

# The Synergy between 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

TR CS-2002-12

## 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 is increased interest in scaling the CPU voltage and clock frequency. Dynamic Voltage Scaling (DVS) is the technique for exploiting hardware capabilities to select an appropriate clock rate and voltage to meet application requirements at the lowest energy cost. Unfortunately, the power and performance contributions of other system components, in particular memory, complicate some of the simple assumptions upon which most of the DVS algorithms have been based.*

*We show that there is a positive synergistic effect between DVS and power-aware memories that can transition into lower power states. This combination can offer greater energy savings than either technique alone. 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. Complementary policies for controlling DVS processors and power aware memory chips offer opportunities to achieve more effective system-wide energy use.*

## 1 Introduction

Energy consumption is becoming a limiting factor in the development of computer systems for a range of applica-

tion domains – from mobile and embedded devices that depend on batteries to large hosting centers that incur enormous 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 CPU supply voltage and clock frequency. 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 Scaling (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, 10, 8, 24, 9], primarily addressing the prediction issue. Most simulation-based studies of these algorithms have focussed solely on CPU energy consumption and have ignored both the power and performance contributions of other system components.

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 overall energy 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 for those solutions based primarily on observations of CPU

load and the inclusion of other components of the system beyond the CPU, especially interactions with memory [10, 8, 7, 17, 18, 19, 24]. Thus, the impact of memory has been considered to be a complicating factor for the straightforward application of DVS.

In this paper, we identify a positive synergy between voltage/frequency scaling of the processor and power-aware memory systems that offer their own power management features.

To evaluate the interactions 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 several periodic applications from the MediaBench suite.

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

- We discuss what it means to have an “energy-balanced” system design. The accepted notion of a balanced system (i.e., performance-based, using Amdahl’s law) suggests 1MB of memory per 1 MIPS of CPU. The implication of this rule-of-thumb using traditional full-power memory chips with modern low-power, DVS-capable processors is that memory energy may dominate processor energy such that the overall impact on system energy of employing DVS is marginal. By better aligning the energy consumption of the processor and memory, the individual power management innovations of each device can produce greater benefits. We demonstrate that effective power-aware memory policies enhance the overall impact of DVS by significantly lowering the power cost of memory relative to the CPU.
- The synergy between DVS and sophisticated power-aware memory go deeper than achieving a lower power design point. 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. Thus, the lowest speed setting of the processor may not deliver the minimal combined energy use of processor and memory. We explore how different power-aware memory control policies affect the frequency setting decision.
- 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 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 cooperatively 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, confirming previous observations in the context of our environment. 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

This section summarizes previous work on DVS. We also provide background on power aware memory.

### 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 [23, 12, 26, 11] or regular intervals [27, 22, 10, 9] and attempt to slow down computation to just fill the period without missing the deadline or carrying work over into the next interval. The algorithm must 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. 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 increase prediction of task execution demands [8, 7, 16, 24, 20]. These studies provide the rationale for our assuming good predictions for a specific workload.

The speed-setting decision has appeared to be straightforward, given good predictions. However recent experimental work [17, 24, 10] has suggested that memory effects should be taken into account. For computations that

Power State or Transition	Power (mW)	Time (ns)
Active	$P_a = 300$	$t_{access}=90$
Standby	$P_s = 180$	-
Nap	$P_n = 30$	-
Powerdwn	$P_p = 3$	-
Stby $\rightarrow$ Act	$P_{s \rightarrow a} = 240$	$T_{s \rightarrow a} = +5$
Nap $\rightarrow$ Act	$P_{n \rightarrow a} = 165$	$T_{n \rightarrow a} = +60$
Pdn $\rightarrow$ Act	$P_{p \rightarrow a} = 152$	$T_{p \rightarrow a} = +25,000$

**Table 1. RDRAM Power State and Transition Values:**

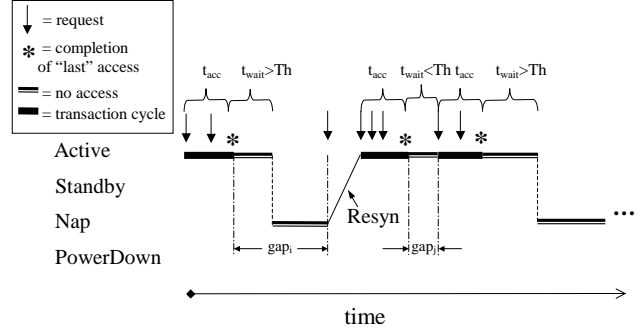
All accesses incur the 90ns active access time. Additional delay (denoted by the +) is incurred for clock resynchronization.

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. We confirm this observation in the context of our target environment and then focus on memory technology that ameliorates the problem.

## 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. We adopt a four-state model consisting of *active*, *standby*, *nap*, and *power-down* 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 implementing dynamic power state transition policies that respond to observable memory access patterns. Such policies often are based on periods of idle time between runs of accesses (which we refer to as *gaps*) and threshold values to trigger transitions. Figure 1 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



**Figure 1. Memory Access Behavior:** A 2-state control policy illustrating the relationship of gaps, thresholds, and state transitions

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 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 our study is to explore power-aware 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 also explore the influence of speed-scaling on the decisions of the power-aware memory controller.

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 con-

sumption 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]. We have performed experiments using four applications from the MediaBench suite [14, 2]: MPEG2decode – an MPEG decoder, PEGWIT – a public key encryption program, G.721 – voice compression, and RASTA – speech pre-processor. The results from these four benchmarks are remarkably consistent. In this paper, we only present data from the MPEG decoder 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 to just accommodate the execution of one task at 50MHz while barely meeting its deadline. We choose a 30ms period and target 3 miss ratios of 2%, 9% and 16%.

## 4 DVS and Power Oblivious Memory

We begin by establishing our base case as 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. This results in a total memory power consumption of 600mW.

We consider the impact of such memory on the effectiveness of DVS for our MPEG decoding benchmark. MPEG’s period of 66ms results in memory 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.

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).

From the data in Table 2 we see that for this 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 only 15%. These relative savings would be even lower if more DRAM chips were used (e.g., in a laptop with eight memory chips). Overall all three benchmarks, the energy savings of DVS in a system with power oblivious memory is consistently between 12%

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

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**

to 16%.

## 5 DVS and Power-Aware Memory

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

### 5.1 Naive Power-awareness

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. This “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 which represents the energy consumed by the DRAM in powerdown mode. The memory energy costs (the sum of the memory energy column and the memory residue for naive) are brought down into the range of CPU energy. In a sense, these two components are *balanced* in terms of energy. The effect of this is to make any power management functions of either the CPU or memory relatively important. Introducing the powerdown capability in the memory yields a 78% total energy savings without frequency scaling (i.e., comparing 10.46mJ to 47.67mJ at 1GHz) and a 85% savings coupled with the best frequency.

However, we note a dramatically different effect of DVS on total energy. At 50MHz, memory remains powered on too long 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. The interesting point is that the lowest total energy consumption is achieved at 600MHz at 7.30mJ. Therefore, total energy has a u-shape as a function of processor frequency/voltage.

This result conflicts with conventional assumptions used in many DVS algorithms which have been concerned only with CPU energy. *Taking into account the energy used by memory with even minimal power management capabilities, it is no longer best to stretch execution to consume the entire period.* In fact, the lowest frequency produces the highest total energy consumption in this case. Instead, the best frequency/voltage for minimizing energy should be obtained by including memory energy in the decision.

### 5.2 Dynamic Power-Aware Memory

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. Next, we investigate the interaction between processor voltage scaling and sophisticated power-aware memory that utilize low power states while a task is active.

In contrast to the naive approach described above, this form of power-aware memory employs memory controller policies that manipulate DRAM power states during the active portion of the task period. By default they 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 MPEG application 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 applies to scenarios with a larger number of chips, but with most in the powerdown state.<sup>2</sup>

<sup>2</sup>We validated this through simulations with eight DRAM chips.

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

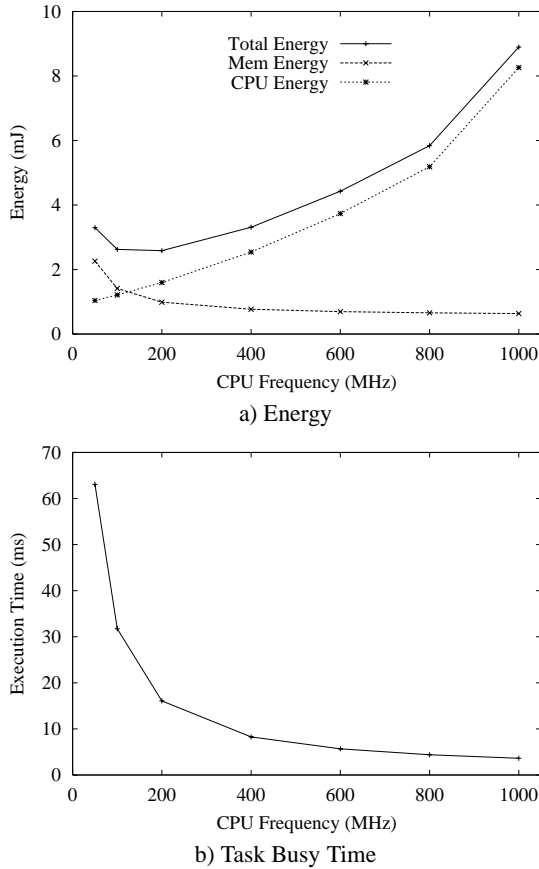


Figure 2. DVS and Power Aware Memory: MPEG Decode

Figure 2 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

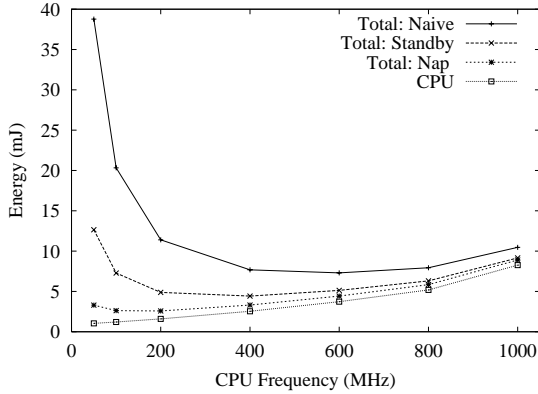
the naive powerdown approach. At high frequency the total energy is comparable to the naive powerdown policy. However, at low frequency the total energy is much lower.

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 that the lowest frequency/voltage that meets performance constraints is not necessarily the frequency that achieves the lowest energy total consumption. However, the penalty for choosing the lowest frequency (as might occur with a DVS algorithm that ignores memory effects) 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 2b.) 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.



**Figure 3. DVS and Memory Controller Policies**

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 3 illustrates these results by showing energy consumption versus frequency. One line is for CPU energy only, the other lines correspond to various power-aware 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 toward 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, other benchmarks 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 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, the public key encryption application in the MediaBench suite, has a miss ratio of 15% in a 16KB 32-way data cache. In our experiments, PEGWIT has a miss ratio

of 2.3% with our 32KB 32-way cache configuration.

### 5.3 Miss Ratio Effects

To explore the influence of memory latency and cache performance on DVS we consider the effect of changing the workload’s miss ratio on voltage setting. Since it is hard to vary the miss ratio with real benchmarks, we use a synthetic benchmark to create three workloads with the same 30ms period but different miss ratios: 2%, 9% and 16%. 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 4a) shows the total energy normalized to the 50MHz value for each workload, while Figure 4b) shows execution time normalized in a similar manner.

From Figure 4 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 execution time of higher miss ratio workloads are limited by memory latency sooner, reducing the benefit of increased clock frequencies, thus causing memory energy to stabilize sooner.

These results indicate that DVS algorithms must consider memory energy consumption when setting voltage levels. Similarly, DVS algorithms should also consider memory’s effect on performance 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.

### 5.4 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  $E_{total} = E_{cpu} + E_{mem}$ .

To estimate processor energy consumption, we multiply the estimated execution time by the estimated power consumption. We use the range of CPU power values given by [15] and represent the power associated with frequency  $f$  by  $P_{cpu}(f)$  in (5). To calculate the execution time, we divide time into 3 parts according to the amount of time spent in each memory state:  $T_{active}$ ,  $T_{nap \rightarrow active}$  and  $T_{nap}$ .  $T_{active}$  is the time spent in *active* power state where accesses are serviced.  $T_{nap \rightarrow active}$  is the time when memory is making transitions from low power state to *active* state. Figure 1

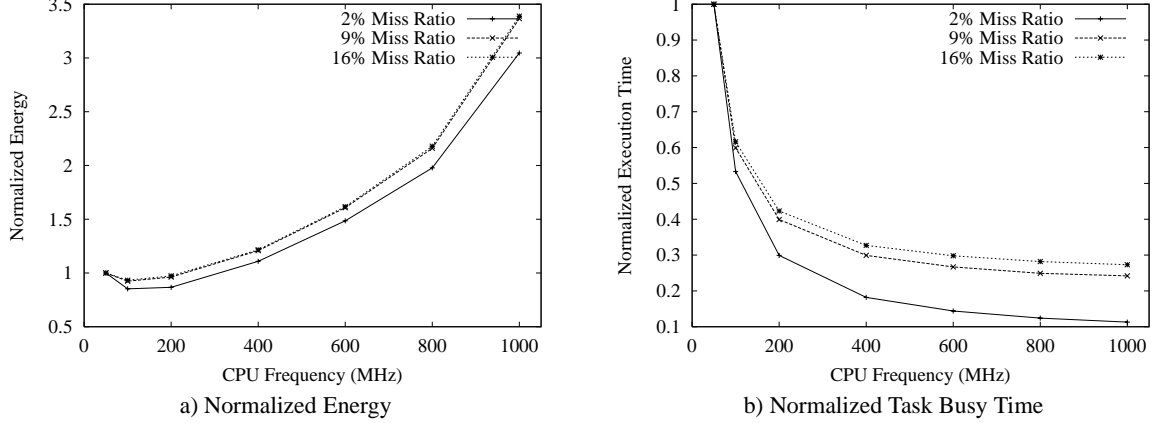


Figure 4. Miss Ratio Effects on DVS: Normalized to the 50MHz value.

shows each transition is followed by an *active* period which services at least one access. To compute these two time values, we need to know how many times the memory makes a power transition and, for each time, how many accesses (cache misses) are serviced. We assume only one access is serviced each time and hence the number of power transitions equals the number of cache misses. We claim it is a reasonable approximation for our inorder processor model. Furthermore, we observed from our simulation results that the number of misses is close to the number of transitions.  $T_{nap}$  is the sum of all periods when CPU does not generate cache misses and the memory remains idle in the low power state. So each instruction except those that trigger a cache miss contributes a cycle to  $T_{nap}$ .

From the above discussion and assuming a base CPI of one, we can calculate the execution time  $T_{exec}$  as follows:

$$T_{active} = t_{access} * N_{misses} \quad (1)$$

$$T_{nap \rightarrow active} = t_{nap \rightarrow active} * N_{misses} \quad (2)$$

$$T_{nap} = \frac{1}{f} * (N_{insts} - N_{misses}) \quad (3)$$

$$T_{exec} = T_{active} + T_{nap \rightarrow active} + T_{nap} \quad (4)$$

The residual time is easily computed by subtracting our estimated execution time from the provided period ( $T_{residual} = T_{period} - T_{exec}$ ). Therefore the energy consumed by CPU and memory, and the total energy as a function of frequency  $f$  can be calculated as follows:

$$E_{cpu} = T_{exec} * P_{cpu}(f) + T_{residue} * P_{leakage} \quad (5)$$

$$\begin{aligned} E_{mem} = & T_{active} * P_{active} + T_{nap} * P_{nap} \\ & + T_{nap \rightarrow active} * P_{nap \rightarrow active} \\ & + T_{residual} * P_{powerdown} \end{aligned} \quad (6)$$

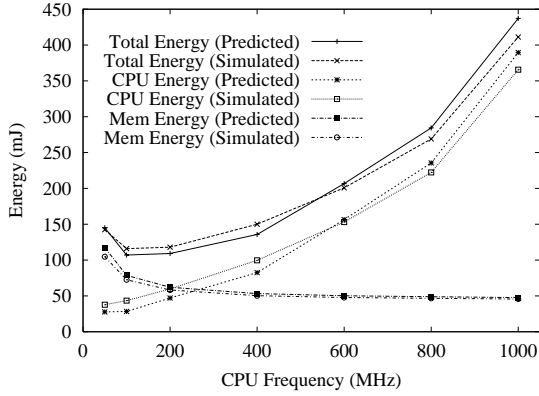
$$E_{total} = E_{cpu} + E_{mem} \quad (7)$$

Note all parameters required to compute the above equations are either available from the hardware specifications ( $t_{access}$ ,  $t_{nap \rightarrow active}$ ,  $P_{active}$ ,  $P_{nap}$ ,  $P_{nap \rightarrow active}$ ,  $P_{powerdown}$ ,  $P_{cpu}(f)$ ,  $P_{leakage}$ ) or easily obtained with existing performance counters on many modern processors ( $N_{misses}$ ,  $N_{insts}$ ,  $f$ ).

We use both synthetic and real workloads to evaluate our energy estimates. Figure 5 shows the measured energy (Simulation) and our predicted energy (Predicted) versus clock frequency for PEGWIT. It includes both the energy of the two components (CPU and memory) and the total energy.

The first observation from these results is that our prediction of each component's energy and total energy matches the general shape of the simulation results. Our model works very well on memory energy prediction and matches simulation closely. We note that the errors in CPU and total energy estimation are primarily due to the fact that the fixed CPU power values we get from [15] can not accurately reflect the actual CPU power consumption obtained from the simulation. Nonetheless, the estimates appear to be sufficient for a DVS algorithm to choose an appropriate frequency.

We also examined how our model and simulation compare for an out-of-order processor, and we get very similar results to the inorder processor. Since the out-of-order processor tends to generate multiple outstanding cache misses, equations (1, 2, 4) generally overpredict the execution time and thus lead to a slightly higher energy estimation for the out-of-order processor than for an inorder processor.



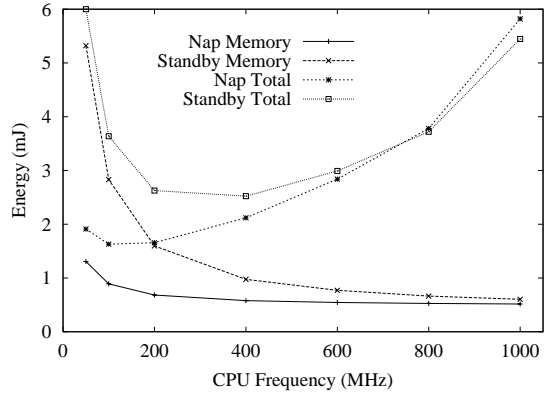
**Figure 5. Energy Prediction – Inorder Processor**

## 6 DVS and Memory Controller Policies

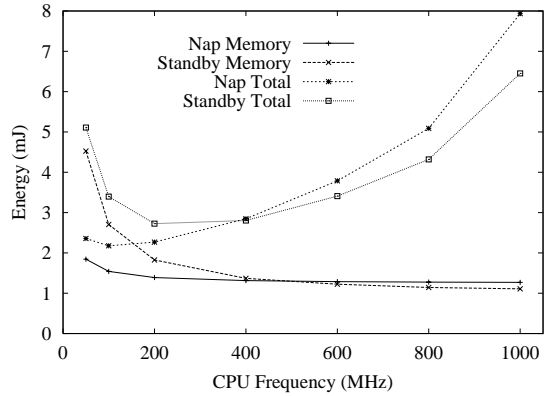
The previous section highlights the importance of using a power aware memory system to fully exploit the potential of DVS. However, previous studies [5, 6] show that different memory controller policies may be appropriate depending on the average gap between clustered accesses. This section explores 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 6 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 the pair of total energy lines from nap as the best policy to standby as the best. For high miss ratio (9%, 16%) workloads, there are also crossover points in the pair of memory energy lines. This crossover point is different for each workload. For the 2% miss ratio, standby becomes the policy of choice for total energy between 600MHz and 800MHz. As the miss ratio increases, this crossover point shifts to lower frequencies. For the 9% and 16% miss ratio workloads, the crossovers for total energy are between 200MHz and 400MHz. The crossover point for memory energy is between 400MHz and 600MHz for the 9% miss ratio and shifts to around 400MHz for the 16% miss ratio.

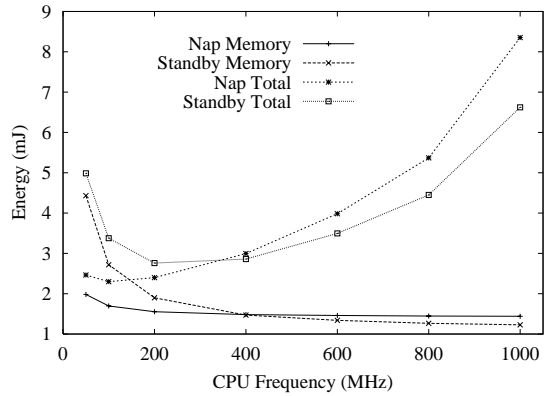
These effects directly result from changes in the average gap between clusters of DRAM accesses (see Table 4). For a fixed miss ratio, the average gap is reduced by increasing the frequency, eventually making standby the policy of choice. Furthermore, for a fixed frequency the higher



a) 2% Miss Ratio



b) 9% Miss Ratio



c) 16% Miss Ratio

**Figure 6. DVS and Memory Controller Policies**

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 in the 46-69ns range for 16% miss ratio, and are around 56ns for 9% miss ratio. This suggests that it may be appropriate

Frequency (MHz)	Average Gap (ns)		
	2% Miss	9% Miss	16% Miss
50	2018	565	465
100	1010	281	231
200	504	140	114
400	252	69	56
600	169	46	38
800	125	33	28
1000	101	28	23

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

for the memory controller to dynamically switch policies based on the gap values.

When considering total energy, CPU energy becomes a contributing factor, especially with lower miss ratios. 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

This work shows there is a synergistic effect between dynamic voltage scaling (DVS) of the processor and power-aware memory control that can produce balanced system design with respect to energy consumption. Our simulation results for four applications from the MediaBench benchmark suite show that combining DVS with power-aware memory achieves greater energy savings than either technique in isolation – a consistent 93-95% savings compared to our full-power base case. By contrast, the energy savings from DVS alone with power oblivious memory are only 12-16%. Using a power-aware memory policy that transitions into nap mode, but without exploiting the DVS capabilities of the processor, yields a total energy savings of 76-85%. The interaction between these two technologies has the greatest impact.

The notion of an energy-balanced system design affects the voltage/frequency setting algorithms of DVS in subtle ways. For a system in which memory power costs are either

dominant or insignificant compared to the range of CPU power values available, the standard assumptions about how performance and energy consumption scale with frequency and voltage of the processor are appropriate in the sense that lowering frequency/voltage can translate into overall energy benefits. Unfortunately, if memory power dominates, then the impact of DVS on the system-wide energy consumption is marginal. On the other hand, configuring a memory system in which the power requirements are sufficiently low relative to the processor without employing at least naive power-aware techniques is not realistic.

Providing power-aware memory can bring the range of power values for the CPU and memory into balance, but in the process introduces tradeoffs between memory and processor energy that complicate the straightforward speed setting policies when total energy is considered. One trend is based on execution times. 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 increasing processor energy. Thus, the lowest frequency may not guarantee the minimal energy use. The effect of longer execution time at lower frequencies is countered by the ability of more sophisticated power-aware policies to exploit longer gaps in memory access patterns during execution to transition into lower power states. The degree of effectiveness of these policies determines the extent to which memory must be a factor in the speed setting decision. For example, comparing the naive power-aware approach first with our standby policy and then with nap, we see the minimal energy point shift toward the lower frequencies and become more compatible with expectations of DVS algorithms.

Recognizing 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 of a given task for a given power-aware memory policy (e.g. nap). Our approach requires information that is 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 design of the memory controller policy for transitioning chips between power states. An important aspect of these policies is determining which of the lower power states to enter when there are no accesses to service. This decision depends on the average gap between clusters of accesses. Simulation results show that as frequency increases from 50MHz to 1GHz, the policy of choice based on memory energy switches from immediately transitioning to the nap state to a policy that immediately transitions to standby

with the crossover point occurring where average gaps are approximately 56ns.

Our investigation offers insights into the factors governing the power-aware memory / DVS interactions and how aspects of each technology can enhance the effectiveness of the other. Complementary policies – DVS algorithms that include the effects of power-aware memory and memory controller policies that respond to frequency adjustments – offer the opportunity to fully exploit the energy benefits of both technologies.

## 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 and T. Mudge. Vertigo: Automatic performance setting for Linux. In *Symp. on Operating Systems Design and Implementation (OSDI)*, 2002.
- [8] 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.
- [9] 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.
- [10] 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.
- [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. MediaBench: 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 IpARM 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.