

# Interactions of Power-aware Memory Systems and Processor Voltage Scaling

Xiaobo Fan   Carla S. Ellis   Alvin R. Lebeck

Department of Computer Science  
Duke University  
Durham, NC 27708 USA  
{xiaobo,carla,alvy}@cs.duke.edu

## Abstract

*Energy consumption is becoming a limiting factor in the development of computer systems for a range of application domains. Since processor performance comes with a high power cost, there has been recent interest in scaling the voltage and clock frequency of the CPU. Dynamic Voltage Scheduling (DVS) is the technique for exploiting the scaling capabilities of the hardware in which an appropriate clock rate and voltage are selected to meet application requirements at the lowest energy cost. Unfortunately, the power and performance contributions of other system components, in particular memory, challenge some of the simple assumptions upon which most of the previous DVS algorithms have been based.*

*We explore interactions between memory and voltage/frequency scaling of the processor. In particular, we consider memory systems that offer their own power management features. We show that power-aware memory is important to fully exploit the potential of DVS but that it does introduce energy tradeoffs that conflict with simplistic scaling models. We argue that memory-based criteria – information that is available in commonly provided hardware counters – are important factors for effective speed-setting in DVS algorithms and we develop a technique to estimate overall energy consumption based on them. We show that frequency scaling influences the design of memory controller policies that transition power-aware memory chips among power states and that the ability to change policies along with frequency may be valuable.*

## 1 Introduction

Energy consumption is becoming a limiting factor in the development of computer systems for a range of application domains – from mobile and embedded devices that depend on batteries to large hosting centers that incur enor-

mous electricity bills. In recognition that the exponential growth in the performance of processors may come at a high power cost, there has been considerable interest in scaling the supply voltage and clock frequency of the CPU. Thus, if application demand does not currently need the highest level of processor performance, a lower power design point can be chosen temporarily. The excitement surrounding voltage/frequency scaling is based on characteristics of the power/performance tradeoffs of CMOS circuits such that the power consumption changes linearly with frequency and quadratically with voltage, yielding potential energy savings for reduced speed/voltage.

Dynamic Voltage Scheduling (DVS) is the technique for exploiting this tradeoff whereby an appropriate clock rate and voltage is determined in response to dynamic application behavior. This involves two issues: (1) predicting future processing needs of the workload and (2) setting a speed (and associated voltage) that should satisfy those performance needs at the lowest energy cost. A number of DVS algorithms have been proposed [27, 22, 21, 12, 9, 7, 24, 8], primarily addressing the prediction issue. Most simulation-based studies of these algorithms have focussed solely on CPU energy consumption, characterized the workload in terms of CPU load, and assumed that performance scales linearly with frequency. Unfortunately, the power and performance contributions of other system components may complicate this simple model.

The importance of considering other system components is supported by the few studies that have been based on actual implementation of DVS algorithms for which measurement results have been disappointing compared to simulation results. This has been attributed to several factors, including inaccuracies in predicting the future computational requirements of real workloads and interactions with other components of the system beyond the CPU, especially memory [9, 7, 17, 18, 19, 24].

In this paper, we explore the nature of these interactions between memory and voltage/frequency scaling of the pro-

cessor. In particular, we consider memory systems that offer their own power management features. To evaluate the interaction between DVS and power-aware memory, we use the PowerAnalyzer simulation infrastructure, a modified version of SimpleScalar [3] that executes ARM binaries and provides detailed power consumption statistics. As a workload, we use an MPEG decoder with a period of 66ms per frame with an input file of 3 different type frames.

Based on our simulation results, this paper makes the following contributions:

- We show that, in order to develop an effective DVS speed-setting strategy, the memory system design must be understood. In a traditional memory design, memory energy so dominates processor energy that the overall impact of DVS may be marginal. Even the simplest memory power management strategy that powers down the DRAM when the processor becomes idle introduces a tradeoff between CPU and memory energy that may negate the energy saving benefits of reducing the CPU frequency/voltage beyond some point.
- We demonstrate that more sophisticated and effective power-aware memory policies may enhance the overall impact of DVS by significantly lowering the power cost of memory relative to the CPU. Thus, we conclude that an appropriate power-aware memory policy can have a complementary effect on DVS.
- Given the energy tradeoffs inherent with a power-aware memory, we argue that the memory access behavior of the workload must be understood in order for the DVS system to predict the energy and performance implications of a particular frequency/voltage setting.
- We develop a technique to estimate overall energy consumption using information available from existing performance counters (i.e., instructions executed, memory references, misses) and show that our estimator is sufficient to capture the general trend in overall energy as frequency changes.
- We show that, if the DVS algorithm must increase the frequency to meet a deadline, the ability to adapt the power-aware memory controller policy for transitioning between different power states is valuable.

The remainder of this paper is organized as follows. The next section discusses background and related work. Section 3 describes our methodology, and Section 4 examines the interactions between DVS and a traditional high power, low latency memory design. We examine the effects of power-aware memory and develop a memory-based estimator of overall energy in Section 5. The influence of DVS on memory controller policy selection is explored in Section 6 and we conclude in Section 7.

## 2 Background and Related Work

### 2.1 Dynamic Voltage Scheduling

Dynamic voltage scheduling has been studied for a wide variety of workloads, including interactive, soft real time, and hard real time applications. Each of these workloads may require a different type of DVS algorithm based on the information available about the tasks, the tolerance for missed deadlines, and the nature of the application behavior. In general, most DVS algorithms divide total execution time into task periods or regular intervals and attempt to slow down computation to just fill the period without missing the deadline or carrying work over into the next interval. There is a requirement to predict the processing demands of future periods, usually from observed past behavior, and use that information to determine the appropriate processor speed and corresponding voltage.

The earliest DVS algorithms fall into the category of interval-based algorithms [27, 22, 9, 8]. These algorithms have little a priori information about the workload and divide time into arbitrary intervals in the sense that the length of the interval has no relationship to the semantics of the tasks involved. The goal of the algorithm is to monitor processor load and to set the clock speed just slow enough to spread the computation over the interval and squeeze out any idle time. By monitoring processor utilization of previous intervals (algorithms differ on the number of past intervals to use for the next decision), the speed is set either up (if CPU load was too high and, possibly, unfinished work remained) or down (if there was too much idle time) for the next interval. How much higher or lower the speed should be set when a change is needed is another decision to be made. Experience with this class of algorithms suggests that they may be effective for well-behaved, regular workloads. However, bursty or irregular behavior causes serious difficulties in the accurate prediction of future needs. The clock speeds tend to oscillate without stabilizing on a good point or else are very slow to react and lag behind needed transitions [9]. There have been attempts to improve the prediction aspect of interval-based DVS algorithms by trying to detect patterns and anticipate bursts [8].

The other distinct category of DVS algorithms integrates scaling into task-oriented, deadline-based, real time schedulers (e.g., earliest deadline first or rate monotonic) [23, 12, 26, 11]. These algorithms assume knowledge about the real time task set, including the worst case execution time (WCET) of each task and the period/deadline. The goal is to determine the speed setting such that computation is spread out over the period while never missing a deadline. These algorithms tend to begin with the determination of a schedule using the WCET, assume performance scales linearly with frequency, and then satisfy schedulability tests. The

algorithms differ in how to deal with online behavior that does not consume the specified WCET. For example, Pillai and Shin [23] propose cycle-conserving algorithms that reduce frequency in response to unused cycles of previous WCETs, revising the speed-setting at each task completion or release.

Recent work that falls somewhere in between the hard real time and the interval-based categories acknowledges the need for more semantic information about the workload to address the prediction issues. Flautner et al [7] derive execution episodes by examining communication patterns. Different types of episodes, classified as interactive, periodic producer, and periodic consumer, suggest different speed-setting policies on a per-task per-episode basis. Lorch and Smith suggest heuristics to detect task periods based on observations of user interface events and I/O activity for use with their PACE speed scheduling algorithm [16]. Pouwelse et al [24] suggest using application-specific information supplied by power-aware applications (e.g., frame type and size for a video decoder). Mosse [20] proposes explicit compiler assistance. These studies provide a rationale for our assuming good predictions for a specific workload.

The speed-setting decision has appeared to be the more straightforward, given good predictions. However recent experimental work [17, 24, 9] has suggested that memory effects should be taken into account. For computations that run to completion, Martin [17, 19] shows there is a lower bound on frequency such that any further slowing degrades the amount of computation that can be performed per battery discharge. For periodic computations, Pouwelse [24] alludes to the problem that the high cost of memory, extended over the whole period, may dominate the overall energy consumption of a system such that even effective DVS of the CPU delivers marginal benefit. This complication of the DVS problem is the focus of our work.

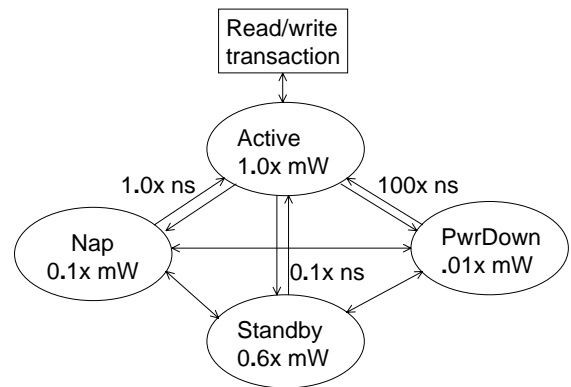
## 2.2 Power-aware Memory

Previous work on power-aware memory systems [13, 4, 5, 6] introduces another complication such that the power consumption of memory varies significantly depending on how effectively the system can utilize a set of power states offered by the hardware. Power-aware memory chips can transition into states which consume less power but introduce additional latency to transition back into the active state in order to be accessed. The lower the power consumption associated with a particular state, the higher the latency to service a new memory request. Figure 1 illustrates a four-state model consisting of *active*, *standby*, *nap*, and *powerdown* states with typical power and latency values as shown in Table 1 (based on RDRAM specifications [25]).

The memory controller can exploit these states by im-

Power State or Transition	Power (mW)	Time (ns)
Active	$P_a = 300$	$t_{acc}=60$
Standby	$P_s = 180$	-
Nap	$P_n = 30$	-
Powerdwn	$P_p = 3$	-
Stby $\rightarrow$ Act	$P_{s \rightarrow a} = 240$	$T_{s \rightarrow a} = +6$
Nap $\rightarrow$ Act	$P_{n \rightarrow a} = 165$	$T_{n \rightarrow a} = +122$
Pdn $\rightarrow$ Act	$P_{p \rightarrow a} = 152$	$T_{p \rightarrow a} = +25,000$

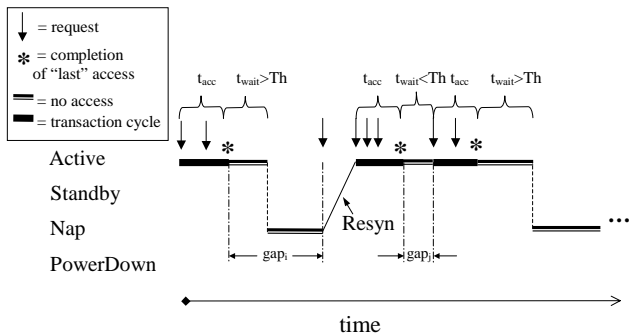
**Table 1. RDRAM Power State and Transition Values:** All accesses incur the 60ns active access time. Additional delay (denoted by the +) is incurred for clock resynchronization.



**Figure 1. RDRAM Power States:** Power costs and transition times are given as relative to the power consumption and access time of active mode.

plementing dynamic power state transition policies that respond to observable memory access patterns. Such policies often are based on periods of idle time (which we refer to as *gaps*) between runs of accesses and threshold values to trigger transitions. Figure 2 shows how a policy that only transitions between *active* and *nap* modes might work. While the memory has outstanding requests, the memory chip stays active. When the idle time gap exceeds a threshold (e.g.,  $gap_i$ ), the chip transitions into *nap* and stays there until the start of the next access (the end of the gap). For gaps shorter than the threshold (e.g.,  $gap_j$ ), the memory remains active.

Operating system page allocation policies that place active pages in the minimum number of DRAM chips fully exploit the capabilities of power-aware memory. Previous studies [13] show that using a sequential first-touch virtual to physical page allocation policy enables unused DRAM



**Figure 2. Memory Access Behavior:** A 2-state control policy illustrating the relationship of gaps, thresholds, and state transitions

chips to enter the powerdown state. Sequential page allocation, by concentrating all memory references to the minimum number of DRAM chips, produces significant energy savings over random page allocation.

Intuitively, we expect that frequency scaling of the processor will affect the timing between memory requests and, thus, the lengths and the number of the gaps seen during execution. This, in turn, should affect the effectiveness of threshold-based transition policies. We explore the impact of such interactions in Section 6.

### 3 Methodology

The primary goal of this paper is to explore memory’s influence on selecting the appropriate frequency/voltage to achieve the lowest energy use while satisfying a known need for a particular level of performance. Therefore, we focus on the factors that affect the speed-scaling decision in meeting those energy/performance goals.

We use a modified version of the PowerAnalyzer [1] simulator from the University of Michigan for our experiments. We modified the simulator to include a detailed RDRAM memory model including the power state transitions described in Section 2. The variable voltage processor we simulate is based on Intel’s XScale [15]. The voltage and frequency values used in our evaluations range from 50MHz and 0.65V to 1000MHz and 1.75V. The power consumption of the CPU at a given frequency/voltage setting is derived in the simulator from actual processor and cache activity. It varies significantly from approximately 15mW at 0.65V up to 2.2W at 1.75V.

We first consider a base case memory design in which the chips are always active, ready for an access. We refer to this case as *power oblivious memory*. A meager step in the direction of power-awareness is called *naive powerdown*

and represents the policy in which the memory chips remain active until task completion at which point they are powered down through the slack time to the end of the period (note data is not lost in this powerdown state). Next, we explore two power-aware memory controller policies called *immediate nap* and *immediate standby*<sup>1</sup>. Each of these policies immediately transitions a DRAM chip to the corresponding power state whenever there are no outstanding accesses to the chip. This is essentially specifying a threshold value of zero on the lengths of gaps that trigger such a transition. In both policies, a DRAM chip enters the powerdown state if it has not observed a reference for 500 microseconds. We assume the OS employs sequential page allocation to complement the hardware capabilities. The immediate transitioning and sequential page allocation represent “best practice” according to previous research [13, 5, 6].

We consider multimedia applications as representative of workloads for low power devices and because they appear to be amenable to good predictions of future processing demands on a per-task basis [24]. Specifically, we use the MPEG decoder from the MediaBench suite [14, 2] running at 15 frames per second (a period of 66ms). We use an input file of 3 frames consisting of one I-frame (intra-coded), one P-frame (predictive) and one B-frame (bidirectional). We present results for the P-frame, the other frames produce similar results. At this frame rate, decoding a single frame at our slowest frequency of 50MHz nearly fills the designated period for all of our experiments. Since the period of our application is set to match its execution time at 50MHz, we can explore energy consumption over the full range of available voltages without concern for missed deadlines. One way of viewing this is that the candidate frequencies which can deliver adequate performance have already been identified so the question of which voltage delivers the best results for our energy metrics – memory energy and total energy (memory + CPU) – can be fully explored.

The other question we need to address is how the DVS algorithm can map the known performance needs of a task into a frequency range that can meet those needs when memory policies and behavior may have an effect on that performance. We explore the variation in execution times, defined as the busy portion of our experimental period, across the frequency range to understand the factors that the DVS algorithm must take into account.

We use a synthetic benchmark that can model a variety of computation times and cache miss ratios to further explore those memory effects in a controlled fashion. For each miss ratio targeted, the synthetic benchmark is configured with a period adequate to just accommodate the execution of one task at 50MHz while barely meeting its deadline. For example, when we target a 9% data cache miss ratio (actually achieving a 9.06% miss ratio), the task executes for approx-

<sup>1</sup>We sometimes shorten these to “nap” and “standby”.

imately 30ms at 50MHz so we choose a 30ms period for that scenario. For this configuration 45% of the instructions are memory references and the misses/instruction is 4.1%. Later experiments configure the benchmark with different miss ratios from 2% to 16.5% and corresponding periods.

## 4 DVS and Power Oblivious Memory

We begin our analysis by illustrating the impact of memory on the effectiveness of DVS for a conventional memory system in which the memory power consumption can be modeled as constant, independent of variations in processor voltage or power state. Therefore, memory energy consumption is easily computed as  $DRAM_{chips} \times ChipPower \times Time$ . For our system configuration we have two chips, with 300mW each, and the MPEG period is 66ms. This results in a total memory power consumption of 600mW and energy consumption of approximately 39.6mJ. In this case, memory dominates overall energy even for the highest voltage setting of the processor (total energy of 47.67mJ). Thus, we expect memory to dilute the impact of DVS on overall energy consumption.

An alternative to keeping memory powered on all the time is to power down both the CPU and memory for the time between task completion and the end of the period. It is this idle time that many DVS algorithms seek to minimize by stretching the execution.

Table 2 shows that our simulation results match our expectations. This table provides statistics on CPU power, execution time, average gap for chip 0, memory energy, CPU energy and total energy for various voltage (frequency) settings. We divide memory and CPU energy into two portions. The first portion corresponds to the energy consumed while the task is executing (the active part of the period). The second portion (labelled “Residue”) is the energy consumed during the time between the task completion and the end of the period (i.e., CPU leakage power and DRAM active for power oblivious memory, DRAM in powerdown for naive powerdown memory).

From the data in Table 2 we see that for the power oblivious memory system the lowest energy is achieved by using the lowest CPU voltage setting. Since the memory power is constant over the entire period, the lowest energy is achieved by minimizing the CPU energy. However, while the CPU energy changes by a factor of 7, the total energy savings from lowering voltage is 14%. These savings would be even lower if more DRAM chips were used (e.g., in a laptop with eight memory chips).

The key problem with the traditional memory design in the context of DVS is that DRAM remains powered on during the idle portion of the period. Power-aware memory offers the opportunity to reduce the energy consumed during idle times by placing DRAM chips into lower power states.

The naive implementation enables the DVS scheduler to issue a “command” that places DRAM into the powerdown state.

Table 2 shows that this naive approach lowers overall energy consumption by dramatically reducing the memory residual energy consumption. However, we note a dramatically different effect of DVS on total energy. At 50MHz, memory remains powered on too much and dominates total energy which equals 38.77mJ. In contrast, at 1GHz execution time does not decrease enough to offset the substantial increase in CPU power and total energy is 10.46mJ. Therefore, total energy has a u-shape as a function of processor frequency/voltage, and the lowest energy is achieved at 600Mhz.

This result is in direct conflict with conventional wisdom used in many DVS algorithms. It is no longer best to stretch execution to consume the entire period. Instead, the best frequency/voltage for minimizing energy should be obtained by including memory energy.

Although the naive powerdown approach can reduce total energy, it does not exploit the full capabilities of power-aware memory. The low power state is entered only after task completion. The following section investigates the interaction between processor voltage scaling and sophisticated power-aware memory that utilize low power states while a task is active.

## 5 DVS and Power-Aware Memory

This section explores the interaction between DVS and power-aware memory. In contrast to the naive approach described above, these policies manipulate DRAM power states during the active portion of the task period. By default they will all place the DRAM chips into powerdown for the idle portion of the task period. We begin by considering the behavior of the *immediate nap* (nap) policy for various frequency values. We note that our workload fits entirely in one memory chip, thus the remaining chip can power down even while the task is active. Since powered down chips consume very little energy, our analysis in this section should apply to scenarios with a larger number of chips, but with most in the powerdown state. We validated this through simulations with eight DRAM chips, but omit the results for brevity.

Figure 3 shows energy versus frequency (a) and execution time versus frequency (b). The three lines in the energy graph correspond to the total energy, memory energy, and processor energy. From this graph, and the data in Table 3, we see that the *immediate nap* policy has significantly different behavior than either a traditional memory system or the naive powerdown approach. At high frequency the total energy is comparable to the naive powerdown policy. However, at low frequency the total energy is much lower.

CPU Freq (MHz)	CPU Pwr (mW)	Exec Time (ms)	Avg Gap0 (ns)	CPU Eng (mJ)	CPU Residue (mJ)	Mem Eng (mJ)	Oblivious		Naive	
							Mem Residue (mJ)	Total Eng (mJ)	Mem Residue (mJ)	Total Eng (mJ)
50	16.5	62.85	34915.7	1.04	0.00	37.71	1.89	40.64	0.02	38.77
100	38.3	31.52	17478.3	1.21	0.00	18.91	20.69	40.81	0.21	20.33
200	99.5	15.85	8755.2	1.58	0.01	9.51	30.09	41.19	0.30	11.40
400	308.6	8.02	4803.0	2.47	0.05	4.81	34.79	42.12	0.35	7.68
600	659.8	5.42	3550.2	3.58	0.10	3.25	36.35	43.28	0.36	7.30
800	1184.2	4.13	2941.9	4.89	0.19	2.48	37.12	44.69	0.37	7.94
1000	2285.2	3.36	2568.7	7.69	0.38	2.02	37.58	47.67	0.38	10.46

**Table 2. DVS with Power Oblivious and Naive Powerdown Memory**

CPU Freq (MHz)	CPU Pwr (mW)	Exec Time (ms)	Avg Gap0 (ns)	Mem Pwr (mW)	Mem Eng (mJ)	CPU Eng (mJ)	Mem Residue (mJ)	CPU Residue (mJ)	Total Eng (mJ)
50	16.5	63.07	34877.64	31.5	1.98	1.04	0.27	0.00	3.30
100	38.1	31.74	17481.24	32.9	1.04	1.21	0.37	0.00	2.62
200	98.5	16.08	9210.23	35.6	0.57	1.58	0.42	0.01	2.58
400	302.1	8.26	5614.57	39.8	0.33	2.50	0.44	0.05	3.31
600	638.9	5.68	4314.45	43.5	0.25	3.63	0.45	0.10	4.43
800	1134.7	4.40	3520.86	46.9	0.21	4.99	0.45	0.19	5.84
1000	2168.7	3.63	2969.17	50.1	0.18	7.88	0.45	0.38	8.90

**Table 3. DVS and Power Aware Memory: MPEG Decode**

As the frequency increases from 50MHz to 1GHz, the total energy initially decreases from 3.30mJ at 50MHz to 2.58mJ at 200MHz, then steadily increases to a maximum of 8.9mJ at 1GHz. As with the naive powerdown policy, these results illustrate the problem with most existing DVS algorithms that ignore memory effects where it is assumed that the lowest frequency/voltage that meets performance constraints also achieves the lowest energy consumption. However, the penalty for choosing the lowest frequency is much lower with the immediate nap policy.

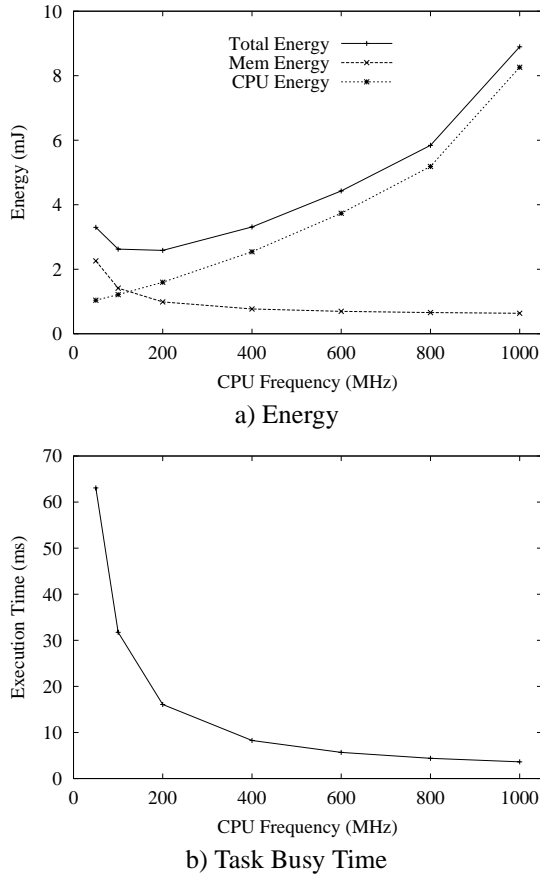
This behavior is explained by examining the processor and memory energy components. Processor energy steadily increases quadratically with the increased voltage required for each higher frequency. In contrast, the memory energy initially decreases from 1.98mJ, then stabilizes at around 0.2mJ. The overall effect is that at low frequencies, memory dominates total energy, while processor energy dominates at high frequencies.

Further insight is gained by examining the effect of increased frequencies on execution time (see Figure 3b.) At 50MHz the task takes 63ms to execute. Although processor energy is minimized at this low frequency, the long execution time causes memory to remain powered on longer. This

increases the memory energy consumption, thus increases the overall energy consumption.

Increasing the frequency initially reduces memory energy consumption more than the processor energy increases, since memory is not powered up as long. However, after 400MHz further increases in frequency fail to significantly improve execution time enough to offset the quadratic increase in CPU power consumption. Note also that average memory power consumption increases with increasing frequency since the average gap decreases and the lower power state can not be exploited for as long or as often during the busy phase. Therefore, memory energy stabilizes while processor energy continually increases.

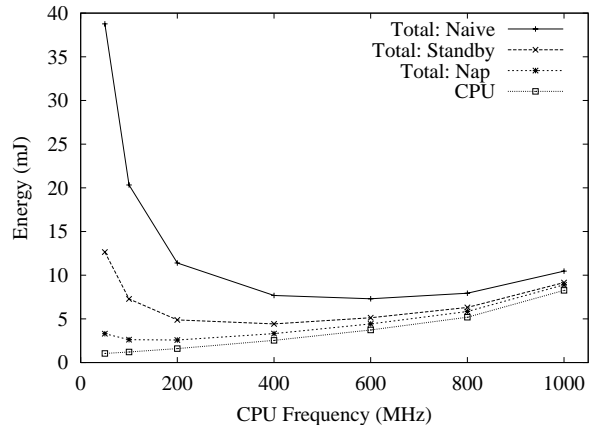
From the discussion thus far, we can make several important observations. First, the naive implementation that powers down memory during idle portions of the period can produce lower overall energy consumption than a power oblivious memory. However, this result conflicts with DVS algorithms that assume the lowest frequency will produce the lowest energy (this assumption only holds for the CPU). Figure 4 illustrates these results by showing energy consumption versus frequency. One line is for CPU energy only, the other lines correspond to various power-aware



**Figure 3. DVS and Power Aware Memory: MPEG Decode**

memory policies and include both CPU and memory energy. The two sophisticated power management policies (standby and nap) lower the overall energy consumption, particularly at the lower frequencies. The standby policy does not reduce the low frequency energy consumption as much as the nap policy because it has a higher base power state. The minimum energy point shifts to the lower frequencies as the power-aware policy becomes more aggressive (from naive to standby to nap). A conclusion to draw from this comparison of memory policies is that more effective power-aware memory management contributes to realizing the potential of DVS.

The final observation from the above discussion concerns the influence that limitations on execution time can have on energy consumption. For MPEG this was simply the linear effects of frequency changes versus the quadratic effects on power consumption. However, many benchmarks may have other execution time bottlenecks. In particular, cache behavior can have a dramatic effect on execution time for some programs. MPEG has a very low data cache miss



**Figure 4. DVS and Memory Controller Policies**

ratio, however several researchers have identified embedded applications that incur miss ratios from 5% to 15% depending on cache configuration. Bishop et al [2] show that PEGWIT, a public key encryption tool in the MediaBench suite, can have a miss rate of 15% in a 16KB 32-way data cache. Therefore, we believe it is important to fully explore the impact of cache performance on DVS with power-aware memory.

## 5.1 Miss Ratio Effects

To further explore the influence of memory latency on DVS we now consider the effect of changing the workload’s miss ratio on voltage setting. We use a synthetic benchmark to create three workloads with different execution periods and miss ratios. One workload has a low miss ratio (2%) and period of 160ms, the next workload has a 9% miss ratio and a 30ms period, the final workload we model has a high miss ratio (16.5%) and a 41ms period. For each workload the 50MHz frequency is sufficient to complete task execution in the requisite period.

Our goal is to examine the behavior of each workload as the processor voltage is scaled. Therefore, we present normalized results to avoid accidental comparisons between workloads. Figure 5a) shows the total energy normalized to the 50MHz value for each workload, while Figure 5b) shows execution time normalized in a similar manner.

From Figure 5 we see similar trends for each workload as frequency increases. Energy initially decreases, then increases dramatically as processor power increases. We note that the overall energy increases more rapidly for higher miss ratios. This is because the higher miss ratio workloads “hit the memory wall” sooner, reducing the benefit of increased clock frequencies, thus causing memory energy to stabilize sooner.

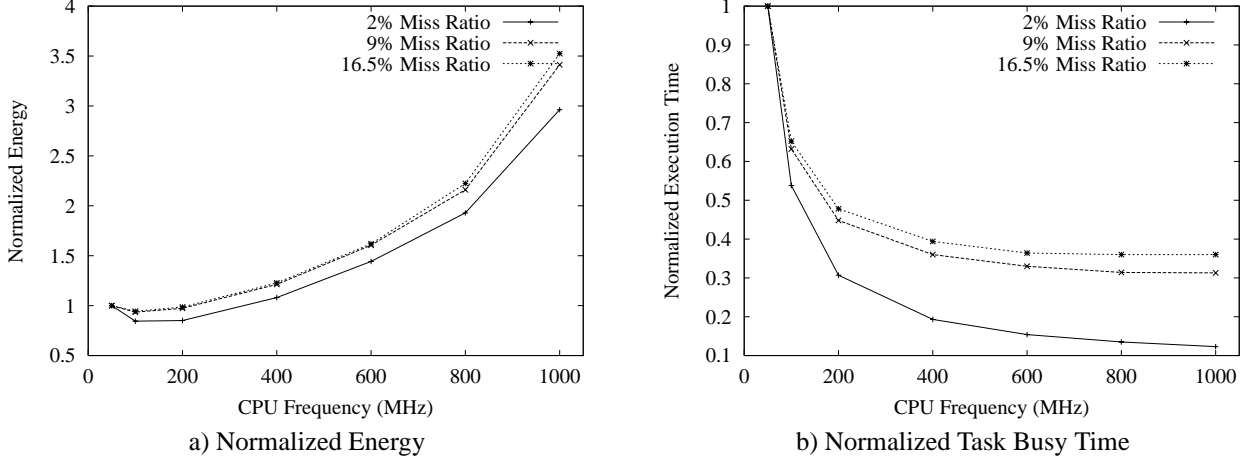


Figure 5. Miss Ratio Effects on DVS: Normalized to the 50MHz value.

These results indicate that DVS algorithms must consider memory energy consumption when setting voltage levels. Obviously, DVS algorithms should also consider "memory wall" effects when determining which frequencies meet the deadlines. The challenge is to develop a method for determining what voltage/frequency level should be used to minimize overall energy and meet deadlines. The following section outlines our approach for meeting this challenge.

## 5.2 Toward Memory-Aware DVS

To incorporate knowledge of memory system effects into a DVS algorithm we must be able to estimate the execution time and overall energy consumption for each available voltage (frequency) setting. The overall energy is determined by  $TotalEnergy = ProcessorEnergy + MemoryEnergy$ . We can obtain the overall energy by obtaining each of the components.

To estimate processor energy consumption we multiply the estimated execution time by the estimated power consumption. We use typical power values [15] and the standard CPU time formula given in Equation 1 [10].

$$CPU_{time} = \frac{Instructions}{Program} \times \frac{Cycles}{Instruction} \times \frac{Seconds}{Cycle} \quad (1)$$

To compute  $\frac{Cycles}{Instruction}$  (CPI) we assume a base CPI of one and include memory hierarchy effects ( $CPI = 1 + r * m * Mp$ ), where  $r$  is the fraction of instructions that are memory references,  $m$  is the cache miss ratio and  $Mp$  is the miss penalty in cycles. This allows us to compute the execution time for each available frequency, eliminating from further consideration those that fail to meet deadlines. We can then obtain an estimate of the processor energy consumption for

each frequency by multiplying the execution time by the power consumption for the particular frequency. We note that all information required to predict the processor energy consumption is readily available with existing performance counters in modern processors.

We can estimate the total memory power consumption by determining the amount of time spent in each of the power states. We focus on the immediate nap policy that uses three power states: 1) active, 2) nap and 3) power-down. We must also consider the energy consumed to transition between the power states. For this analysis we assume the only power consuming transitions are from nap to active, and that a chip that enters powerdown is never accessed again.

We can compute the total memory energy for a single period (Equation 2) by computing the energy consumed while in the active state, the nap state, transitioning between nap and active, and while powered down during the idle portion of the period. Each product term in Equation 2 corresponds to the energy ( $Time \times Power$ ) for each of these states or transition. The power consumption values for each term in Equation 2 are given in product specifications. Therefore, to determine the overall energy we only need to determine the time values for each product term. Note our residual energy calculation ignores the threshold time incurred before entering powerdown.

$$E_{memory} = T_{active} * P_{active} + T_{nap} * P_{nap} + T_{nap \rightarrow active} * P_{nap \rightarrow active} + T_{residual} * P_{powerdown} \quad (2)$$

The residual time is easily computed by subtracting our CPU time estimate from the provided period ( $T_{residual} = T_{period} - CPU_{time}$ ). Similarly, we can compute the time

spent in the active state by subtracting the nap and nap to active times (see Equation 3).

$$T_{active} = CPU_{time} - T_{nap} - T_{nap \rightarrow active} \quad (3)$$

The next step is to determine the time spent in the nap state and the time spent transitioning back to active. The time spent in the nap state can be computed by multiplying the number of gaps by the average gap as shown in Equation 4. Similarly, we can compute the time spent in transition from nap to active by multiplying the number of gaps by the transition time (see Equation 5). Note that this approach assumes that each gap requires a transition.

$$T_{nap} = N_{gaps} * t_{average\_gap} \quad (4)$$

$$T_{nap \rightarrow active} = N_{gaps} * t_{nap \rightarrow active} \quad (5)$$

Precisely modeling memory energy consumption requires accurate values for the number of gaps and the average gap. While there may be many ways to determine these values, our goal is to show that with information readily available from existing performance counters we can approximate memory energy consumption. We begin by assuming that the number of gaps is equivalent to the number of misses. This assumption ignores any clustering of misses that may occur and is reasonable for our in-order execution processor model. Furthermore, we observed from our simulation results that the number of misses is close to the number of gaps.

Given the number of gaps, we compute the average gap by the following:  $gap = \frac{cycletime * Instructions}{misses}$ . This calculation assumes a base CPI of one (the peak performance of our processor). We then substitute into the previous equations and obtain the total energy including both processor and memory. Furthermore, our energy equation is general enough to capture a variety of applications with different characteristics.

We use the workload with a 9% miss ratio to evaluate our energy estimates. This workload executes 981,481 instructions, 444,702 memory references and incurs 40,297 misses. We assume every miss incurs a full nap to active transition, for a total miss penalty of 180ns. For processor power, we use typical values [15]. Figure 6 shows the measured energy (Simulation) and our predicted energy (Miss Counters) versus clock frequency. For comparison, we also include a prediction (Gap Prediction) that uses the measured number of gaps, average gap and execution time values from the simulator for each clock frequency.

The first observation from these results is that our prediction of total energy matches the general shape of the simulation results. We also observe that the miss counter prediction generally underpredicts the energy, except at the

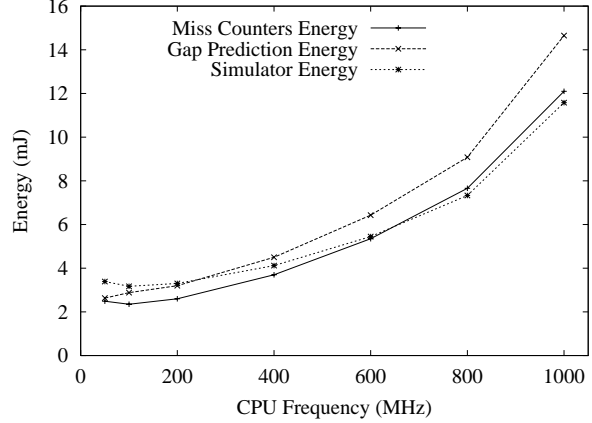


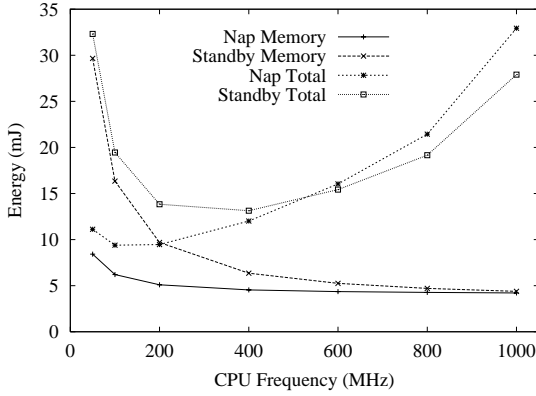
Figure 6. Predicting Energy Consumption

highest frequencies. Figure 6 also shows that using the measured gap values tends to overpredict energy consumption. However, we note that errors in energy calculation can also occur because of the processor power values. We use typical values [15], while the actual processor power for this workload is lower. We are currently investigating refinements to our model that reduce this error. Nonetheless, the estimates appear to be sufficient for a DVS algorithm to choose an appropriate frequency.

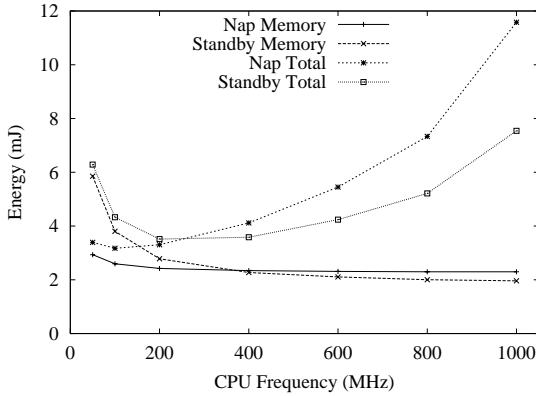
## 6 DVS and Memory Controller Policies

The previous section highlights the importance of using a power aware memory system to fully exploit the potential of DVS. However, previous studies [5, 6] show that different memory controller policies may be appropriate depending on the average gap between clustered accesses. This section explores the impact of DVS on memory controller policy selection by exploring how changes in processor frequency influence the average gap observed by the memory controller.

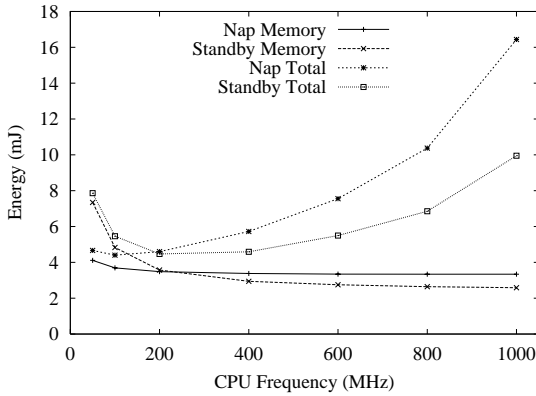
Figure 7 shows energy versus processor frequency for the three workload configurations with different miss ratios. Our evaluation considers the two memory controller policies, immediate nap and immediate standby, as described in Section 3. From these graphs we make the following observations. First, for all workloads, as the frequency increases, there is a crossover point in both the pair of total energy lines and the pair of memory energy lines from nap as the best policy to standby as the best. This crossover point is different for each workload. For the 2% miss ratio, standby becomes the policy of choice for total energy between 400MHz and 600MHz. The crossover point for memory energy is at 1000MHz. As the miss ratio increases, this crossover point shifts to lower frequencies. For the 9% miss ratio, both crossovers are between 200MHz and



a) 2% Miss Ratio



b) 9% Miss Ratio



c) 16.5% Miss Ratio

**Figure 7. DVS Effects on Memory Controller Policy**

400MHz, while for 16.5%, the crossover is at 200MHz.

These effects directly result from changes in the average gap between clusters of DRAM accesses (see Table 4). For a fixed miss ratio, the average gap is reduced by increasing the frequency, eventually making standby the policy of choice. Furthermore, for a fixed frequency the higher

Frequency (MHz)	Average Gap (ns)		
	2% Miss	9% Miss	16.5% Miss
50	2439	556	456
100	1217	276	226
200	607	136	110
400	304	69	56
600	202	46	36
800	152	34	28
1000	120	27	22

**Table 4. Frequency and Miss Ratio Effects on Chip 0 Average Gap**

miss ratios incur a smaller average gap. This pushes the crossover point to lower frequencies for higher miss ratios.

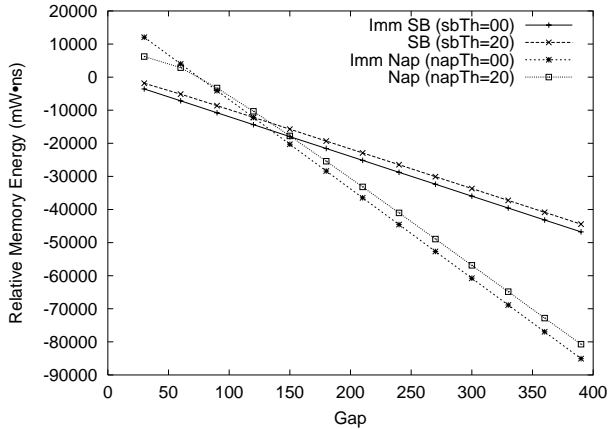
The memory controller policy most directly affects memory energy. As we see in Table 4, the average gap values at the crossover frequencies for memory energy are all in the same 69-136ns range across all three miss ratios. This suggests that it may be appropriate for the memory controller to dynamically switch policies based on the gap values.

To validate this idea, we adopt the Petri Net modeling approach developed in [6] for analyzing power state control policies. By matching the timings in the Petri Net model to the parameter values used in the simulation, we derive analytical results indicating an average gap value of 136ns as the crossover between nap and standby for optimizing memory energy consumption. Figure 8 shows the Petri Net results. The plot shows the relative memory energy consumption for different state transition thresholds as gap increases.<sup>2</sup> The lines represent four different policies: immediate standby (standby threshold = 0), standby with standby threshold of 20ns, immediate nap (nap threshold = 0), and nap with nap threshold of 20ns. The first observation confirms previous results that zero thresholds are best among a family of policies differing in their threshold values. The main point of this figure is the crossover of the best policy (lowest line) from immediate standby to immediate nap at 136ns.

The difference between the 136ns result and the experimental values can be explained by the modeling of the memory access pattern as an exponential distribution in the Petri Net which does not capture the actual memory behavior of the workload. However, we feel the analysis strongly supports using the average gap to dynamically select a control policy.

When considering total energy, CPU energy becomes a

<sup>2</sup>Note that since increasing gaps correspond to decreasing frequencies, this plot appears flipped left-to-right compared with previous graphs.



**Figure 8. Petri Net Results:** Memory energy, relative to always active, for different thresholds

contributing factor, especially with lower miss ratios. We observe that only for the high miss ratio workload does the standby policy produce the absolute lowest total energy for executing the task (at 200MHz). For the lower miss ratio workloads, the frequency required to reduce the average gap to the point where standby is the better policy causes the CPU power consumption to become a significant component of the total energy. We also note that at low frequencies, the nap policy is always the best policy. This is because for the standby policy, memory begins to dominate overall energy. The nap policy exploits the larger gaps to further reduce memory power consumption.

The significance of these results is that, if the DVS algorithm is forced to choose one of the higher frequencies to meet a required deadline, then the complementary ability to set the memory controller policy from immediate nap to immediate standby may be beneficial.

## 7 Summary and Conclusions

Dynamic voltage scheduling (DVS) is a popular technique for reducing processor power and energy consumption. However, DVS algorithms that ignore memory effects may result in incorrect processor voltage/frequency settings. This paper explores the interaction between memory and processor DVS.

Our analysis uses a detailed processor and memory system simulator with a multimedia benchmark. The simulation results reveal that traditional high power, low latency memory designs can dominate overall energy consumption. In contrast, power-aware memory systems that enable individual DRAM chips to transition to lower power states create a tradeoff between memory and processor energy. At low frequencies memory dominates overall energy since it

remains powered up for too long. As frequency increases, total energy initially decreases but then increases with processor energy. This is because execution time initially decreases, reducing memory energy, but does not decrease much for continued increases in frequency. At this point memory energy stabilizes, but processor energy continues to increase due to increased power consumption.

Given the tradeoff between memory and processor power consumption and memory’s influence on execution time, we propose a technique to estimate execution time and the total energy consumption for a given task. Our approach requires only three parameters to characterize the workload behavior: 1) the number of instructions to execute, 2) the number of memory references, and 3) the number of cache misses. These values are all easily obtained with existing performance counters on many modern processors. We show that our execution time and energy estimates are sufficient to capture the tradeoff between memory and processor energy consumption, and can be used by a DVS algorithm to select an appropriate voltage/frequency setting.

Our analysis also reveals that frequency scaling influences the choice of memory controller policy for transitioning chips between power states. Power-aware memory transitions a chip to a lower power state when there are no accesses to service. An important aspect of these policies is determining which lower power state to enter. This decision is based on the average gap between clusters of accesses. Simulation results show that as frequency increases from 50MHz to 1GHz, the policy of choice switches from immediately transitioning to the Nap state to a policy that immediately transitions to Standby. The increased frequencies reduce the average gap, making the cost of transitioning out of Nap too expensive.

The results presented in this paper show that power-aware memory offers increased opportunities to exploit voltage scaling. Furthermore, DVS algorithms must include memory effects to fully exploit the potential of voltage scaling. Building upon these insights about the factors governing the memory-DVS interactions, we plan to explore DVS algorithms that include memory effects and refine our energy estimate model in our future work.

## References

- [1] The SimpleScalar-Arm Power Modeling Project. [//www.eecs.umich.edu/tnm/power/](http://www.eecs.umich.edu/tnm/power/).
- [2] B. Bishop, T. Kelliher, and M. Irwin. A Detailed Analysis of MediaBench. In *1999 IEEE Workshop on Signal Processing Systems*, November 1999.
- [3] D. C. Burger, T. M. Austin, and S. Bennett. Evaluating Future Microprocessors-the SimpleScalar Tool Set. Technical Report 1308, University of Wisconsin-Madison Computer Sciences Department, July 1996.

- [4] V. Delaluz, M. Kandemir, N. Vijaykrishnan, A. Sivasubramiam, and M. Irwin. DRAM Energy Management using Software and Hardware Directed Power Mode Control. In *Proceedings of 7th Int'l Symposium on High Performance Computer Architecture*, January 2001.
- [5] X. Fan, C. S. Ellis, and A. R. Lebeck. Memory Controller Policies for DRAM Power Management. In *Proceedings of the International Symposium on Low Power Electronics and Design (ISLPED)*, pages 129–134, August 2001.
- [6] X. Fan, C. S. Ellis, and A. R. Lebeck. Modeling of DRAM Power Control Policies using Deterministic and Stochastic Petri Nets. In *Lecture Notes in Computer Science: Proceedings of the Workshop on Power Aware Computing Systems*. Springer-Verlag, February 2002. To appear.
- [7] K. Flautner, S. Reinhardt, and T. Mudge. Automatic performance setting for dynamic voltage scaling. In *The Seventh Annual International Conference on Mobile Computing and Networking 2001*, pages 260–271, 2001.
- [8] K. Govil, E. Chan, and H. Wasserman. Comparing algorithms for dynamic speed-setting of a low-power CPU. In *Proceedings of First Annual International Conference on Mobile Computing and Networking*, November 1995.
- [9] D. Grunwald, P. Levis, K. Farkas, C. Morrey, and M. Neufeld. Policies for Dynamic Clock Scheduling. In *Proceedings of Symposium on Operating Systems Design and Implementation (OSDI)*, October 2000.
- [10] J. L. Hennessy and D. A. Patterson. *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann, 2 edition, 1995.
- [11] C. Hughes, J. Srinivasan, and S. Adve. Saving Energy with Architectural and Frequency Adaptations for Multimedia Applications. In *Proceedings of the 34th International Symposium on Microarchitecture (MICRO 34)*, December 2001.
- [12] C. M. Krishna and Y.-H. Lee. Voltage-clock-scaling techniques for low power in hard real-time systems. In *Proceedings of the IEEE Real-Time Technology and Applications Symposium*, pages 156–165, May 2000.
- [13] A. R. Lebeck, X. Fan, H. Zeng, and C. S. Ellis. Power Aware Page Allocation. In *Proceedings of the Ninth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS IX)*, pages 105–116, November 2000.
- [14] C. Lee, M. Potkonjak, and W. Mangione-Smith. Media-Bench: A Tool for Evaluating and Synthesizing Multimedia and Communications Systems. In *30th International Symposium on Microarchitecture (MICRO 30)*, pages 330–335, December 1997.
- [15] S. Leibson. XScale (StrongARM-2) Muscles In. *Microprocessor*, September 2000.
- [16] J. R. Lorch and A. J. Smith. Improving dynamic voltage scaling algorithms with PACE. In *Joint International Conference on Measurement and Modeling of Computer Systems*, pages 50–61, 2001.
- [17] T. Martin. *Balancing batteries, power and performance: System issues in CPU speed-setting for mobile computing*. PhD thesis, Carnegie Mellon University, 1999.
- [18] T. Martin and D. Siewiorek. Non-ideal Battery and Main Memory Effects on CPU Speed-Setting for Low Power. *Transactions on Very Large Scale Integrated Systems, Special Issue on Low Power Electronics and Design*, 9(1):29–34, February 2001.
- [19] T. L. Martin, D. P. Siewiorek, and J. M. Warren. A CPU Speed-Setting Policy that Accounts for Nonideal Memory and Battery Properties. In *Proc. 39th Power Sources Conf.*, pages 502–505, June 2000.
- [20] D. Mosse, H. Aydin, B. Childers, and R. Melhem. Compiler-assisted dynamic power-aware scheduling for real-time applications. In *Workshop on Compilers and Operating Systems for Low-Power (COLP'00)*, October 2000.
- [21] T. Pering, T. Burd, and R. Brodersen. Voltage Scheduling in the lpARM Microprocessor System. In *Proceedings of International Symposium on Low Power Electronics and Design*, 2000.
- [22] T. Pering, T. D. Burd, and R. W. Brodersen. The Simulation and Evaluation of Dynamic Scaling Algorithms. In *Proceedings of the International Symposium on Low Power Electronics and Design*, August 1998.
- [23] P. Pillai and K. G. Shin. Real-time dynamic voltage scaling for low-power embedded operating systems. In *Proceedings of the 18th Symposium on Operating Systems Principles*, pages 89 – 102, October 2001.
- [24] J. Pouwelse, K. Langendoen, and H. Sips. Dynamic voltage scaling on a low-power microprocessor. In *The Seventh Annual International Conference on Mobile Computing and Networking 2001*, pages 251–259, 2001.
- [25] Rambus. *RDRAM*, 1999. <http://www.rambus.com/>.
- [26] V. Swaminathan and K. Chakrabarty. Real-time task scheduling for energy-aware embedded systems. In *Proceedings of the IEEE Real-Time Systems Symp. (Work-in-Progress Session)*, November 2000.
- [27] M. Weiser, B. Welch, A. Demers, and S. Shenker. Scheduling for Reduced CPU Energy. In *USENIX Association, Proceedings of First Symposium on Operating Systems Design and Implementation (OSDI)*, November 1994. Monterey CA.