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

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 DVS algorithms are 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 (89% vs. 39% and 54%). 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.
Keywords: Power-Aware, Memory System, DVS, Control Policy, Synergy.

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 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: predicting future processing needs of the workload and 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 [28, 24, 23, 13, 9, 7, 26, 8], 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 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 [9, 7, 6, 19–21, 26]. 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 newer memory systems that offer their own power management features. Based on our simulation results, this paper makes the following contributions:

- 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. We discuss what it means to have an "energy-balanced" system design. The implication of a balanced system 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 explore how different power-aware memory control policies affect the frequency setting decision. The synergy between DVS and sophisticated power-aware memory goes 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 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 CPU frequency changes. 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.

The remainder of this paper is organized as follows. The next section discusses background and related work. Section 3 describes our research roadmap and 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. We conclude in Section 6.

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 [25, 13, 27, 11, 17] or regular intervals [28, 24, 9, 8] 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 [7, 6, 18, 26, 22]. 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 [19, 26, 9, 10] has suggested that memory effects should be taken into account. For computations that run to completion, Martin [19, 21] shows there is a lower bound on frequency such that any further slowing degrades a metric defined as the amount of computation that can be performed per battery discharge. In recognition of the interaction between CPU and memory, Hsu [10] proposes a compiling technique to identify program regions and set appropriate CPU frequencies for them. For periodic computations, Pouwelse [26] 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 [14, 3–5] 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. The lower the power consumption associated with a particular state, the higher the latency to service a new memory request. We adopt a three-state model consisting of *active*, *standby*, and *powerdown* states with power and latency values as shown in Table 1 (based on Infineon Mobile-RAM [12]). We use 90ns as the average delay of accessing a 32-byte cache block. Additional delay (denoted by the +) is incurred for clock resynchronization.

Table 1. Mobile-RAM Power State and Transition Values

Power State or Transition	**Power (mW)**	**Time (ns)**
Active	$P_a = 275$	$t_{access} \approx 90$
Standby	$P_s = 75$	-
Powerdown	$P_p = 1.75$	-
Stby $\rightarrow$ Act	-	$T_{s\rightarrow a} = 0$
Pdn $\rightarrow$ Act	$P_{p\rightarrow a} = 138$	$T_{p\rightarrow a} = +7.5$

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 the idle time between runs of accesses (which we refer to as *gaps*) and threshold values to trigger transitions. Fig. 1 shows how a policy that transitions among *active*, *standby*, and *powerdown* modes might work. When the memory has outstanding requests, the memory chip stays active. Upon the completion of servicing requests, the chip automatically goes to *standby*. Note there is no additional latency to transition back to *active* from *standby*. When the idle time, gap, exceeds a threshold (e.g., $gap_i > Th$), the chip transitions to *powerdown* and stays there until the start of the next access. For gaps shorter than the threshold (e.g., $gap_j < Th$), the memory remains in *standby*.

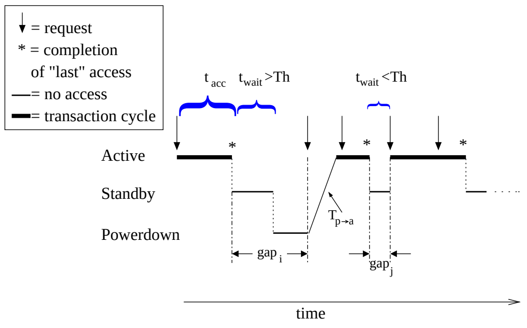


Fig. 1. Memory Access and Power Control

Operating system page allocation policies that place actively used pages in the minimum number of DRAM chips fully exploit the capabilities of power-aware memory. Previous studies [14] 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.

3 The Synergy between DVS and Power-Aware Memory

In this section, we first establish a roadmap for investigating the impact of DVS and power-aware memory on each other and how they can adapt to be mutually advantageous. Second we introduce our experimental environment and workload selection.

3.1 Roadmap

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. Since most DVS algorithms divide total execution time into task periods or episodes, through this paper all problems are discussed in the time domain of one task period.

We first consider a base case memory design in which the chips only transition between *active* and *standby*. It is the standard operating mode of most current DRAMs. We refer to this case as *standard memory*. A meager step in the direction of power-awareness is called *naive powerdown* and represents the policy in which the memory chips operate normally until task completion at which point they are powered down through the slack time to the end of the period. Next we explore memory controller policies that potentially transition a chip to a lower power state when it is otherwise not servicing request. This is done during the task's whole period (execution and slack time). The controller might wait for a threshold amount of time in *standby* before making the transition. According to previous research [14, 4, 5] and our experiments, the immediate transition and sequential page allocation represent "best practice". We refer to this policy as *immediate powerdown* or *aggressive*.

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.

3.2 Methodology

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 Mobile-RAM memory model including the power state transitions described in Section 2. We use two memory chips with a total capacity of 64MB. The variable voltage processor we simulate is based on Intel's XScale [16]. 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. The 1-level on-die cache is 32KB with 32B blocks and 32-way associativity. On average it takes about 90ns to service a cache miss without incurring the memory state transition delay.

As shown in Table 1, we use an SDRAM based memory model. In particular, we use close-page policy during the transition from $active$ to $standby$. Note data is not lost in the $powerdown$ state due to the refresh operation. We assume the impact of the refresh on the gap is minor since the refresh cycle is several orders of magnitude longer than the gap (ms vs. ns). For completeness, we also used a Rambus-DRAM based four-state model and obtained qualitatively similar results. Due to the similarity of results and the popularity of SDRAM based memory platforms (i.e., Mobile-RAM, DDR RAM, etc.), we only present results for the Mobile-RAM memory model.

We consider multimedia applications as representative workloads for low power mobile devices and because they appear to be amenable to good predictions of future processing demands on a per-task basis [26]. We have performed experiments using four applications from the MediaBench suite [15, 2]: MPEG2dec – an MPEG decoder, PEGWIT – a public key encryption program, G721 – voice compression, and RASTA – speech pre-processor. The results from these four benchmarks are remarkably consistent. Therefore we present results primarily for 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 can be fully explored.

To further explore those memory effects in a controlled fashion, we use a synthetic benchmark that can model a variety of computation times and cache miss ratios. 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 Standard Memory

We begin by taking standard Mobile-RAM as our base. The memory chip stays in *active* mode servicing requests and transitions to *standby* upon the completion of servicing requests. For our system configuration we have two chips, each consuming 275mW in the *active* state and 75mW in the *standby* state.

We consider the impact of such memory on the effectiveness of DVS for our MPEG decoding benchmark. We expect memory to dilute the impact of DVS on overall energy consumption. First for standard Mobile-RAM operated in *active* and *standby* states, memory energy consumption can be calculated as the sum of the energy consumed in *active* and that consumed in *standby*. Also because the number of accesses and

Table 2. DVS with Standard and Naive Powerdown Memory

							Standard		Naive	
CPU Freq (MHz)	CPU Pwr (mW)	Exec Time (ms)	Avg Gap0 (ns)	CPU Eng (mJ)	CPU Residue (mJ)	Mem Eng (mJ)	Mem Residue (mJ)	**Total Eng (mJ)**	Mem Residue (mJ)	**Total Eng (mJ)**
50	16.5	65.18	36446.5	1.08	0.00	9.81	0.12	**11.01**	0.00	**10.89**
100	38.3	32.63	18659.8	1.25	0.00	4.93	5.01	**11.18**	0.12	**6.30**
200	99.9	16.35	9487.3	1.63	0.01	2.48	7.45	**11.58**	0.17	**4.30**
400	311.0	8.22	4786.1	2.55	0.05	1.26	8.67	**12.53**	0.20	**4.07**
600	669.3	5.50	3199.3	3.68	0.10	0.86	9.07	**13.72**	0.21	**4.86**
800	1210.1	4.15	2413.4	5.02	0.19	0.65	9.28	**15.15**	0.22	**6.09**
1000	2354.7	3.34	2201.3	7.86	0.38	0.53	9.40	**18.17**	0.22	**8.99**

MPEG's period of 66ms are fixed, the memory energy consumption stays roughly constant (9.93mJ) as processor speed varies.

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 busy part of the period). The second portion (labeled "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 standby for standard memory).

From the data in Table 2 we see that for this standard 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 39%. These relative savings would be even lower if more DRAM chips were used (e.g., in a laptop with eight memory chips).

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 still consumes relatively high power (75mW) during the slack 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 slack 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 51% total energy savings without frequency scaling (i.e., comparing 8.99mJ to 18.17mJ at 1GHz) and a 68% 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 10.89mJ. In contrast, at 1GHz execution time does not decrease enough to offset the substantial increase in CPU power and total energy is 8.99mJ. The interesting point is that the lowest total energy consumption (4.07mJ) is achieved at 400MHz. 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.

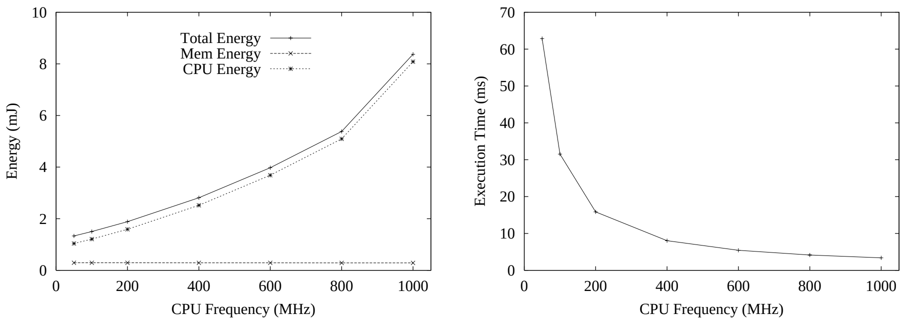
5.2 Dynamic Power-Aware Memory

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 busy.

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 busy portion of the task period. By default they all place the DRAM chips into powerdown for the slack portion of the task period. We begin by considering the behavior of the immediate powerdown (*aggressive*) 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 busy.

Fig. 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 *aggressive* 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 increases from 1.36mJ at 50MHz to 8.53mJ at 1GHz. These results illustrate that when aggressive memory power management is applied, CPU energy becomes dominant and traditional DVS begins to work as expected without having to exclude from consideration the energy consumption of memory. This behavior is explained by examining the processor



a) Energy b) Task Busy Time

Fig. 2. DVS and Power Aware Memory: MPEG Decode–Aggressive Policy

and memory energy components. Processor energy steadily increases quadratically with the increased voltage required for each higher frequency. In contrast, the total memory energy (busy and slack portion) stabilizes at around 0.28mJ (as explained in Sec 5.4).

Table 3. DVS and Power Aware Memory: MPEG Decode

CPU Freq (MHz)	CPU Pwr (mW)	Exec Time (ms)	Avg Gap0 (ns)	Mem Pwr (mW)	CPU Eng (mJ)	CPU Residue (mJ)	Mem Eng (mJ)	Mem Residue (mJ)	Total Eng (mJ)
50	16.5	65.18	36398.3	4.23	1.08	0.00	0.28	0.00	1.36
100	38.3	32.63	18641.7	4.94	1.25	0.00	0.16	0.12	1.53
200	99.9	16.36	9482.3	6.35	1.63	0.01	0.10	0.17	1.92
400	310.7	8.23	4782.6	9.13	2.56	0.05	0.08	0.20	2.88
600	668.1	5.52	3203.3	11.88	3.69	0.10	0.07	0.21	4.07
800	1207.1	4.16	2496.6	14.52	5.03	0.19	0.06	0.22	5.50
1000	2347.6	3.35	2541.4	16.66	7.87	0.38	0.06	0.22	8.53

From the discussion thus far, we can make several important observations. First, the naive implementation that powers down memory during slack portions of the period can produce lower overall energy consumption than a standard 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). Fig. 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 aggressive power management policy lowers the overall energy consumption, particularly at the lower frequencies. *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.*

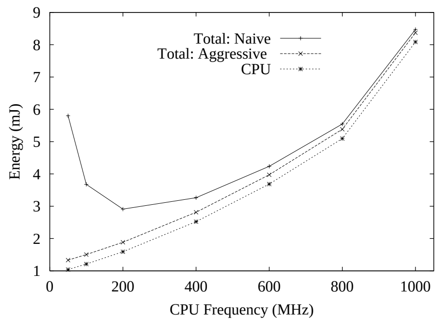


Fig. 3. DVS and Memory Controller Policies

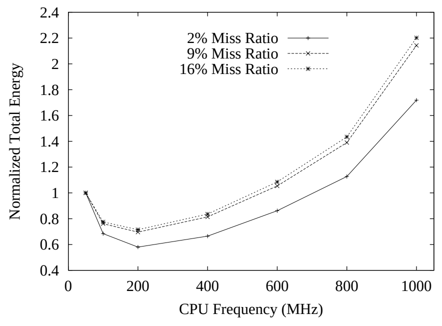
The final observation from the above discussion concerns the influence that limitations on execution time can have on energy consumption. For MPEG this was simply the linear effects of frequency changes versus the quadratic effects on power consumption. However, other benchmarks have other execution 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

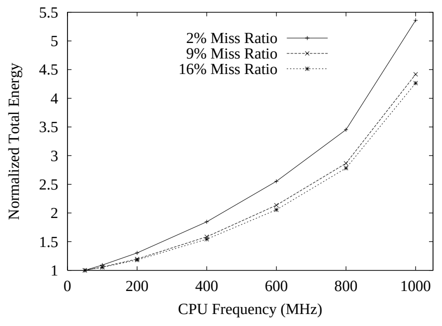
To explore the influence of memory latency and cache performance on DVS we consider the effect of changing the workload's miss ratio on voltage setting. Since it is hard to vary the miss ratio for a fixed cache configuration with real benchmarks, we use a synthetic benchmark to create three workloads with the same 30ms period but different miss ratios: 2%, 9% and 16%. The synthetic benchmark runs 1.0–1.7 million instructions. We manipulate the instruction number and the ratio of instruction type (computation/control/memory) to generate different miss ratios while maintaining the roughly equal execution time. 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. Fig. 4a) and 4b) show the total energy normalized to the 50MHz value for the naive and aggressive policy, respectively.

From Fig. 4 we see similar trends for each workload. For the naive policy (Fig. 4a), energy initially decreases, then increases dramatically as processor power increases. For the aggressive policy (Fig. 4b), energy increases as frequency increases. We note that for the aggressive policy, the total energy increases more rapidly for lower miss ratios. This is because as miss ratio increases memory energy increases, making CPU less dominant. Also, for a given miss ratio the total memory energy remains constant as



a) Naive Policy



b) Aggressive Policy

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

frequency varies, this additive factor makes total energy increase less rapidly for high miss ratio benchmarks than for low miss ratio ones. For the naive policy, however, we note that the overall energy energy decreases less rapidly and increases more rapidly for higher miss ratios. Firstly, because the *standby* power during execution is greater than the residue power during task slack time, memory energy does not remain constant but decreases as frequency increases, forming a trend opposite to CPU energy, thus making total energy a u-shape. Secondly, the execution time of higher miss ratio workloads are limited by memory latency sooner, making memory energy decrease less rapidly and reducing the benefit of increased clock frequencies.

These results indicate that DVS algorithms should 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

In this section, we show how to use available information to calculate component energy (CPU and memory) and total energy for each processor frequency level.

Estimating Energy. 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 [16] and represent the power associated with frequency f by $P_{cpu}(f)$. To calculate the execution time, we divide it into 3 parts according to the power state of the memory: T_{act}, $T_{L \rightarrow act}$ and T_L. We use L to denote one of the low power states. It can be $standby$ for naive power aware memory or $powerdown$ for aggressive power aware memory. T_{act} is the time spent in the $active$ power state where accesses are serviced. $T_{L \rightarrow act}$ is the extra time when memory is making transitions from the low power state to the $active$ state. Fig. 1 shows each transition is followed by an $active$ duration 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 transition, 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, our simulation results agree with this approximation. T_L is the sum of all durations when the CPU does not generate cache misses and the memory remains in the low power state. So each instruction, except those that trigger a cache miss, contributes a cycle to T_L.

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

$$T_{act} = t_{access} N_{misses} \tag{1}$$

$$T_{L \rightarrow act} = t_{L \rightarrow act} N_{misses} \tag{2}$$

$$T_L = \frac{1}{f}(N_{insts} - N_{misses}) \tag{3}$$

$$T_{exec} = T_{act} + T_{L \rightarrow act} + T_L \tag{4}$$

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

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

$$E_{mem} = T_{act} P_{act} + T_L P_L + T_{L \rightarrow act} P_{L \rightarrow act} + T_{residual} P_{powerdown} \tag{6}$$

$$E_{total} = E_{cpu} + E_{mem} \tag{7}$$

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

Evaluation. We use both synthetic and real workloads to evaluate our energy estimates. Fig. 5 shows the measured energy (Simulation) and our predicted energy (Predicted) versus clock frequency for PEGWIT.

The first observation 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. We note that the errors in CPU and total energy estimation are primarily due to the fact that the fixed CPU power values from [16] can not accurately reflect the actual 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

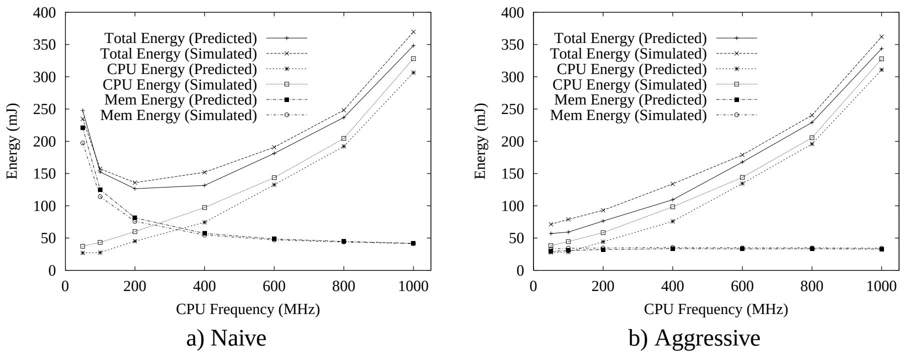


Fig. 5. Energy Prediction – Inorder Processor

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, leaving a side-effect of being more accurate (as illustrated in Fig. 6).

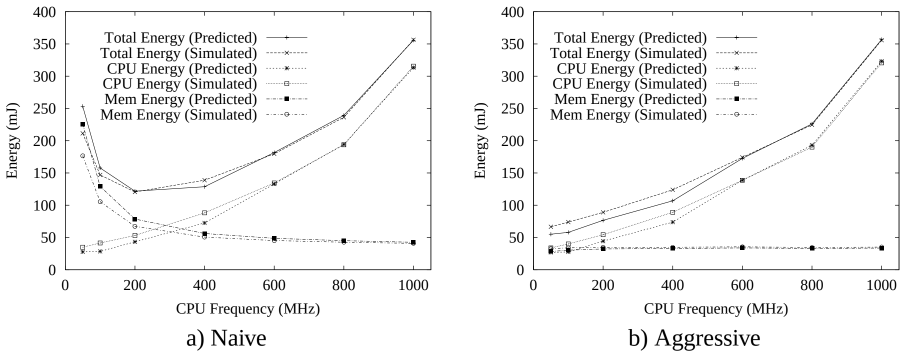


Fig. 6. Energy Prediction – Out-of-order Processor

6 Summary and Conclusions

This work shows there is a synergistic effect between dynamic voltage scaling (DVS) of the processor and power-aware memory control. 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 89% savings compared to our standard base case. By contrast, the energy savings from DVS alone with standard memory are only 39%. Using a power-aware memory policy that transitions into powerdown mode, but without exploiting the DVS capabilities of the processor, yields a total energy saving of 54%. The interaction between these two technologies has the greatest impact.

Traditional DVS works under the assumption that CPU power dominates. Unfortunately, in a common mobile system with standard memory and low power processor, memory power is comparable to CPU power, and thus dilutes the benefit from DVS. For applications with predictable periodicity, naive power management can be used to reduce task slack time energy. However, the memory energy scales with frequency and voltage differently from the CPU energy. Therefore the tradeoff between memory and processor energy has to be considered when setting the correct speed to minimize total energy. Aggressive power management further reduces memory energy so that CPU becomes dominant again. It makes the benefit from DVS more pronounced and the best speed setting becomes compatible with 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. 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.

As a natural extension to the above work, we also explored the impact of DVS on the memory access behavior and hence the selection of memory control policies. Due to the space limitation, we only give our conclusion here. If the DVS algorithm has to increase the frequency (i.e. to meet a deadline), the memory controller policy should adapt to the change of access behavior to gain maximum benefit (i.e. switch from an aggressive transition policy to a moderate one due to shortened gaps).

References

1. The SimpleScalar-Arm Power Modeling Project. //www.eecs.umich.edu/ jringenb/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. V. Delaluz, M. Kandemir, N. Vijaykrishnan, A. Sivasubramiam, and M.J. Irwin. DRAM energy management using software and hardware directed power mode control. In *7th Int'l Symp. on High Performance Computer Architecture*, January 2001.
4. X. Fan, C. S. Ellis, and A. R. Lebeck. Memory controller policies for DRAM power management. In *International Symp. on Low Power Electronics and Design (ISLPED)*, pages 129–134, August 2001.
5. X. Fan, C. S. Ellis, and A. R. Lebeck. Modeling of DRAM power control policies using deterministic and stochastic petri nets. In *Workshop on Power Aware Computing Systems*, February 2002.
6. K. Flautner and T. Mudge. Vertigo: Automatic performance setting for Linux. In *Symp. on Operating Systems Design and Implementation (OSDI)*, 2002.
7. K. Flautner, S. Reinhardt, and T. Mudge. Automatic performance setting for dynamic voltage scaling. In *7th Annual International Conference on Mobile Computing and Networking*, pages 260–271, 2001.

8. K. Govil, E. Chan, and H. Wasserman. Comparing algorithms for dynamic speed-setting of a low-power cpu. In *1st 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 *Symp. on Operating Systems Design and Implementation (OSDI)*, October 2000.
10. C. Hsu, U. Kremer, and M. Hsiao. Compiler-directed dynamic voltage/frequency scheduling for energy reduction in microprocessors. In *International Symp. on Low Power Electronics and Design (ISLPED)*, pages 275–278, August 2001.
11. C. Hughes, J. Srinivasan, and S. Adve. Saving energy with architectural and frequency adaptations for multimedia applications. In *34th International Symp. on Microarchitecture*, December 2001.
12. Infineon. *Mobile-RAM*, 2001. http://www.infineon.com/.
13. C. M. Krishna and Y-H. Lee. Voltage-clock-scaling techniques for low power in hard real-time systems. In *IEEE Real-Time Technology and Applications Symp.*, May 2000.
14. A. R. Lebeck, X. Fan, H. Zeng, and C. S. Ellis. Power aware page allocation. In *9th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS IX)*, pages 105–116, November 2000.
15. C. Lee, M. Potkonjak, and W. Mangione-Smith. Mediabench: A tool for evaluating and synthesizing multimedia and communications systems. In *30th International Symp. on Microarchitecture (MICRO 30)*, December 1997.
16. S. Leibson. XScale (StrongARM-2) Muscles In. *Microprocessor Report*, September 2000.
17. J. Lorch and A. J. Smith. Operating system modifications for task-based speed and voltage scheduling. In *1st International Conf. on Mobile Systems, Applications, and Services*, May 2003.
18. J. R. Lorch and Alan Jay Smith. Improving dynamic voltage scaling algorithms with pace. In *Joint International Conference on Measurement and Modeling of Computer Systems*, 2001.
19. T. Martin. Balancing batteries, power and performance: system issues in cpu speed-setting for mobile computing. In *PhD thesis, Carnegie Mellon University*, 1999.
20. 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.
21. T. L. Martin, D. P. Siewiorek, and J. M. Warren. A cpu speed-setting policy that accounts for nonideal memory and battery properties. In *39th Power Sources Conf.*, June 2000.
22. 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.
23. T. Pering, T. Burd, and R. Brodersen. Voltage scheduling in the lpARM microprocessor system. In *International Symp. on Low Power Electronics and Design (ISLPED)*, 2000.
24. Trevor Pering, Thomas D. Burd, and Robert W. Brodersen. The simulation and evaluation of dynamic scaling algorithms. In *International Symp. on Low Power Electronics and Design (ISLPED)*, August 1998.
25. P. Pillai and K. G. Shin. Real-time dynamic voltage scaling for low-power embedded operating systems. In *18th Symp. on Operating Systems Principles*, October 2001.
26. 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.
27. V. Swaminathan and K. Chakrabarty. Real-time task scheduling for energy-aware embedded systems. In *IEEE Real-Time Systems Symp. (Work-in-Progress Session)*, November 2000.
28. M. Weiser, B. Welch, A. Demers, and S. Shenker. Scheduling for reduced CPU energy. In *1st Symp. on Operating Systems Design and Implementation (OSDI)*, November 1994.