high-priority, while major applications typically specify a deadline-based model.

High-priority tasks are defined in the lpARM system to be short, sporadic tasks which require a quick response time but do not present a significant system load. They execute before all other threads in the system (high-priority tasks themselves are ordered and execute highest-priority-first). High-priority tasks are ignored by the voltage scheduler and therefore do not directly effect the processor speed: they would merely complete before the system could increase speed, since and are short in duration.

Rate-based threads sustain a predictable rate of execution, supplied by the application, without an explicit deadline. Jobs such as compilation, which should finish in "a reasonable amount of time," fall into this category. The deadline for these tasks is automatically calculated by the internal scheduler and update dynamically as the task executes. Rate-based threads should be scheduled so that they appear to be executing on a single-threaded system running at their specified speed: two 10 MHz sustained rate threads will time-share a system run at 20 MHz.

The deadline-based model divides applications into execution units called *frames*, each of which has a measure of *work* which should be completed by its *deadline* [4]. A frame is an application-specific unit, such as a video frame, and work is defined in terms of processor cycles. The desired rate of processor execution in a predictable single-threaded system is easily determined by the equation *speed = work/deadline*; it is the job of the voltage scheduler to determine the processor speed given a set of multiple tasks and estimated parameters.

Slowing the processor down will cause many application deadlines to be missed; therefore, applications are required to handle over-deadline frames gracefully, typically by an added level of output buffering. Allowing missed deadlines enables the voltage scheduler to accommodate the average-case workload instead of the worst-case, reducing total energy consumed. Allowing *multiple* missed frames, instead of just one, would further increase the flexibility afforded the voltage scheduler; however, exploring this trade-off is beyond the scope of this paper.

2.3 Scheduling Algorithm

The voltage scheduling algorithm used assumes all tasks are sporadic and calculates the minimum speed necessary to complete all tasks assuming they are all currently runnable. Given that threads are sorted in EDF order, this speed can be found by:

$$speed = \underset{\forall(i \le n)}{MAX}\left(\frac{\sum_{j \le i} work_j}{deadline_i - currenttime}\right)$$

This algorithm can be implemented in *O(n)* time where *n* is the number of scheduled threads; the sorted list of threads can be maintained incrementally as task deadlines are updated.

Including non-runnable tasks in the voltage scheduling reserves future execution cycles for them, preventing the system from underestimating processing requirements. It is possible for the scheduler to overestimate processing required if a non-runnable task presents a very large future workload; however, this case is rare and does not significantly effect performance.

The processor speed is reevaluated each time a thread is added or removed from the system, or when a deadline is reached. Therefore, the calculated target speed needs only to be valid until the next task deadline. Figure 3 shows an example voltage schedule with three tasks. In this example, the processor will be set to run at 48% processor speed until Task A completes, at which time the schedule will be reevaluated. Other speeds shown are intermediate results only that are not necessarily used by the system (because of reevaluation). Note that the voltage schedule is independent of the task's start times; for example Task C's start time is considerably after Task B's deadline without effecting the schedule. This relaxation is valid because the algorithm is periodically reevaluated; in any case, however, a task will not start executing until its start time is reached.

	StartTime	Deadline	Workload	Σ Workload	Σ /Deadline
Task A	0	3	144	144	48
Task B	2	5	74	218	44.4
Task C	7	9	88	306	34

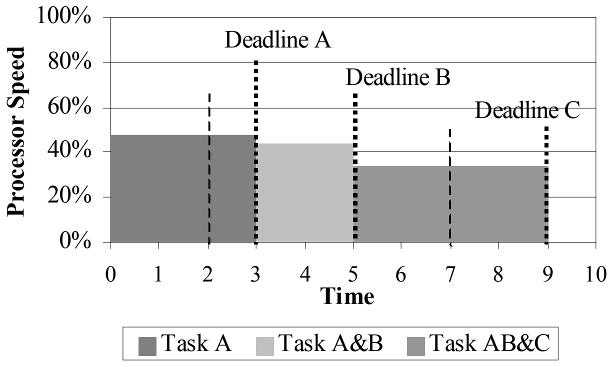

Figure 3: Example Voltage Schedule

2.4 Workloads & Schedule Smoothing

The estimated workload, used by the scheduling algorithm, is determined empirically at run-time using a exponential moving average, reevaluated each time an application frame is completed: $W_{new} = (W_{old}*k + W_{last})/(k + 1)$, where W_{new} is the new estimate, W_{old} is the old estimate, and is W_{last} the recently completed frame's cycle count. Outlier samples, which correspond to frames requiring more work than could possibly be completed before deadline, are assumed to be initialization frames and ignored. An application may either explicitly specify an initial work estimate or use the computation required for its first frame to initialize the sequence. Rate-based applications specify their workload as sustained rate of execution, defined in terms of processor speed (equivalent to *work/time*). High-priority threads do not need to specify a workload estimate because they are not handled by the voltage scheduler.

Applications with high frame-to-frame computation variance could potentially consume more energy than well-behaved applications because the system will have difficulty accurately predicting the next frame's workload: one frame may execute over deadline, while the next would be under deadline. To mitigate this problem, the voltage scheduler schedules threads aiming to complete twice the amount of work in twice the deadline, effectively aggregating the variance of adjacent, a technique called *schedule smoothing*.

3. EXPERIMENTAL RESULTS

The effectiveness of voltage scheduling is evaluated by comparing the results of a voltage-scheduled system against those of fixed-voltage systems, shown in Figure 4. For each benchmark, the energy is normalized to the energy consumed while running that benchmark at maximum processor speed. "DVS" is the energy consumed when applying the voltage scheduling algorithm of the previous section. "Nominal" represents an application-specific reasonable fixed-speed for the different benchmarks, described below in more detail.

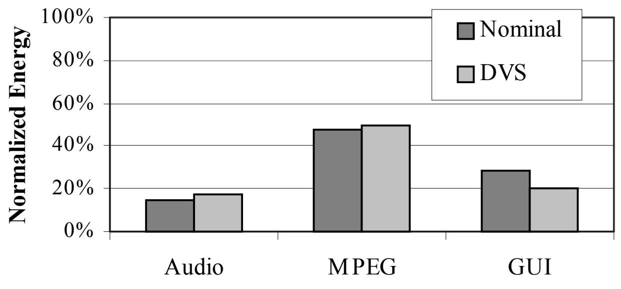

Figure 4: Basic Voltage Scheduling Results

The energy savings afforded by DVS are extremely application dependent. For some applications, energy is reduced to under 20% of maximum, while in other situations DVS does not significantly reduce energy consumption at all. Several different aspects of voltage scheduling, supported by a description of the benchmarks themselves, are discussed below.

3.1 Benchmarks

Two common multimedia tasks and a graphical user interface are used to evaluate voltage scheduling:

- **Audio -** IDEA decryption of a 10-second 11 KHz mono audio stream, divided into 1 kB frames with a 93 ms deadline.
- **MPEG -** MPEG-2 decoding of an 80-frame 192x144 video at 8 frame/sec.
- **GUI -** A simple address-book user interface allowing simple searching, selection, and database selection. 432 frames are processed, each defined as a user triggered event, such as pen-down, which ends when the corresponding action has been completed. A deadline of 50 ms is used for each frame, roughly modeled on the human visual perception time.

The data for the Audio and MPEG benchmarks is brought into the system by low-lever interrupt handlers and placed in system FIFOs. The main application thread pulls data from the FIFO, performs the necessary computation, and transfers the result to an intermediate de-jitter buffer. A separate thread is then used to transfer the data to the output device before the next frame's deadline. This output buffering is necessitated by the need to tolerate missed deadlines, caused by the application of voltage scheduling; therefore, the stated deadline only represents a *suggested* completion time. These benchmarks will fail when any single frame exceeds *twice* its deadline, when the output thread would be unable to output its data in a timely manner. "Nominal" for Audio and MPEG represents the minimum fixed-speed at which all frames are output correctly.

User input, from a screen pointing device, is the primary mode of input for the GUI application. A screen event, such as a pen-down event, passes through the operating and windowing systems and then is passed to the application. A 50 ms deadline representing human visual perception time is used for GUI events: screen updates faster than this will not be noticed by the user [6][13]. Most GUI events, such as simple button updates, can easily complete before this deadline. Some events, however, are extremely computationally expensive; opening a new dialog window, for example, takes 260 ms when the processor is running at max-speed. There is no 'correct' speed for these events: they represent a user-dependent energy/speed trade-off. For the long-running events, execution defaults to a rate-based 40 MHz, opting for a balance between energy-efficiency and low-latency. "Nominal" for GUI represents a fixed 40 MHz speed for all events, resulting in the same perceived quality-of-service for the long-running events; DVS is able to beat this nominal setting by reducing the speed for short frames without impacting the performance of long-running frames.

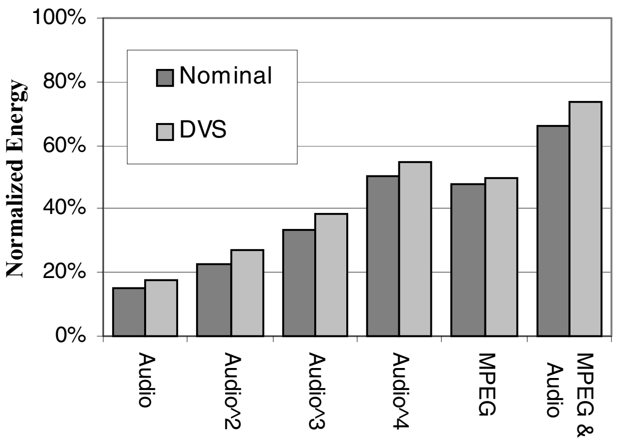

Figure 6: Multiple Benchmark Execution

3.2 Event Distributions

Frame-to-frame computation histograms, shown in Figure 5, detail how the individual benchmarks impact, and are effected by, voltage scheduling. An application's average execution requirement, represented by the max-speed distribution, is the primary factor in determining energy consumption. The processing required for Audio frames is quite low compared to the processing required for MPEG. Correspondingly, the Audio max-speed distribution is farther to the left than MPEG's. Less works simply means that the benchmark executes fewer instructions and can be run at a lower voltage

The visual effect of voltage scheduling is to adjust the distribution to center around the 100% deadline mark. For the computationally-intensive Audio and MPEG benchmarks, all frames are equally effected by voltage scaling, a 50% reduction in processor speed increases all frame completion times by 50%. However, close examination of the GUI frame distributions indicates that many long-running frames are relatively unaffected by changes in processor speed. A large component of these frames is *hard idle time*: idle time which is spent waiting for external events and is not computation bound. The simplest example of hard idle time is waiting for an external network packet, which takes the same amount of time to arrive regardless of the processor speed.

3.3 Simultaneous Benchmark Execution

Figure 6 shows the execution of concurrent benchmarks. For all of these combinations, the voltage scheduler behaves as expected: adapting the processor speed to accommodate the extra workload. Since data is normalized to a max-speed processor, the increased number of executed instructions has no effect on the normalized energy consumed; the increase is due solely to an increase in the processor speed/voltage. The *actual* energy for running Audio^2,

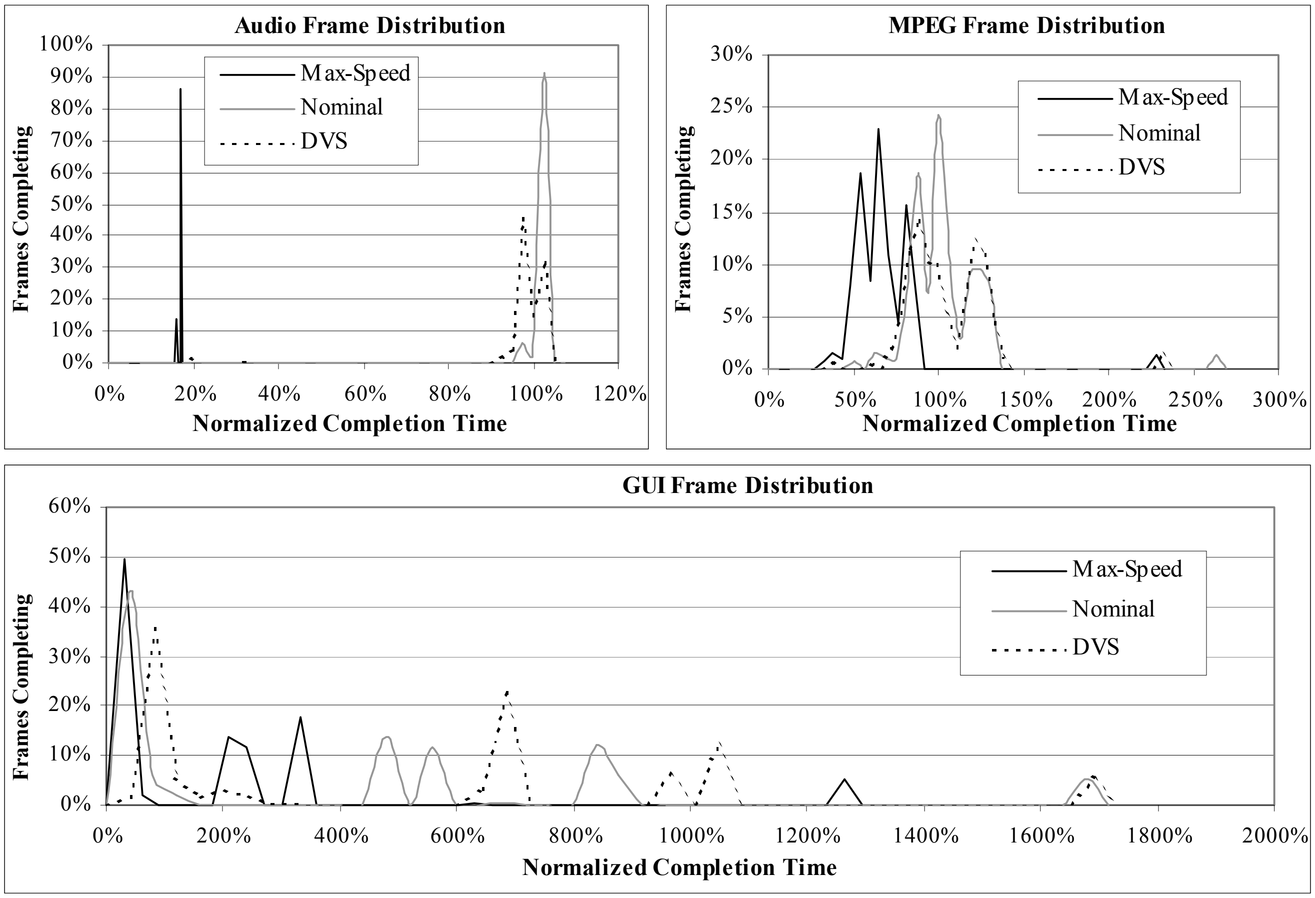

Figure 5: Frame Completion Histograms

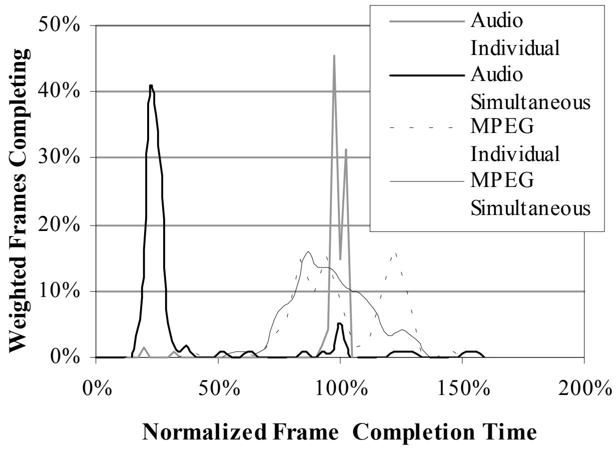

Figure 7: Multiple Benchmark

however, would be greater than twice the energy consumed by Audio. Running two MPEG decodes simultaneously is not possible because it would overload the processor.

Figure 7 shows the frame completion time histograms for the Audio and MPEG benchmarks running simultaneously. As the processor loading is increased by adding more threads, the completion times of the Audio frames are decreased dramatically. This shift is caused by the combination of the increased processor speed, due to executing multiple benchmarks, and the shorter Audio frame deadlines, which are typically scheduled first due to the EDF temporal scheduling policy. Therefore, although executing multiple benchmarks will increase the energy consumed by the system, the real-time behavior of the system, as measured by over-deadline frames, is improved. (Although for the lpARM system this has little benefit because the applications can tolerate delayed frames.)

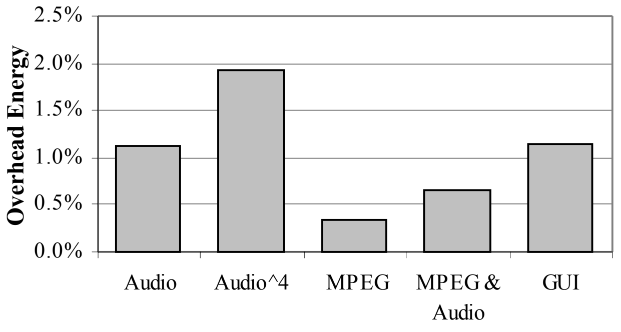

Figure 8: Scheduling Overhead

3.4 Scheduling Overhead

Figure 8 shows the scheduling overhead for several different benchmark combinations, measured by the percentage of energy going towards the execution of the scheduler thread. In all cases, the overhead is quite small: less than 2%. The voltage scheduler is triggered around start-times and deadlines: the more computation is required per-frame the smaller the relative impact. Additionally, as the number of tasks increases, the overhead increases due to additional scheduling complexity.

4. RELATED WORK

Dynamic Voltage Scaling for a microprocessor system first appeared in [7][15], which applied the fundamental speed/energy trade-off to a collection of UNIX workstation traces. For these works, the processor speed was scaled based on the global processor utilization of fixed size intervals: no knowledge of individual threads was used. Our earlier work, [12] evaluates these interval-based voltage scheduling algorithms using the same basic environment presented in this paper. Since interval-based scheduling does not understand application processing requirements, e.g. deadlines and computation estimates, it behaves poorly for workloads that consist of more than a single well-behaved task.

[16] presents a theoretical basis for using real-time constraints very similar to those used in this paper and an off-line O($n \log^2 n$) scheduling algorithm which optimally schedules a given set of tasks, given complete and accurate knowledge. Additionally, it gives formal proofs necessary to construct the optimal algorithm. It does not consider work estimation, frame-to-frame execution variance, scheduling of unknown tasks, or analysis of real applications.

[10] presets a theoretical analysis of scheduling a single task assuming the capacitance/cycle not constant. Our work assumes a constant capacitance, which we believe is reasonable for compiler-generated general-purpose applications.

[9] analyzes the theoretical scheduling of a hard real-time DVS system, using worst-case instead of average-case work estimates. They present an independently developed on-line scheduling algorithm that is designed to optimally schedule a set of active tasks (no future start-times). In [8], the same authors analyze the incorporation of a DVS microprocessor into system-on-a-chip synthesis tools.

5. CONCLUSION

Dynamic Voltage Scaling is an extremely effective technique for reducing the energy consumed by a portable microprocessor system. Our results show that energy can be reduced by up to 80%, depending on the application workload.

The analysis of a complete environment, provided by our simulator, shows that DVS can be efficiently integrated into existing operating systems without extensive modification. Since processor speed is independent of temporal scheduling, voltage scheduling can be implemented on top of an existing operating system with little modification. Additionally, standard real-time constraints provided by start-times and deadlines provide an convenient framework, making existing applications easy to integrate. Relaxing the deadline constraint and allowing application frames to complete after their deadlines enables a scheduler to absorb the effects of high frame-to-frame application variance, which could otherwise increase energy consumption.

Acknowledgments

This work was funded by a grant from DARPA and made possible by cooperation by ARM Ltd. The authors would also like to thank Eric Anderson for his many helpful insights and conversations.

References

[1] *ARM 8 Data-Sheet*, Document Number ARM DDI0080C, Advanced RISC Machines Ltd, July 1996.

[2] T. Burd and R. W. Brodersen, “Energy efficient CMOS microprocessor design,” *Proc. 28th Hawaii Int’l Conf. on System Sciences,* Vol.1, pp. 288-297, Jan. 1995.

[3] T. Burd, T. Pering, A. Stratakos, R. Brodersen, “A Dynamic Voltage-Scaled Microprocessor System”, 2000 IEEE International Solid-State Circuits Conference Digest of Technical Papers, San Francisco, Feb. 2000.

[4] A. Burns and A. Wellings, *Real-Time Systems and Programming Languages, second edition,* Addison-Wesley, 1997.

[5] A. Chandrakasan, S. Sheng, R. W. Brodersen, “Low-power CMOS digital design,” *IEEE Journal of Solid-State Circuits*, Vol. 27, pp. 473-484, Ap. 1992

[6] Y. Endo, Z. Wang, J. B. Chen, and M. Seltzer, “Using Latency to Evaluate Interactive System Performance,” *Proc. 2nd Symp. on Operating Systems Design and Implementation*, Nov. 1996.

[7] K. Govil, E. Chan, H. Wasserman, “Comparing Algorithms for Dynamic Speed-Setting of a Low-Power CPU”, *Proc. 1st Int’l Conference on Mobile Computing and Networking,* Nov 1995.

[8] I. Hong, D. Kirovski, G. Qu, M. Potkonjak, and M. Srivastava. Power optimization of variable voltage core-based systems. In *Design Automation Conference*, 1998.

[9] I. Hong, M. Potkonjak, M. Srivastava. On-Line Scheduling of Hard Real-Time Tasks on variable Voltage Processor. IEEE/ACM International Conference on Computer-Aided Design, Nov 1998.

[10] T. Ishihara, H. Yasuura, “Voltage Scheduling Problem for Dynamically Variable Voltage Processors,” *Proc. 1998 Int’l Symp. on Low Power Electronics and Design.*

[11] C. Liu and J. Layland. Scheduling algorithms for multiprogramming in a hard real-time environment. *CACM 20*, 1973.

[12] T. Pering, T. Burd, and R. W. Brodersen, “The Simulation and Evaluation of Dynamic Voltage Scaling Algorithms,” *Proc. 1998 Int’l Symp. on Low Power Electronics Design*.

[13] B. Shneiderman, *Designing the User Interface*, Addison-Wesley, 1992.

[14] M. Srivastava, A. Chandrakasan, R. Brodersen. Predictive system shutdown and other architectural techniques for energy efficient programmable computation. IEEE Transactions on VLSI Systems, 4(1), 1996.

[15] M. Weiser, B. Welch, A. Demers, and S. Shenker, “Scheduling for reduced CPU energy,” *Proc. 1st Symp. on Operating Systems Design and Implementation*, pp. 13-23, Nov. 1994.

[16] F. Yao, A. Demers, and S. Shenker. A scheduling model for reduced CPU energy. In IEEE Annual Foundations of Computer Science, pages 374-382, 1995.