POWER AWARE MEMORY SYSTEM

by

Xiaobo Fan

Department of Computer Science
Duke University

Date: _____

Approved:

_____
Carla S. Ellis, Supervisor

_____
Alvin R. Lebeck, Supervisor

_____
Daniel J. Sorin

_____
Robert A. Wagner

_____

Dissertation submitted in partial fulfillment of the
requirements for the degree of Doctor of Philosophy
in the Department of Computer Science
in the Graduate School of
Duke University

2004

ABSTRACT

(Computer Science)

POWER AWARE MEMORY SYSTEM

by

Xiaobo Fan

Department of Computer Science
Duke University

Date: _____

Approved:

_____
Carla S. Ellis, Supervisor

_____
Alvin R. Lebeck, Supervisor

_____
Daniel J. Sorin

_____
Robert A. Wagner

_____

An abstract of a dissertation submitted in partial fulfillment of the
requirements for the degree of Doctor of Philosophy
in the Department of Computer Science
in the Graduate School of
Duke University

2004

Abstract

Main memory is becoming an important target for energy optimization. Memory technology is becoming available that offers power management features. We address the problem of memory energy optimization in several levels and stages. First, in the software level, we revisit traditional virtual memory page allocation and propose a power aware page allocation scheme to complement the hardware power management strategies. We explore the interaction of page placement with static and dynamic hardware policies. Using both trace-drive and execution-driven simulations, our results make a compelling case for a cooperative hardware/software approach for exploiting memory power management features.

Second, we investigate what the appropriate hardware control policy should be given a set of hardware power states. A device with power management features can be transitioned down to a low power state to save energy when it is not in use. Only when the time of staying in low power is long enough, can the energy saving cover the transition cost. Power management makes decisions on if and when to make transitions during each idle period. Without perfect knowledge about the access pattern, the power manager has to make a guess on the upcoming idle time. However, single value prediction fails to capture the uncertainty of the access patterns; probabilistic modeling fails to capture the variety and non-stationarity of the access patterns. While identifying the limitations of these approaches, we propose a probability-based online approach for adapting to access patterns and making decisions efficiently. Simulation results show that our strategy saves more energy than other available strategies, and is general and efficient enough to be applied to various devices.

Finally, in recognization that Dynamic Voltage Scaling (DVS) has been studied

extensively to optimize CPU power consumption while not considering other components' contribution, we explore the interactions between memory and processor voltage scaling. The results indicate that effective CPU speed settings should take into account the information of memory access behavior.

Acknowledgements

This work is under the joint supervision of Dr. Carla S. Ellis and Dr. Alvin R. Lebeck. I am deeply indebted to both of them for their guidance, support and patience, which will benefit me in my lifetime.

I am thankful to Dr. Dan Sorin and Dr. Robert Wagner for serving on my committee and giving me valuable comments. Also to the colleagues at Duke, in particular, Tong Li and Heng Zeng, for the helpful discussions and feedbacks.

Finally I appreciate all the support and encouragement my wife, Ying, gave me during the whole course, all the joyful moments my baby girl, Sophie, brought me during the final writing stage, and all the endeavors my parents made to raise and educate me since I was born.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation

Energy efficiency is becoming increasingly important in system design. Mobile devices need to reduce power consumption to extend battery lifetime. Even for servers and desktops, ever increased power consumption causes heat dissipation and cooling problem and begins to limit performance improvement. Energy efficiency is also desirable in all computing platforms from the economic and environmental points of view.

Recent years have seen many system components being targeted for energy optimization. Dynamic voltage scaling [90, 75, 74, 46, 27, 22, 77, 26] has been proposed to reduce CPU power consumption. Disk spin-down policies [16, 17, 29, 47, 55] have been devised to save energy consumption of hard disks. Main memory—one major component in computer systems—has not received enough research attention in terms of energy optimization. With the introduction of low power processors and novel displays, main memory is consuming a growing proportion of the system power budget. This percentage can increase dramatically in mobile devices that have no secondary storage and rely on memory to retain data, i.e. 50% for a pocket computer [20] versus 9% for a conventional laptop. Therefore, memory system energy efficiency starts to become an important target in system design.

The Memory industry is meeting this demand by making memory chips with multiple power states and thus offering power management possibilities. Besides a single power mode in conventional DRAM, there are several different lower power

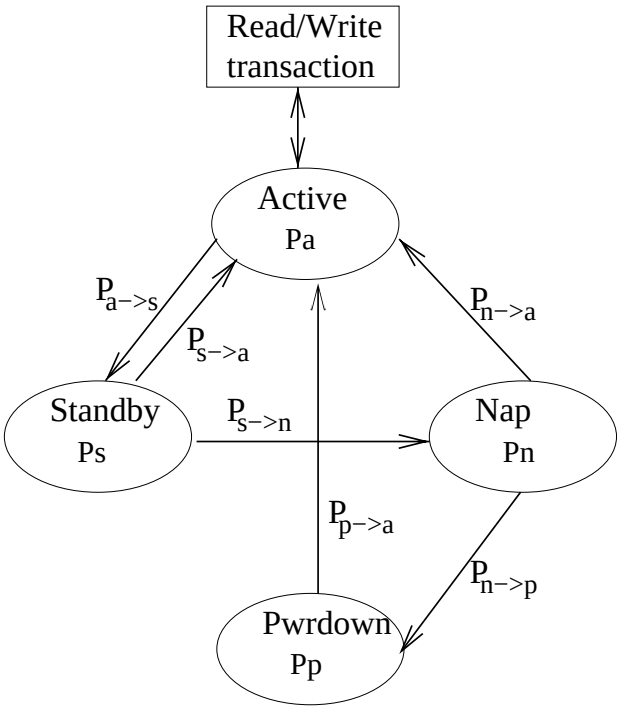


Figure 1.1: RDRAM Power States and Transitions

states. One concrete example is Direct Rambus DRAM (RDRAM) technology [81] that provides four power states: active, standby, nap and powerdown, with power consumption in decreasing order and access latency in increasing order. Figure 1.1 illustrates the power states transitions of one RDRAM chip. All read/write transactions have to be performed in the *active* state. Thus additional latency has to be spent to resynchronize the chip to *active* if the memory access is directed to a low power chip. Lower power states incur longer resynchronization. Table 1.1 shows a group of typical power and latency values based on the RDRAM specification [81]. Another feature of RDRAM that makes power management possible is its narrow bus topology and high operating frequency. One single RDRAM chip can deliver the same or even higher bandwidth than multiple interleaved conventional chips. Therefore each chip can be set to appropriate power state independently.

Traditional DRAM manufacturers are also entering this arena by lowering SDRAMs'

Power State or Transition	**Power** (mW)	**Time** (nS)
Active	$P_a = 300$	$t_{acc} = 60$
Standby	$P_s = 180$	-
Nap	$P_n = 30$	-
Powerdown	$P_p = 3$	-
Standby $\rightarrow$ Active	$P_{s \rightarrow a} = 240$	$T_{s \rightarrow a} = +6$
Nap $\rightarrow$ Active	$P_{n \rightarrow a} = 165$	$T_{n \rightarrow a} = +120$
Powerdown $\rightarrow$ Active	$P_{p \rightarrow a} = 152$	$T_{p \rightarrow a} = +25,000$

Table 1.1: RDRAM Power State and Transition Values. All accesses incur the 60ns active access time. Additional delay (denoted by the +) is incurred for clock resynchronization.

base power consumption and equipping them with power management features. The advantage of taking this path is that the new memory is compatible with the huge base of the SDRAM-based platforms, and to exploit the power management feature only small portion of the memory controller needs to be changed. Samsung, Infineon and Micron all rolled out a series of SDRAM-based low power memory [84, 36, 65] targeted at next-generation mobile systems and cellular handsets. Figure 1.2 illustrates the power states and transitions of the Infineon Mobile-RAM and Table 1.2 shows its power and latency parameters.

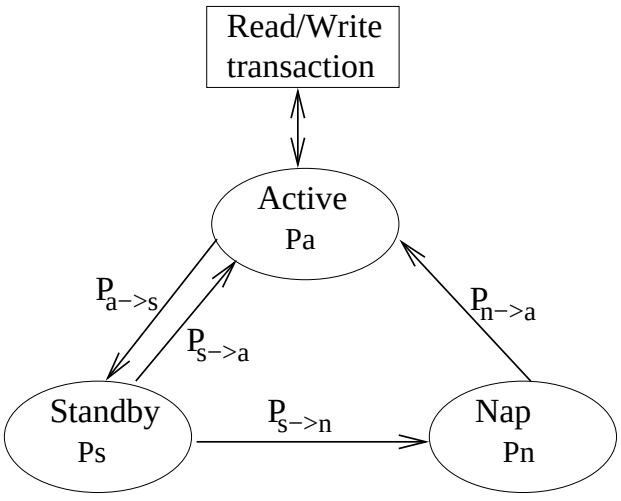


Figure 1.2: Mobile-RAM Power States and Transitions

Power State or Transition	**Power** (mW)	**Time** (nS)
Active	$P_a = 275$	$t_{acc} \approx 90$
Standby	$P_s = 75$	-
Nap [1]	$P_n = 1.75$	-
Standby $\rightarrow$ Active	-	$T_{s \rightarrow a} = 0$
Nap $\rightarrow$ Active	$P_{n \rightarrow a} = 138$	$T_{n \rightarrow a} = +7.5$

Table 1.2: Mobile-RAM Power State and Transition Values

The challenge for computer system designers is how to utilize the power management feature of the hardware and exploit the trade-off between power and performance. The goal is to develop a power aware memory system with improved energy efficiency and acceptable performance.

1.2 Design Space

A Computer system is a complex hierarchical system. The memory subsystem is no exception. We need to address the energy optimization problem from different perspectives. Figure 1.3 shows a two-dimension design space. In the operating system level, virtual addresses are mapped into physical addresses by the page allocation algorithm. In particular for RDRAM with no interleaving, the physical address directly relates to a certain memory chip. Traditional virtual memory systems are not aware of the physical chip location associated with each memory access, and thus usually exhibit random access behavior. To fully exploit the hardware's power management feature, we need to re-examine virtual memory page allocation policies. Sequential first-touch placement policy allocates physical memory pages in a sequential order, filling up one chip before moving on to the next. As a result, a minimum number of chips are occupied by one application. Energy can be saved by simply turning off the

[1] This state is actually called *Powerdown* in Infineon Mobile-RAM. To avoid confusion with RDRAM, we use *nap* to refer to it.

other unoccupied chips, and potentially by the reduction of amortized access cost due to more clustered accesses. To determine the benefits of power aware page allocation we compare different placement policies in Chapter 3. Our results show that sequential page allocation can dramatically improve energy efficiency of the main memory. We also explore the possibility of other more complex page allocation policies.

Hardware control, as shown in Figure 1.3, constitutes the other dimension of the design space. It can be implemented in the memory controller, which performs power state transitions for each individual memory module based on the observed memory access behavior. To evaluate different page allocation schemes in the software dimension, we use a simple static policy and a threshold-based dynamic policy.

However, choosing the appropriate threshold for the dynamic policy is still done experimentally. An effective power management strategy should capture and adapt to the dynamics of the access patterns. We first investigate single value prediction. Because of the uncertainty of the program behavior, it is very difficult to chase the variation with a single value. Next we try to analytically model access patterns with probability distributions. Due to the complexity of the program behavior, in most cases they don't fit in with known distributions. As a result, we take an empirical approach for learning access patterns and making power management decisions on the fly.

We use a sliding window to capture the recent access pattern, and use a tree structure to record the energy consumption associated with each candidate threshold. As the window slides, the tree structure can be efficiently updated to reflect the up-to-date energy consumptions and find the optimal threshold. Since we use a single metric c, the break-even point, to capture the power characteristics of a device, our proposed strategy can be naturally extended to a whole class of devices that share the similar state-based power management feature, which we refer to as *non-*

scalable devices. We compare our strategy with different other strategies on three types of devices–wireless card, memory and disk. The simulation results show that our strategy adapts to various devices and workloads, and approaches the optimal strategy.

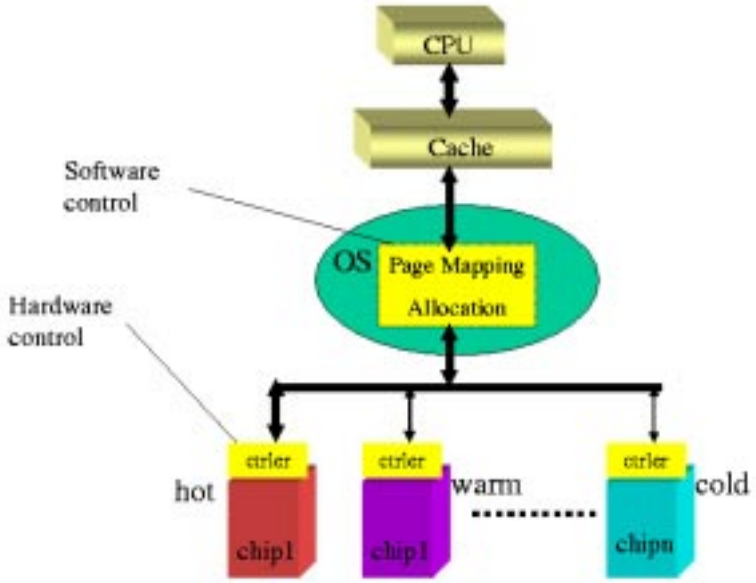


Figure 1.3: Two Dimensions of Control Policies

1.3 Interactions of Power Aware Memory System and Processor Voltage Scaling

In the last part of my research, I investigate the interactions of power-aware memory system and voltage scaled processors. Dynamic Voltage Scaling (DVS) is the technique for exploiting the performance/power tradeoff of the processor whereby the operating system is responsible for selecting an appropriate clock frequency in

response to applications' dynamic CPU utilization. CMOS circuits have the characteristics that the power consumption changes linearly with frequency and quadratically with voltage. Also because the required voltage lowers as running frequency decreases, energy could be saved when the processor frequency is scaled down. Most DVS algorithms try to predict applications' performance requirement and scale the processor to the lowest speed that satisfies applications' need in order to minimize the processor's energy consumption. However, the power and performance contributions from other system components, in particular memory, invalidate some of the simple assumptions upon which most DVS algorithms have been based. Lower processor speed incurs longer execution time and hence more power consumption on memory, which could dominate the total energy and decrease the overall efficiency. Thus the lowest speed might not be optimal considering the memory effect. On the other hand, power aware memory has to be adaptable to the variations of memory access behavior due to the change of processor frequency.

We explore the nature of these interactions between power aware memory and processor voltage scaling. We use a synthetic benchmark to simulate a typical multimedia workload. Power and performance configurations are acquired from Intel's newest scalable processor Xscale [54] and Infineon Mobile-RAM [36]. Our results show that without applying power control policy memory dominates total energy and CPU part is negligible. Applying control policy on power aware memory makes the power consumptions of memory and processor comparable and their interactions interesting. Our simulations of executions with different CPU speeds show that at low frequency settings, memory dominates overall energy. As frequency increases, execution time decreases, initially reducing overall energy. However, for reasonable miss ratios, further increases in frequency reduce execution time very little while processor power consumption increases dramatically. To capture this tradeoff between mem-

ory and processor energy for determining an appropriate voltage/frequency setting, we develop a technique to estimate overall energy consumption using information available from existing performance counters (i.e., instructions executed, memory references, misses). We show that our estimator is sufficient to capture the general trend in overall energy as frequency changes.

The rest of the document is organized as follows. Chapter 3 discusses different page allocation policies and their effect on energy efficiency. In Chapter 4, we investigate memory access behavior, and develop and evaluate an online power management strategy. Chapter 5 presents our investigation of the interactions of power aware memory and processor voltage scaling. Finally, we conclude in Chapter 6 and describe future work.

Chapter 2

Related Work

2.1 Architectural and OS-level Related Work

A consortium of companies has developed a specification [37] that addresses the lower-level OS/device interface, providing one model for gross system-wide power states and per-component power states as a basis for the development of OS-directed power management [93]. Recent work with Odyssey [69, 23] demonstrates how system support for application-aware adaptation can benefit energy efficiency.

Another related area involves operating system page placement policies. Virtual memory page research originally concentrated on techniques for improving program execution time, focusing on replacement algorithms. Recent studies examined *page coloring* policies for selecting appropriate physical page frames to minimize cache misses [4, 43]. Other recent work has studied page placement aimed at improving TLB performance [82] or NUMA multiprocessor memory access [49, 50, 89, 24, 6].

Architectural studies have examined the impact of software structure on power consumption [12, 64, 88]. Other architectural studies investigated processor design [8, 60, 75], focused specifically on the memory hierarchy [28, 30, 40, 68], or examined ways to optimize DRAM refresh counts [71]. However, to our knowledge, ours is the first quantitative study to explore the interaction of page allocation with power aware hardware to orchestrate the use of power modes being provided in emerging memory devices.

Follow-up work includes migrating array data into same banks to exploit more temporal locality [13] and clustering similar pages into small number of nodes in a

multi-task OS environment [32]. Other studies [80, 10, 31] bear the similar idea of grouping temporally affined accesses and degrading ununsed parts to save total cost.

2.2 Device Level Power Management

Previous studies focusing on component power management include work on scheduling for low power processor modes [57, 58, 90], spindown policies for disks and alternatives [3, 15, 16, 17, 29, 47, 55, 91], and managing wireless communication [35, 45, 86]. Common themes that appear in many power management strategies are the identification and prediction of idleness in the activity patterns of a component.

One approach is to summarize the previous samples and predict the future occurrence using a single value. For DRAM power management, Delaluz et al. [14] show potential benefits of an adaptive policy that attempts to predict the time between consecutive accesses in a cache-less memory system as a basis for deciding when to make transitions. Lu et al. [59] use *session* to describe the active period of a hard disk, and based on the prediction of session length make spin-down decisions. Other studies [11, 33] present similar prediction schemes and try to apply them to a broader range of devices. Techniques proposed in these approaches implicitly assume a smoothly changing length of idleness/activity, which usually does not hold for most real access patterns, which nevertheless exhibit statistical stability.

In contrast to the single value prediction, probabilistic approaches try to predict the future access pattern using probability distributions. One direction of studies [18, 19, 72, 85, 79, 78] is to use known functions to model the distribution so that the optimal policy can be solved analytically. These approaches make strong assumption that device access patterns can be characterized by a family of known distributions, which doesn't hold in most cases. They also incur relatively high computation cost and are thus not suitable for making an online policy.

The other set of studies make no assumption about the distribution and learn it by simply accumulating the historical data. Histogram is used in [56, 92] to probabilistically estimate processor demand and improve traditional DVS algorithm. Histogram is also used in [47, 2, 39] to capture idle times of a hard disk and a wireless card. Despite the limitations, which will be addressed in detail in Section 4.2.2, these studies provide valuable reference for designing a general and efficient online power management strategy.

2.3 Dynamic Voltage Scaling

The earliest DVS algorithms fall into the category of interval-based algorithms [90, 75, 27, 26]. The goal of the algorithm is to monitor processor load and to set the clock speed just slow enough to spread the computation over the interval and squeeze out any idle time. Experience with this class of algorithms suggests that they may be effective for well-behaved, regular workloads. However, bursty or irregular behavior causes serious difficulties in the accurate prediction of future needs. The clock speeds tend to oscillate without stabilizing on a good point or else are very slow to react and lag behind needed transitions [27]. Subsequent work has attempted to improve the prediction aspect of interval-based DVS algorithms by trying to detect patterns and anticipate bursts [26].

The other distinct category of DVS algorithms integrates scaling into task-oriented, deadline-based, real time schedulers (e.g., earliest deadline first or rate monotonic) [76, 46, 87]. These algorithms assume knowledge about the real time task set, including the worst case execution time (WCET) of each task and the period/deadline. The goal is to determine the speed setting such that computation is spread out over the period while never missing a deadline. These algorithms tend to begin with the determination of a schedule using the WCET, assume performance scales linearly with

frequency, and then satisfy schedulability tests.

Recent work that falls somewhere in between the hard real time and the interval-based categories acknowledges the need for more semantic information about the workload to address the prediction issues. Flautner et al. [22] derive execution episodes by examining communication patterns. Pouwelse et al. [77] suggest using application-specific information supplied by power-aware applications (e.g., frame type and size for a video decoder). Mosse [67] proposes explicit compiler assistance.

The speed-setting decision has appeared to be the more straightforward, given good predictions. However recent experimental work [61, 77, 27] has shown that the linear performance assumption does not hold when memory is taken into account. For computations that run to completion, Martin [61, 63] 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 [77] alludes to the problem that the high cost of memory, extended over the whole period, may dominate the overall energy consumption of a system such that even effective DVS of the CPU delivers marginal benefit. This complication of the DVS problem is the focus of our work.

Chapter 3

Power Aware Page Allocation

In this chapter, we re-visit virtual memory page allocation polices in light of multi-state DRAM. Given hardware support that can determine when to transition between power states, the operating system may further improve energy efficiency by allocating physical pages in a manner that fully exploits the hardware. Since traditional virtual memory system is not aware of the memory chip configuration, we believe that it would allocate physical pages randomly. As a first step, page allocation should cluster one application's pages into the minimum number of DRAM chips. We use sequential first-touch to fulfill this. To be able to control each chip independently, we assume that physical addresses are not interleaved across DRAM chips. We can interleave across multiple internal banks inside each chip.

Using power aware page allocation, energy is saved in two dimensions: 1) unoccupied chips are created and turned off and 2) access locality is increased in allocated chips. The first dimension is easily achieved by sequential allocation. However, it may not create maximum subsequent locality since it captures only an initialization phase of the program. Thus, we also consider the potential for limited reassignment intended to cluster pages with similar access patterns within Rambus DRAM chips. The Frequency policy attempts to improve upon an initial allocation of frequently accessed pages at some point into the execution. Identification of candidates for reassignment is done with small per-page hardware counters, recording frequency of accesses to each page, outside of the L1 and L2 caches, over a window of time. A limited number of the most frequently accessed pages are then moved into a common chip. In our formulation of this scheme, a block of free page frames is reserved in

one chip during initial placement to serve as a destination during this later one-time reallocation.

Related work in page allocation area includes page coloring policies for reducing cache misses [4, 43], and page placement aimed at improving TLB performance [82]. They all attempt to exploit the benefit by leveraging the flexibility of virtual-physical address mapping. In this sense, our approach shows some resemblance to theirs. Next we present our experiment methods and discuss results.

3.1 Hardware Power Management

This section explains various hardware policies for controlling DRAM power states. Since each chip is controlled independently, the memory controller can implement a variety of power management policies. In this paper we investigate two types of policies: static and dynamic.

3.1.1 Static Policies

The static schemes we investigate correspond to placing all DRAM chips in a single power state. We note that for an access to occur, the DRAM chip must first transition to the active state. Only when there are no outstanding requests for the device does it return to the specified static power state. Our first static policy assumes that all DRAM devices are in the active state. This corresponds to a conventional performance oriented design, targeted at reducing execution time.

The next three static schemes place all DRAM chips in the standby, nap, and powerdown state, respectively, when there are no accesses to service. These policies correspond to implementations targeting energy efficiency by sacrificing performance, since the memory access time increases as the power consumption is reduced. Ideally, we want to maximize performance while minimizing energy consumption. The

remainder of this section describes policies with this goal.

3.1.2 Dynamic Policies

To obtain higher performance and energy efficiency we must relax the constraint that each DRAM chip return to the same base power state when there are no pending accesses. This allows the possibility of exploiting locality in the program's memory access pattern to reduce energy consumption. To accomplish this, we need to dynamically determine the power state of each chip. Clearly, a chip needs to be in the active state to perform an access. The more difficult decision is to determine when the chip should transition to a lower power state.

Our approach uses the time between accesses to a chip as a metric for transitioning to lower power states. If a chip is not accessed for a threshold amount of time it transitions to the next lower power state. This allows individual chips to reside in different power states, based on their individual access patterns.

The threshold values are an important parameter in this approach. Too large a threshold and the chip will spend too much time in the higher power state, increasing energy consumption. In contrast, if the threshold is too small, then the chip will transition into a slower, but lower power state, increasing execution time. In Section 3.4, we employ a worse-case based competitive analysis to determine a ballpark of the candidate threshold values. A more thorough investigation of hardware policies will be presented in Chapter 4.

3.2 Page Allocation

The focus of this chapter is to re-examine virtual memory page allocation policies in light of new DRAM technology. Previous page allocation studies ignored which actual DRAM chips contained the allocated page frame. In contrast, our work fo-

cuses specifically on this parameter in an effort to maximize energy efficiency. Given hardware mechanisms, as described above, that can determine when to transition between power states, the operating system may further improve energy efficiency by allocating physical pages in a manner that fully exploits the hardware. As a first step, the page allocation should cluster an application's pages into the minimum number of DRAM chips.

To determine the benefits of power aware page allocation (see Section 3.4) we compare random and sequential first-touch placement policies. Our first policy randomly chooses a DRAM chip for the physical page. We believe that the allocation policies in conventional operating systems would appear to be essentially a random assignment with respect to chip selection.

The sequential first-touch policy allocates pages in the order they are accessed, filling an entire DRAM chip before moving on to the next. This scheme minimizes the number of DRAM chips utilized for a given application. Therefore, the hardware can automatically place unused DRAMs in the powerdown state, and hence potentially reduce energy consumption.

This new form of page coloring targets reducing power consumption rather than improving performance. However, we note that conventional page coloring for improved cache performance can still be utilized when selecting pages from within a DRAM chip. We also assume that physical addresses are not interleaved across DRAM chips. We can interleave at the word, cache line, or page granularity within the DRAM chip, since each chip will likely contain multiple independent banks.

Finally, experience has shown that first-touch is often not representative of subsequent locality since it may capture only an initialization phase of the program. Thus, we also consider the potential for limited reassignment intended to cluster pages with similar access patterns within DRAM chips. The Frequency policy attempts to im-

prove upon an initial allocation of frequently accessed pages at some point into the execution. Identification of candidates for reassignment is done with small per-page hardware counters, recording frequency of accesses to each page, outside of the L1 and L2 caches, over a window of time. A limited number of the most frequently accessed pages are then moved into a common chip. In our formulation of this scheme, a block of free page frames is reserved in one chip during initial placement to serve as a destination during this later one-time reallocation. Of course, this could be repeated, but we leave multiple "corrections" as future work.

In Section 3.4, an offline version (counting over the entire trace and then placing pages accordingly) is first considered in order to ascertain that "better" placements are possible using such frequency information. Then the online policy, described above, is simulated, including the costs of page migration.

3.3 Methodology

To evaluate energy efficiency, we use the *energy-delay* product [25]. This metric captures our goal of achieving high performance (seconds) while minimizing energy consumption (joules).

To compute energy efficiency, we developed two simulators: a trace-driven simulator and a detailed execution-driven out-of-order processor simulator. One of the primary considerations that went into our experimental design was the choice of a workload that would seem appropriate to mobile/wireless devices. The availability of traces from a set of popular applications used on laptops motivated the development of our trace-driven simulator. While these traces satisfied the need for a representative workload for the target environment, they had disadvantages for memory research: low miss rates and the constraints of trace-driven simulation (e.g., no detailed processor timing). Thus, the execution-driven simulator was developed to address the

need for a more detailed processor/memory model and more memory-intensive benchmarks.

3.3.1 Trace-Driven Simulation

The trace-driven simulator processes instruction and data address traces and uses a simplified Rambus DRAM model. This simulator models a two-level cache hierarchy with a 16KB, direct-mapped level one cache and a 256KB direct-mapped second-level cache, both caches have 32B cache blocks. We also model the individual Rambus DRAM chips and their associated power state. Each cache is lockup-free and can have up to eight outstanding misses. In this simulator, we do not model memory bus contention or the internal DRAM banks. Instead we optimistically assume all requests to a single Rambus DRAM can be overlapped (i.e., no bank conflicts). In these studies we only model the transition from the lower power state to *active*. The transitions from active to lower power states are assumed to incur no delay or energy consumption. These assumptions are removed in our execution-driven simulator.

To drive our simulator we use instruction traces from personal productivity applications executing on an Intel processor with Microsoft Windows NT. These traces, provided by the University of Washington Etch project [53], include instruction and data accesses for several popular applications typical of those used on laptops today. Table 3.1 provides information on the applications we use. The first six benchmarks are from the NT traces.

For timing considerations (necessary to compute energy consumption), we use a simplified processor model that executes one instruction per cycle, and never stalls due to long latency operations (i.e., execution only stalls when the maximum number of outstanding misses is reached). We assume a 500Mhz processor clock, the level one cache takes 2 cycles to access, while the level two cache incurs an additional 10

	Benchmark	Description	Insts Exec'ed (Millions)	Size (MB)
Trace Driven	acrord32	Adobe Acrobat Reader 3.0 PDF file reader.	408	9.73
	compress	SPEC95 version of Unix compress utility.	403	0.849
	go	SPEC95 version of game go.	315	1.05
	netscape	Netscape Navigator 3.1 web browser.	92	9.95
	powerpnt	Microsoft PowerPoint 7.0b slide preparation package.	209	12.5
	winword	Microsoft Word 7.0 word processor	351	11.2
Exec Driven	bzip	SPEC2000 compression.	100	180
	compress	SPEC95 version of Unix compress utility.	100	32
	go	SPEC95 version of game go.	100	1
	gcc	SPEC2000 compiler.	100	32
	vpr	SPEC2000 FPGA placement and routing.	100	37

Table 3.1: Benchmarks

cycles. We simulate a non-interleaved main memory system with eight 32Mb DRAM chips, for a total main memory capacity of 32MB.

3.3.2 Execution-Driven Simulation

To overcome the limitations of trace-driven simulation, we augmented the SimpleScalar execution-driven simulator [9] with a Rambus DRAM model based on the detailed timing and power specifications of Rambus RDRAM. We use a 400Mhz 8-issue processor that can have up to 256 active instructions and 128 memory operations. The first-level cache is 32KB, 4-way set-associative with 32B blocks, while the second-level cache is 256KB, direct-mapped, with 64B blocks. Each cache can have up to 16 outstanding misses. The simulator executes Alpha binaries.

Our Rambus DRAM model uses the values from Table 1.1, but includes further details, such as multiple banks per chip, open page and close page policies, and various interleaving strategies for mapping physical addresses to specific chips and

banks within chips. This simulator provides a more accurate model of timing at all levels of the memory hierarchy, including contention at each level and within each Rambus device and transitions from higher to lower power states. In particular, active to either nap or powerdown takes 8 cycles, standby to nap takes 12 cycles, nap to powerdown takes 61 cycles because we must first enter the active state. Active to standby either takes 1 cycle or 73 cycles, depending on the DRAM page mode. We simulate a non-interleaved main memory system with eight 256Mb chips for a total capacity of 256MB.

Due to excessive simulation time, we fast-forward the simulator over the first 4 billion instructions, and then simulate in detail the next 100 million committed instructions. This allows us to skip over program initialization, however page placement is based on accesses from the beginning of program execution (during the fast-forwarding). In addition to the two SPEC95 benchmarks for which NT traces also exist (compress and go, above), we use three integer programs from the SPEC2000 suite (bzip, gcc, and vpr) for our execution-driven analysis (described at the bottom of Table 3.1). These three were chosen because they exhibited the highest data cache miss ratios. For all benchmarks, we use the reference input data set.

3.4 Experimental Results

This section presents our results on power management for DRAM. We begin with analysis of static hardware power state policies and their interaction with page allocation (Section 3.4.1). This is followed in Section 3.4.2 by analysis of the effects of page allocation on dynamic hardware power management policies. We then investigate the effects of open/close DRAM page policies and interleaving on energy efficiency.

The main results from this study are:

1. Cooperative hardware and software for power aware page allocation can improve main memory energy efficiency, measured in terms of Energy•Delay, by 6% to 55% over the best static policy.

2. Nap mode is the most energy efficient static policy for our applications.

3. Power aware page allocation without dynamic hardware support can improve energy efficiency by up to 30% over static nap, depending on application characteristics.

4. Dynamic hardware schemes do not improve energy efficiency for random page allocation.

3.4.1 Static Power State Policies

In this section we evaluate the static policies that uniformly place all DRAM chips in the same power state. We note that for an access to occur, the DRAM chip must first transition to the active state. Only when there are no outstanding requests for the device does it return to the specified static power state. Our first static policy assumes that all DRAM devices are in the active state. This corresponds to a conventional performance oriented design, targeted at reducing execution time. The next three static schemes place all DRAM chips in the standby, nap, and powerdown state, respectively, when there are no accesses to service. These policies correspond to implementations targeting energy efficiency by sacrificing performance, since the memory access time increases as the power consumption is reduced.

We begin by evaluating DRAM power management techniques in the context of random physical page allocation. In other words, the operating system is oblivious to the power management capabilities of the underlying hardware.

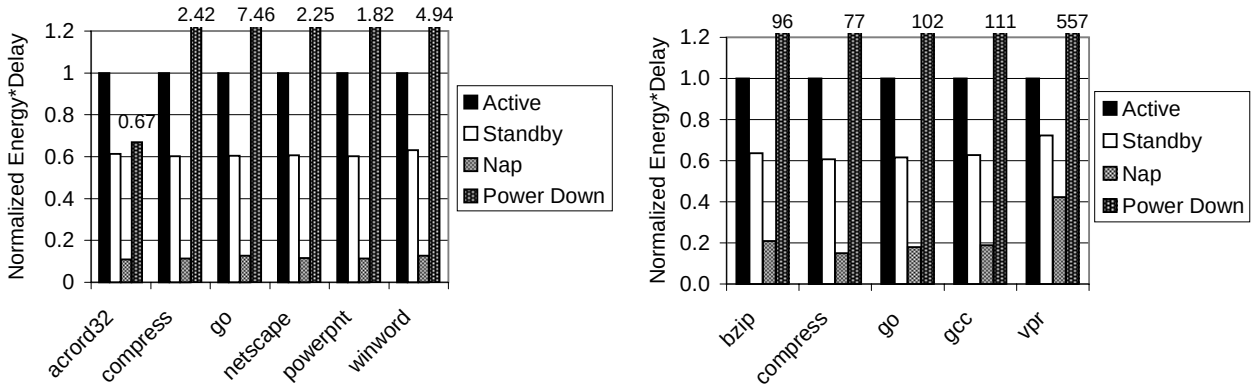


Figure 3.1: Static Policies and Random Page Allocation: Energy•Delay normalized to *active* policy

Figure 3.1 shows the Energy•Delay for the four static policies (*active*, *standby*, *nap*, and *powerdown*) normalized to the *active* policy for each program. Table 3.2 shows the absolute values for runtime, energy, and Energy•Delay product. From Figure 3.1 we see that placing all DRAM chips in the *nap* state provides the lowest Energy•Delay product for all applications in both simulations. *Nap* achieves approximately 15% of the Energy•Delay of *active* for the trace-driven simulations, while it achieves 20% to 40% of *active* for the execution-driven results. *Powerdown* is generally the poorest performing, followed by *active*.

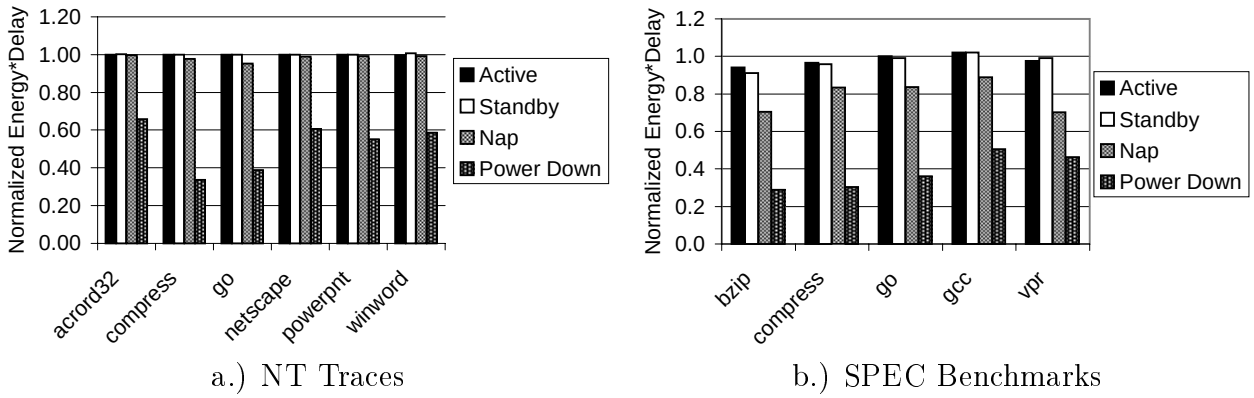
These results match our expectations, since *powerdown* incurs a significant increase in access delay, while *active* consumes too much energy when it is not servicing requests. The notable exception is acrord32, where *powerdown* is better than *active*. This is due to the low rate at which acrord32 generates DRAM accesses. From Tables 3.1 and 3.2 we see that acrord32 has the lowest rate of DRAM accesses. Therefore, it still achieves energy savings even though its delay increases. We also note that the Energy•Delay of *powerdown* is directly related to the rate at which benchmarks generate DRAM accesses.

Standby is the next best mode after *nap* achieving 60% of *active* in the trace-driven simulations, and 60% to 70% of *active* in the execution-driven simulations. *Standby* is worse than *nap* because the additional time penalty of *nap* causes only a slight increase in total run time, while the power reductions are very large (30mW vs 180mW).

We note that the relative Energy-Delay values for *active*, *standby*, and *nap* follow the relative ratios of power consumption. This is particularly true for the trace-driven simulations, and is a direct result of the low L2 miss rates exhibited by those programs (< 1%). The extremely high time penalty of *powerdown* is too much for even these low miss rates, and Energy-Delay increases dramatically.

Impact of Page Allocation

We now examine the benefits of sequential-first-touch page allocation over random page allocation for the static hardware power management schemes. Figure 3.2 shows the Energy•Delay of sequential allocation normalized to the Energy•Delay of random allocation. From Figure 3.2a we see that page allocation has very little effect on energy efficiency for *active*, *standby*, or *nap*, using the trace-driven simulations, producing at most a 6% reduction for *nap* (go). For these policies with random allocation, each chip consumes near its minimum energy because the programs have very low miss ratios. Packing all the program's pages into the minimum number of chips reduces the unused chips' energy by very little, which is offset by the increase in energy consumption for the more utilized chips. We note that sequential page allocation dramatically improves the energy efficiency for the *powerdown* static policy, achieving 30% to 70% of the random allocation. This is because the delay to transition out of *powerdown* is extremely long, and consumes a significant amount of energy. When program text and data are packed into the minimum number of



a.) NT Traces b.) SPEC Benchmarks

Figure 3.2: Benefits of Sequential Page Allocation for Static Policies: each Energy•Delay result normalized to same policy with Random Allocation

chips, each chip is likely to statisfy more requests when it reaches the active state than when pages are randomly spread across chips. This observation is supported by our data that shows an increase in the number of references that occur when the target chip is already in the active state.

In contrast to the trace-driven results, our execution-driven results (see Figure 3.2b) show that power aware page allocation does improve energy efficiency for the *nap* policy by 12% to 30%. In particular, we note that compress and go, the SPEC95 benchmarks, show larger improvements than those observed in the trace-driven experiments. This is due in part because first-touch produces lower L2 cache miss ratios and part to the more detailed processor model used by SimpleScalar. Recall, our trace-driven simulator does not model data dependencies or finite processor resources, which minimizes the effects of long latency operations. Our execution-driven simulator accurately models these constraints and the corresponding additional delays when long latency operations cause resources (e.g., instruction buffers) to be overcommitted. Finally, as before, we see very little improvement for *active* and *standby*, while *powerdown* benefits the most from sequential page allocation.

3.4.2 Dynamic Power State Management

We now examine more sophisticated hardware support for DRAM power management. By dynamically determining each chip's power state based on recent references, we hope to improve overall energy efficiency. Figure 3.3 shows the Energy•Delay of various dynamic policies normalized to the static *nap* policy for our trace-driven simulations using sequential first-touch allocation. Each bar in the graph represents a different set of thresholds (in nano-seconds) for transitioning from *active* to *nap* (x) and from *nap* to *powerdown* (y), represented as x/y. Table 3.3 shows the raw data for the static nap and powerdown schemes along with the best dynamic scheme.

Choosing Threshold Values

Before discussing the results, first we introduce how we choose several sets of candidate thresholds. We determine a loose lower bound on the time required to be spent in a lower power state in order to overcome the transition costs by analytically computing the penalty vs. reward for transitioning to the lower power state. By competitive analysis [7], choosing this bound as the threshold performs at most twice bad as the optimal policy. Therefore, we use that bound to guide the choice of threshold values to explore.

Assume T_p, T_n and T_s are the times of staying in *powerdown*, *nap* and *standby* to improve the $E \bullet D$ product. Assume P_p, P_n, P_s and P_a are per-chip power consumption for *powerdown*, *nap*, *standby* and *active* states respectively; $P_{p\to a}$, $P_{n\to a}$ and $P_{s\to a}$ are transition (resynchronization) power consumption from *powerdown*, *nap* and *standby* to *active*; $T_{p\to a}$, $T_{n\to a}$ and $T_{s\to a}$ are transition (resynchronization) time from *powerdown*, *nap* and *standby* to *active*.

We observe a single independent transition– *active* $\to$ *powerdown*. Assume E_0 and T_0 are the original energy consumption and run time without power state tran-

sition. We have

$$E_{new} \bullet D_{new} = (E_0 + T_{p\to a}P_{p\to a} - (P_a - P_p)T_p) \bullet (T_0 + T_{p\to a})$$

In order for $E_{new} \bullet D_{new} < E_0 \bullet T_0$

$$T_p > \frac{E_0 T_{p\to a} + T_0 T_{p\to a} P_{p\to a} + T_{p\to a}^2 P_{p\to a}}{(P_a - P_p)(T_0 + T_{p\to a})}$$

because $T_{p\to a} << T_0$, and $T_{p\to a}P_{p\to a} << E_0$, and the *active* power is $P_0 = \frac{E_0}{T_0}$, we have

$$T_p > \frac{P_{p\to a} + P_0}{P_a - P_p} T_{p\to a}$$

Now we pick $\frac{P_{p\to a}+P_0}{P_a-P_p}T_{p\to a}$ as the lower bound of our time threshold between accesses to a chip for transitioning from *active* to *powerdown*. Similarly, we have

$$T_n > \frac{P_{n\to a} + P_0}{P_a - P_n} T_{n\to a}$$

$$T_s > \frac{P_{s\to a} + P_0}{P_a - P_s} T_{s\to a}$$

Substituting variables with the values from our RDRAM configuration in Table 1.1, we have $T_p > 9131ns$, $T_n > 103ns$, $T_s > 27ns$.

As we can see from the above lower bounds, T_n is in the same magnitude as T_s, and considering the extra hardware overhead of *standby* state, we don't remain in *standby*. The threshold of *active* to *nap* should be in magnitude of $10^2 ns$, and *active* to *powerdown* in $10^4 ns$.

Discussion

From Figure 3.3 we see that the combination of power aware page allocation and dynamic hardware policies can produce Energy•Delay values that are 50% to 94% of the static *nap* policy. Five of the six benchmarks achieve 80% or lower.

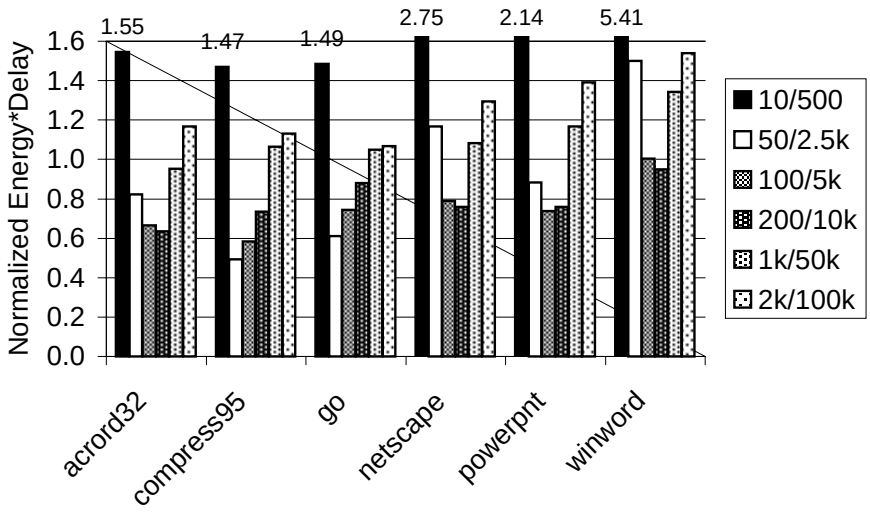


Figure 3.3: Dynamic Power Management and Sequential Page Allocation (NT Traces): normalized to Static *nap* policy. Each threshold combination for transitioning from *active* to *nap* (x ns) and from *nap* to *powerdown* (y ns) is represented as x/y.

Our execution-driven results show that dynamic hardware policies improve energy efficiency of sequential page allocation by 42% for bzip, 43% for compress, 50% for go, 55% for gcc, and 30% for vpr over static *nap*, the best static policy. Furthermore, this is an overall improvement of 50% to 60% compared to static *nap* using random page allocation. Due to excessive simulation time we did not perform as exhaustive of an evaluation as with the trace-driven studies. The best results in the limited experiments we did perform are produced by threshold values of 0ns between *active* and *standby*, 2,000ns between *standby* and *nap*, and 50,000ns between *nap* and *powerdown*. While further improvements might be achieved by fine tuning the thresholds, these results are sufficient to show that cooperative hardware/software techniques can improve energy efficiency.

The energy efficiency of sequential-first-touch page allocation and dynamic hardware power state management is significantly better than using a traditional full-power memory system, e.g., static *active*. Our cooperative hardware/software schemes achieve from 7% to 20% of the Energy•Delay for the static *active* policy for the

execution-driven simulations and from 1% to 10% of the Energy•Delay for the trace-driven experiments.

An important observation from our simulation results is that dynamic power state management does not improve energy efficiency for random page allocation over the static *nap* policy. In particular, for the execution-driven experiments above, the dynamic policies with random placement are over an order of magnitude worse than the static *nap* policy for two of the benchmarks. This poor performance is a result of moving to the *powerdown* state too soon, and incurring the large delay and corresponding energy consumption to transition out of *powerdown*. This overhead can be reduced by increasing the *nap* to *powerdown* threshold, and thus preventing any chip from entering *powerdown*. We verified this behavior through simulation, and achieved energy efficiency comparable to the static *nap* policy. Further tuning of the other thresholds produced only minor benefits. We also note that for sequential page allocation, the higher *powerdown* thresholds do not significantly change the results from those presented above. This is important since we want the dynamic polices to produce comparable results to the static schemes in cases where the operating system is unable to successfully perform power aware page allocation.

3.4.3 Frequency-based Page Placement

The primary goal of page placement thus far has been to cluster all pages into the minimum number of DRAM chips. In this section, we present results from an alternative placement technique that further refines page allocation based on access frequency. To achieve this, we first construct a histogram of page accesses offline. The results of this profile run are then used to determine initial page placement, starting with the most frequently accessed page and continuing to the least accessed.

Figure 3.4 shows the Energy•Delay for both the frequency and sequential first-

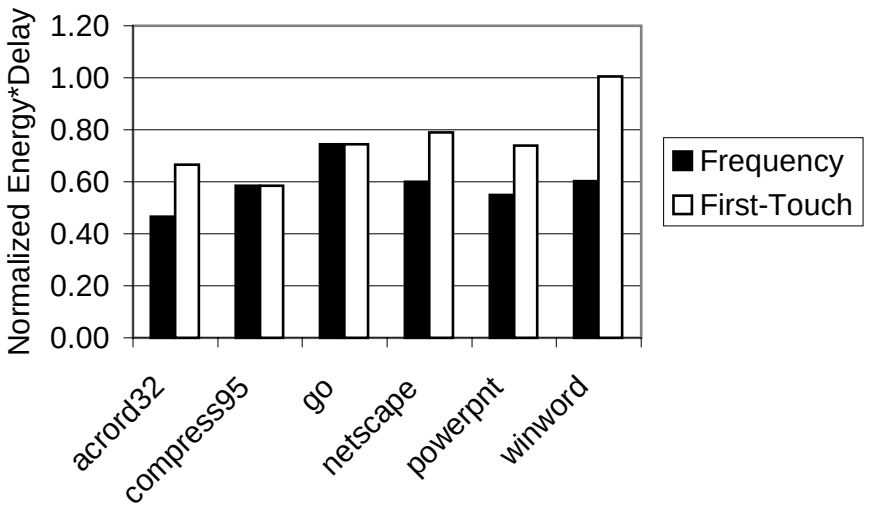


Figure 3.4: Frequency vs. Sequential First-Touch Page Allocation (NT Traces) for Dynamic Policy, thresholds of 100ns/5,000ns, normalized to sequential first-touch static *nap*.

touch allocation policies and the dynamic hardware policy with thresholds of 100ns/5,000ns normalized to sequential first-touch static *nap*. These results clearly show that first-touch is not the best placement policy. Compress and go do not show any benefit since they both fit entirely on a single chip. Acrord32, netscape, and powerpoint all reduce the Energy•Delay by approximately 20% beyond the values achieved by first-touch. Winword exhibits the largest benefit of frequency based placement, achieving 60% of the static *nap* value, whereas first-touch did not improve energy efficiency at all.

We are currently investigating online techniques to reassign pages based on reference frequency. Our initial implementation reserves 128 physical pages in chip 0, reallocates the 128 most frequently accessed pages from the other chips to chip 0, and then packs the remaining pages into the smallest number of chips. We execute the program for a 100ms warmup period to skip initialization, and then sample page accesses for 2ms. We associate a 10-bit saturating counter with each physical page, and increment the appropriate counter for each page accessed during the sample pe-

riod. At the end of the sample period, the OS sorts the counters and performs the movement and repacking operations, and resumes program execution. We include the cost of page moves as 0.011ms and 0.008mJ, obtained by measuring the energy and delay of a bcopy using our execution-driven simulator.

The above implementation produces a 10% reduction in Energy•Delay for winword, the program with the most opportunity, over static *nap*. This is because winword is a long running program that accesses a large amount of memory. We did not see any improvement for the other programs. The other programs either do not run very long or do not stress the memory system much. Furthermore, the other programs achieve significant gains from first-touch, while winword does not. We are currently investigating other applications and other, less hardware intensive, techniques for obtaining page reference frequency. However, we note that conventional page reference counting may not directly apply since large L2 caches can filter many accesses, whereas it is L2 misses that dictate DRAM access frequency.

3.4.4 Alternative DRAM Architectures

In this section we examine the effects of two important DRAM architectural alternatives: DRAM page policy and interleaving.

Open vs. Close Page Policy

The previous execution-driven results use a 0ns threshold for transitioning from *active* to *standby*. This is a result of the detailed DRAM model used in those simulations. Most current DRAM devices support two operating modes: open page and close page. These modes indicate what occurs after the DRAM services a request. In open page mode, data from a DRAM page[1] remains on the sense amplifiers in anticipation

[1] A DRAM page is one row of an internal DRAM bank.

of future accesses to nearby data. However, subsequent accesses to a different DRAM page incur an additional *precharge* delay before fetching the appropriate DRAM page. In contrast, close page mode immediately precharges the DRAM bank after an access in an attempt to avoid the precharge delay. If the same DRAM page is accessed, it incurs higher delay than the open page technique, since the data must be fetched again.

The DRAM page policy relates to power management in that an important difference between the *active* and *standby* power states is whether there is data on the sense amplifiers. To enter *standby* all pages must be closed (i.e. precharge all sense amplifiers). Furthermore, the resynchronization delay of *standby* applies only to the column address and data bus and can be completely overlapped with the row activate command. Therefore, in close page mode, when there are no requests to any banks of a chip, it enters *standby* (0ns threshold from *active*), since this will not introduce any additional delay, but can reduce energy consumption. This is the policy used to obtain the previous results. In open page mode, a device can remain in the *active* state while it retains data on the sense amplifiers. The threshold for transitioning to *standby* determines when all DRAM pages on a device should be closed. For our configuration, there is also an additional 73 cycle delay incurred to issue appropriate commands to close the open DRAM pages.

We use the execution-driven simulator to evaluate the impact of open vs. close page modes on energy efficiency. The trace-driven simulator does not model the DRAM devices in sufficient detail to perform this study. We use the dynamic hardware policies with sequential first-touch page allocation and non-interleaved main memory. Our simulations show that close page mode produces Energy•Delay values 20% lower than open page mode.

Interleaving

The results thus far do not use any interleaving; physical addresses are mapped sequentially to each chip, so chip 0 contains physical pages 0 to N-1, chip 1 contains N to 2N-1, etc. However, we do interleave cache blocks across internal DRAM banks.[2] This allows sequential cache block accesses within a page to overlap much of their DRAM latency.

Alternatively, we may get higher performance if we can spread pages across DRAM chips, potentially exposing more parallelism by reducing DRAM bank conflicts. For example, we could interleave at the page granularity, such that physical pages are allocated in a round-robin manner across chips (e.g., page 0 to chip 0, page 1 to chip 1, etc.). While this may reduce execution time, it forces many chips to be active, similar to random page allocation. The operating system could still pack pages into the minimum number of DRAM chips, but that produces the same DRAM access pattern as no interleaving. It also has the additional disadvantage of potentially using only a subset of large physically indexed caches.

Execution-driven simulation results reveal that page-grain interleaving produces Energy•Delay values close to random page allocation, as expected. Further experiments that vary the cache block interleaving within a DRAM chip reveal no significant differences among alternatives.

[2]RDRAM has 32 internal banks, a maximum of 16 can be accessed in parallel.

Benchmark	**Policy**	**DRAM Accesses**	**Run Time** (ms)	**Energy** (mJ)	**Energy•Delay** (Joule•second)
acrord32	Active	2142947	821.60	1971.84	1.6201
	Standby	2144482	830.79	1194.39	0.9923
	Nap	2140029	836.07	211.85	0.1771
	Power Down	2138783	1811.26	598.93	1.0848
compress95	Active	3460487	805.80	1933.92	1.5584
	Standby	3460256	806.23	1162.98	0.9376
	Nap	3450740	807.27	221.22	0.1786
	Power Down	3446425	2465.30	1526.98	3.7645
go	Active	5306343	631.66	1515.98	0.9576
	Standby	5326190	632.35	913.66	0.5778
	Nap	5307126	635.93	192.41	0.1224
	Power Down	5397678	3626.58	1970.60	7.1465
netscape	Active	927654	186.06	446.54	0.0831
	Standby	935725	187.15	269.80	0.0505
	Nap	928205	189.73	50.58	0.0096
	Power Down	932157	695.25	268.59	0.1867
powerpnt	Active	1890243	419.17	1006.01	0.4217
	Standby	1888144	420.00	605.71	0.2544
	Nap	1895739	422.83	113.25	0.0479
	Power Down	1786613	1382.50	554.91	0.7672
winword	Active	6186470	762.86	1830.87	1.3967
	Standby	6202047	786.38	1120.78	0.8814
	Nap	6185615	805.36	221.68	0.1785
	Power Down	6180983	3926.53	1757.27	6.9000
bzip	Active	493496	124.82	299.58	0.0374
	Standby	493351	125.58	189.17	0.0238
	Nap	493333	138.78	56.40	0.0078
	Power Down	491150	3647.10	982.09	3.5818
compress	Active	171871	115.09	276.21	0.0318
	Standby	171869	115.11	168.06	0.0193
	Nap	171886	125.24	38.16	0.0048
	Power Down	171823	3524.68	698.08	2.4605
go	Active	624912	260.96	626.31	0.1634
	Standby	625651	261.41	385.24	0.1007
	Nap	630775	293.45	99.39	0.0292
	Power Down	631931	8264.94	2021.63	16.7086
gcc	Active	335303	99.78	239.48	0.0239
	Standby	335262	100.57	149.20	0.0150
	Nap	335813	112.48	39.73	0.0045
	Power Down	335857	3406.77	777.38	2.6483
vpr	Active	2268727	227.14	545.14	0.1238
	Standby	2266131	232.11	385.02	0.0894
	Nap	2271568	271.83	191.98	0.0522
	Power Down	2272356	13211.38	5216.70	68.9199

Table 3.2: Raw Data for Static Policies with Random Allocation

Benchmark	**Policy**	**Run Time** (ms)	**Energy** (mJ)	**Energy•Delay** (Joule•second)
acrord32	Nap	837.69	211.11	0.1768
	Power Down	1860.70	383.34	0.7133
	Dynamic 100ns/5,000ns	915.84	128.70	0.1179
compress95	Nap	807.33	216.26	0.1746
	Power Down	2767.50	457.89	1.2672
	Dynamic 100ns/5,000ns	805.96	126.56	0.1020
go	Nap	636.90	183.14	0.1166
	Power Down	4064.43	682.03	2.7721
	Dynamic 100ns/5,000ns	638.39	136.02	0.0868
netscape	Nap	190.03	49.89	0.0095
	Power Down	724.81	155.85	0.1130
	Dynamic 100ns/5,000ns	212.54	35.25	0.0075
powerpnt	Nap	423.28	112.16	0.0475
	Power Down	1432.95	295.56	0.4235
	Dynamic 100ns/5,000ns	453.05	77.43	0.0351
winword	Nap	809.23	218.86	0.1771
	Power Down	4039.94	1000.87	4.0435
	Dynamic 100ns/5,000ns	911.53	195.14	0.1779
bzip	Nap	132.77	41.61	0.0055
	Power Down	3037.55	340.82	1.0353
	Dynamic 2,000ns/50,000ns	122.85	25.95	0.0032
compress	Nap	121.06	32.91	0.0040
	Power Down	2560.44	291.30	0.7458
	Dynamic 2,000ns/50,000ns	115.60	20.21	0.0023
go	Nap	286.51	85.29	0.0244
	Power Down	6785.82	890.64	6.0437
	Dynamic 2,000ns/50,000ns	262.26	45.80	0.0120
gcc	Nap	111.84	36.02	0.0040
	Power Down	2987.83	447.47	1.3370
	Dynamic 2,000ns/50,000ns	101.33	18.89	0.0019
vpr	Nap	273.39	134.26	0.0367
	Power Down	12199.88	2618.03	31.9396
	Dynamic 2,000ns/50,000ns	231.88	112.27	0.0260

Table 3.3: Raw Data for Static Policies and Best Dynamic Policy with Sequential Allocation

Chapter 4

Device Level Power Management

4.1 Introduction

As we have seen from Chapter 3 and our previous work [51], simple static policy and dynamic threshold-based policy are used to control power state transitions. However, the optimal threshold out of the candidate pool is chosen offline by experimentally testing all candidate thresholds. The resulting threshold is workload-specific. Therefore, to make device level power management feasible and effective, we need to find an online and adaptive solution.

The goal of the work in this chapter is to design a dynamic power management strategy that adapts to various access patterns and device characteristics, and is simple and efficient enough to be implemented as an online control policy. Furthermore, initiated by memory, our study is conducted by taking into consideration a class of devices (i.e. disk, network card, etc.) that share the same power management attributes with memory. As a result, our strategy is general enough to be applied to a variety of devices.

In general, there are two classes of devices with multiple power states: scalable and non-scalable. A non-scalable device (i.e. memory, disk, network card, etc.) functions at a high power state but can be transitioned down to a low power state when not in use. Therefore, the only opportunity for it to reduce power consumption is during the idle time when there is no work pending. When in low power state, it has to be powered up before starting to function again. This transition incurs extra energy cost.

By contrast, the CPU can function at all levels of power states although possibly at a degraded speed. We call it a scalable device. Because of the non-linear relation between the power consumption and speed of the CPU, it can be scaled down to a lower speed and save energy. Assuming it is possible to predict the workload demand and its deadline, there may be little or no impact on performance as we can stretch the work to the level where it can still be finished in time [73, 22, 77, 46, 56, 92].

My work is focused on non-scalable devices. For this class of devices, the goal is to predict the length of the upcoming idle time and try to make worthy transitions by comparing the cost and benefit of the transition. The problem seems to be trivial. However, several challenges exist in designing an effective and practical power management strategy.

- Due to the net energy cost incurred by a wrong transition, the prediction accuracy has to be sufficiently high to achieve an overall energy saving.
- The prediction of idle time and the decision making of power state transition have to be performed in a very efficient way since the power management is done in real time.
- It is desirable to have a power management strategy that is general to a variety of hardware devices and adaptive to various access patterns, so that a universal module can be implemented in the computer system to manage all necessary devices.

This chapter addresses the above challenges.

There has been an extensive set of research on the power management of non-scalable devices. In general, there are two approaches depending on how to predict the idle time: single value prediction and probability distribution prediction. Single

value prediction assumes a smooth variation between adjacent idle times so that the upcoming idle time can be estimated to a single value by summarizing the historical data. In reality, however, individual idle times do not exhibit stability on single values. On the contrary, as a whole they exhibit stability on some probability distributions. Using distribution to describe the upcoming idle time bypasses the problem of inter-idle-time uncertainty and thus is presumably more effective. Therefore, we take the second approach.

Knowing the probability distribution of the idle time, we can always compute an optimal threshold for keeping the device in high power before making the power transition. Probabilistically, using this threshold incurs the minimum energy cost.

When it comes to the learning and representation of the distribution, there are two kinds of approaches: parametric and non-parametric. The parametric approach tries to formulate the distribution using a known family of functions so that the optimization problem can be solved analytically. However, the realistic access pattern may be a mixture of many distributions and can change over time. In most cases, it is impossible to match it to a known function. In addition, the relatively high computation cost also limits the usability of the parametric modeling in an online environment.

Therefore, we take an empirical approach for learning the idle time distribution and optimizing the power control threshold. We make no assumptions about the idle time distribution. We simply collect idle time samples, and for each sample we update a data structure to reflect its contribution to the distribution and choose from a candidate threshold pool the one which incurs the least energy consumption on the current distribution of samples. In this chapter, we make the following primary contributions:

- By showing the non-stationarity of the device access pattern, we justify the

necessity of using a window to keep the recent idle time samples and age the old ones.

- We propose an algorithm for efficiently updating the energy cost for the candidate threshold pool and outputing the optimal threshold.
- We present a guideline for choosing a range for the candidate threshold and offer a proof to validate its correctness.
- We use simulation to evaluate different power management strategies on three hardware devices. The results show that our strategy performs consistently better than others and approaches the optimal.

The rest of the chapter is organized as follows. The next section discusses background and related work, and elaborates our motivation. Section 4.3 presents our power management strategy and discusses the related issues. We evaluate our strategy in Section 4.4 and conclude in Section 4.5.

4.2 Background & Problem

Figure 4.1 illustrates how a threshold-based strategy works, where σ is the threshold value, p_{act} is the power consumption of the active state when the device is servicing requests, p_{high} is the high power consumption, p_{low} the low power consumption, p_{up} and t_{up} are the power consumption and delay for switching up and p_{down} and t_{down} are those for switching down. Note there is no delay for switching between active and high power state. Basically they are the same state except that active state incurs additional power consumption for servicing requests. Separating them is just for a discussion convenience.

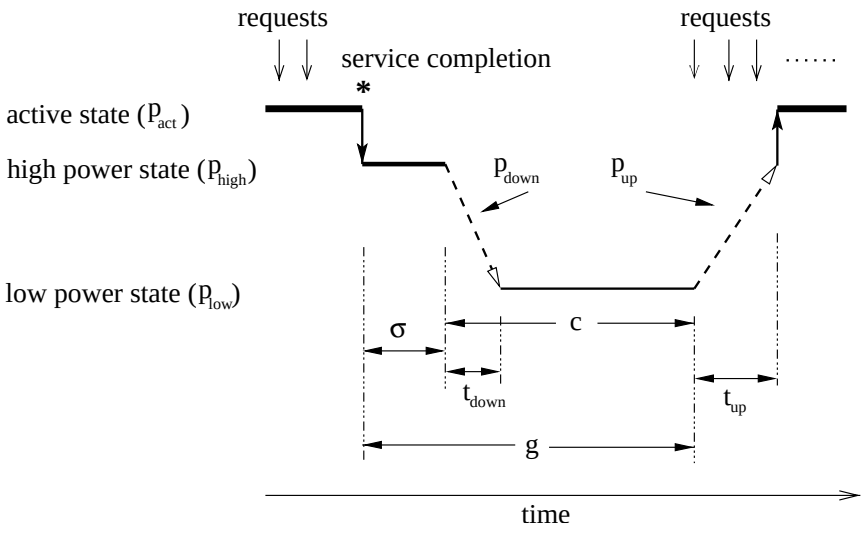


Figure 4.1: Power State Transitions

Dynamic power management tries to save energy by transitioning the device to a low power state when it is idle. We define *gap* as the time when the device stays idle, which starts right after it finishes servicing the current request queue and ends just before the next request comes. Since there is extra energy cost incurred when switching between power states, the time for staying in the low power state has to be long enough to cover the transition cost. Based on Figure 4.1, the break-even point (throughout this chapter we denote it by c) can be computed as follows:

$$Energy(make - transition) = Energy(stay - active)$$

$$\Leftrightarrow\ p_{down}t_{down} + p_{low}(c - t_{down}) + p_{up}t_{up} = p_{high}c$$

$$\Leftrightarrow\ c = \frac{(p_{down} - p_{low})t_{down} + p_{up}t_{up}}{p_{high} - p_{low}} \tag{4.1}$$

Given a certain device, c reflects its power characteristics and is a constant. A Power Manager (PM) tries to acquire the most accurate information about the upcoming gap and compare it with c to make power transition decisions. Therefore, the knowledge of gap directly affects the effectiveness of the power management. One reasonable assumption can be made here: PM can speculate the future gaps

by analyzing the historical gaps. Usually there are two categories of strategies for speculating the future: single value prediction and probability-based prediction. This section discusses these two techniques.

4.2.1 Single Value Prediction

Many prediction schemes [11, 14, 33, 59] have been proposed to predict the gap length using past gaps. They use a single value to characterize the upcoming gap. If this predicted value is greater than c, the component is transitioned down immediately; otherwise, it stays in the high power state. Because an underestimation wastes energy on high power and an overestimation wastes energy on unnecessary transition, the prediction hit rate determines the energy savings. Therefore, these schemes assume a smoothly changing gap and all the information about the next gap can be described with a single value. They work only on certain access patterns with smoothly changing gaps. For access patterns that exhibit fluctuating behavior, a prediction-based approach could incur very high energy cost due to the increased mis-prediction rate.

Exponentially averaging the past gaps to estimate the future gap is the most commonly used predictive technique. The recursive function is $e_{i+1} = f \cdot e_i + (1-f) \cdot g_i$, where e_{i+1} is the new estimation, e_i is the previous estimation, g_i is the previous gap and f is the preserving factor. Figure 4.2 shows how prediction works on a real disk access trace [83]. The X-axis is the gap sequence number representing a group of consecutive gaps, and the Y-axis is the gap length. The straight line labeled c is the break-even gap length. There are three prediction lines using exponential averaging with different preserving factors. There is one prediction line using uniform averaging. At each time instance, if the predicted gap length falls in the same side of c as the "Real Gap" length, it is counted as one prediction hit since the decision on power

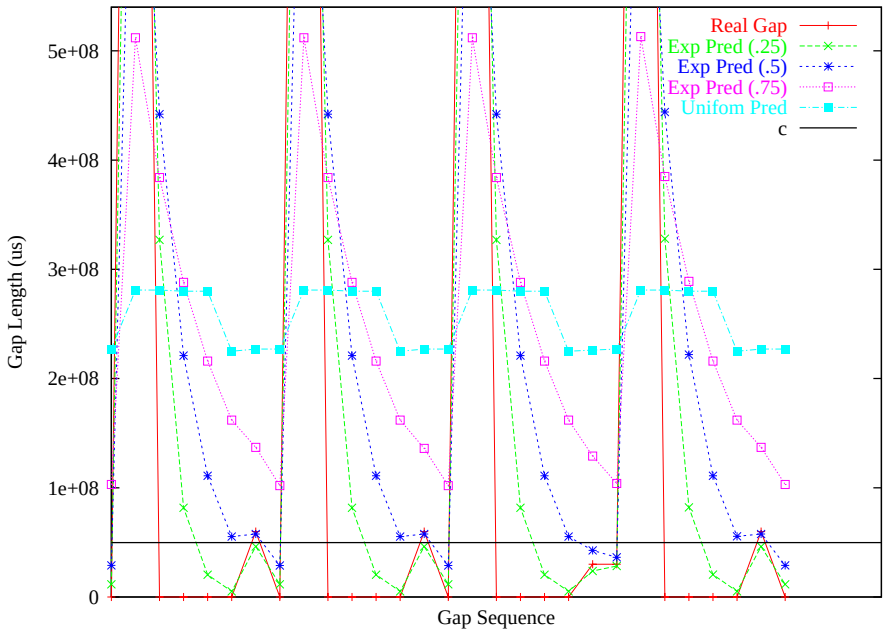


Figure 4.2: Single Value Gap Prediction

state transition would be correct. Since the "Real Gap" line fluctuates around c, the predicted lines lag behind the variation. Therefore, it is impossible for a single value prediction scheme to produce an accurate estimation. The underlying reason is that the variation in the historical data cannot be incorporated in a single value.

To reduce the energy cost incurred by mis-predictions, Hwang [33] introduced a timer to recalculate the estimation during long gaps and an upper bound to limit the maximum gap increase. However, more parameters have to be chosen carefully in a case by case fashion and the fundamental problem has yet to be solved.

4.2.2 Probabilistic Approach

Instead of making decisions at the beginning of each gap, a threshold-based strategy waits for a threshold amount of time. On small gaps it avoids unworthy transitions. On big gaps it has to spend a threshold amount of time in high power. However, by incorporating the knowledge of past gaps and carefully setting this threshold one can

expect to save more energy.

Formally, given the gap probability distribution $f(g)$, the problem can be expressed as follows:

$$E(\sigma) = \int_0^{\sigma} g f(g) dg + (c + \sigma) \int_{\sigma}^{\infty} f(g) dg \tag{4.2}$$

$$\text{choose } \sigma \in [0, \infty) \text{ that minimizes } E(\sigma)$$

where σ is the threshold and $E(\sigma)$ is the energy cost function. The first term in Equation 4.2 represents the expected energy cost for the gaps smaller than σ and the second represents the expected energy cost for those greater than σ where transitions are taken. We can use c, as computed in Equation 4.1, to denote the transition cost. Other variables, i.e. g and σ, are used in the same way to denote energy. Apparently, the energy computed by Equation 4.2 is a relative value. However, it suffices to be an equivalent target function for minimizing the absolute energy consumption. A formal verification of this claim is given as follows.

Based on Figure 4.1, the expected energy consumption for each gap, $E'(\sigma)$, can be computed as follows

$$\begin{aligned}
E'(\sigma) = & \int_0^{\sigma} p_{high} g f(g) dg + \int_{\sigma}^{\infty} (p_{high}\sigma + p_{down}t_{down} + p_{up}t_{up}) f(g) dg \\
& + \int_{\sigma}^{\infty} (g - \sigma - t_{down}) p_{low} f(g) dg \\
= & p_{high} \int_0^{\sigma} g f(g) dg + (p_{high} - p_{low})(\sigma + \frac{(p_{down} - p_{low})t_{down} + p_{up}t_{up}}{p_{high} - p_{low}}) \int_{\sigma}^{\infty} f(g) dg \\
& + p_{low} \int_{\sigma}^{\infty} g f(g) dg \\
= & (p_{high} - p_{low})[\int_0^{\sigma} g f(g) dg + (\sigma + c) \int_{\sigma}^{\infty} f(g) dg] + p_{low} \int_0^{\infty} g f(g) dg \quad \textbf{(by Eq. 4.1)} \\
= & (p_{high} - p_{low}) E(\sigma) + p_{low} \int_0^{\infty} g f(g) dg \qquad \textbf{(by Eq. 4.2)}
\end{aligned}$$

Therefore, minimizing $E'(\sigma)$ is equivalent to minimizing $E(\sigma)$. Note to minimize the energy consumption, *c is the only value the power management strategy has to know about the device.*

If the power management policy can be relaxed to be non-deterministic, the threshold σ will extend to be a vector $\vec{\sigma} = (\sigma_1, \sigma_2, ..., \sigma_n)$. A probability vector $\vec{\pi} = (\pi_1, \pi_2, ..., \pi_n)$ gives π_i as the probability of choosing σ_i. Then the optimization problem becomes:

$$E(\vec{\sigma}) = \sum_i E(\sigma_i) \cdot \pi_i \tag{4.3}$$

$$\text{choose } \vec{\sigma} \text{and } \vec{\pi} \text{that minimize } E(\vec{\sigma})$$

$$\text{subject to: } \sigma_i \in [0, \infty] \text{and } \sum_i \pi_i = 1$$

Now the problem boils down to the knowledge and representation of the probability distribution of gaps. There are three families of thoughts when coping with this problem: worse-case based competitive analysis, parametric modeling and empirical approach.

Competitive Analysis

The power management problem can be mapped into the classic online rental-to-buy problem, which has been studied extensively in the domain of competitive analysis [7, 41, 42, 38, 48, 66]. Competitive analysis is concerned with comparing the worst-case performance of an on-line algorithm with that of the optimal offline algorithm which has complete knowledge of the future, and the ratio between the costs of the two algorithms are defined as the competitive ratio of the on-line algorithm. As an agnostic and pessimistic approach, competitive on-line algorithms make no assumption about the future access pattern and are interested in finding the algorithm that

performs best in the worst scenarios. In reality, where worst cases are infrequent and future can be learned from the history, their performance is too conservative. However, they can serve as a lower bound to evaluate other history-based techniques. For example, a simple fixed threshold of c can perform at most twice as bad as the optimal policy, and a non-deterministic strategy can achieve an asymptotic competitive ratio of $e/(e-1) \approx 1.58$.

Parametric Modeling

From Equation 4.2 and 4.3 we know if we can formulate the probability distribution $f(g)$, we might be able to solve the problem analytically. A good amount of work [72, 85, 79, 78, 18, 19] has been done to model access patterns as exponential or Pareto distribution so the system can be treated as a Markov or semi-Markov process and then the optimization problem can be solved accordingly. Despite its appealing feature that an analytical solution can be obtained, this kind of approach has several limitations which limit their usability.

- First, they make a strong assumption that the gap can be characterized by a known distribution. It may be true in certain scenarios (i.e. single application, one execution epoch, fixed user request pattern, etc.). Figure 4.3 shows the gap distribution of the benchmark *compress* under the trace-drive simulation with a simple processor model. It matches well to an exponential distribution that has the same mean. It also passes a χ^2 test with a significance level of 5%. However, in a real environment, where requests from multiple processes are mixed before sending to the device and the upper level program behavior can change over time, it would be impossible to capture the gap with a known distribution. For example, Figure 4.4 shows how a distribution of the wireless access gaps differs from the known distributions. The distribution is collected

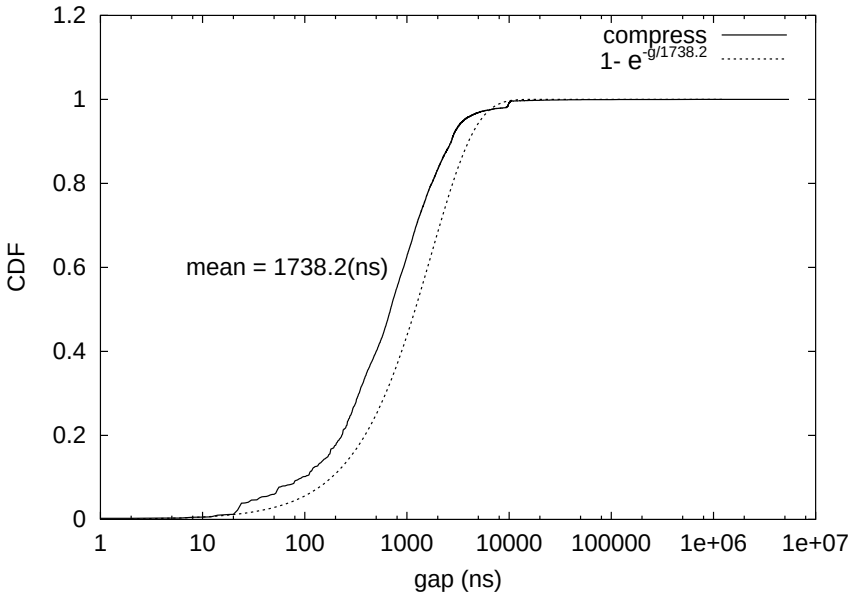


Figure 4.3: Gap Distributions vs. Exponential (compress)

from a campus wireless network [44] and reflects the real access pattern.

- Second, both modeling of the request pattern and solving of the optimization problem incur high computation cost and mostly have to be done offline. This makes them impractical to be implemented as a general online algorithm.

- Third, they all assume a stationary distribution during the whole period where the modeling and optimization are performed once. However, even inside a well modeled period there might exist non-stationarity which can be exploited to do finer management and gain more benefit.

All these limitations prevent parametric modeling from being a general and practical power management solution.

Empirical Approach

An empirical approach makes no assumption about the gap distribution and learns it by bookkeeping the historical data. The policy optimization is performed using this

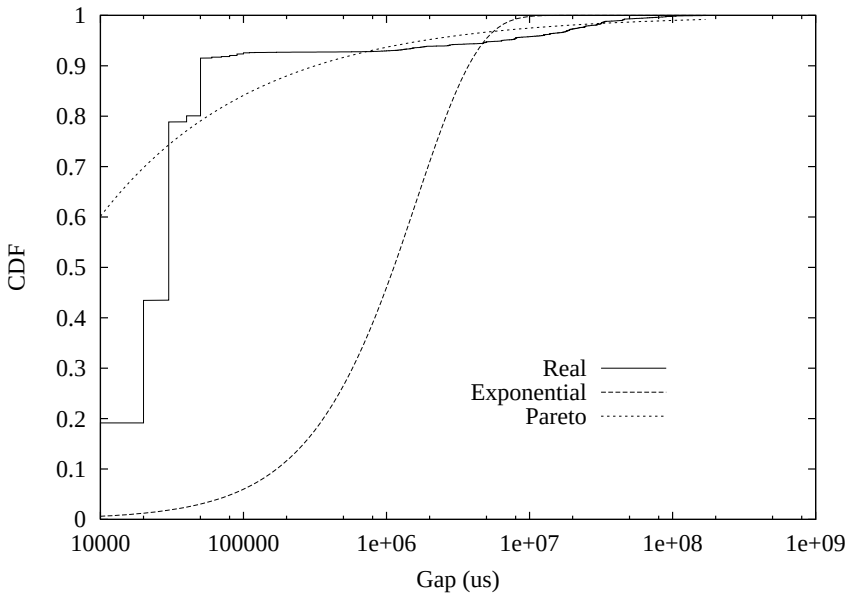


Figure 4.4: Wireless Gap Distribution

data periodically. This technique has been widely adopted in the area of dynamic power management [2, 39, 47, 56, 92]. Lorch's use of a kernel function [56] and Yuan's use of a histogram [92] are for estimating CPU demand and improving a DVS algorithm. Their optimization is specialized for CPU, a scalable device with non-linear power characteristics, and thus cannot be used on non-scalable devices. A histogram is also used in [47, 2, 39] to track idle times (gaps) of disk and wireless card. Given a histogram, a discrete form (Equation 4.4) of Equation 4.2 can be used to compare the expected energy cost for different thresholds, where N_{g_i} is the count of gap g_i and $\{\sigma_1, \sigma_2, ..., \sigma_n\}$ is the pool of candidate thresholds.

$$E(\sigma) = \sum_{g_i \leq \sigma} g_i N_{g_i} + (c + \sigma) \sum_{g_i > \sigma} N_{g_i} \tag{4.4}$$

$$\text{choose } \sigma \in \{\sigma_1, \sigma_2, ..., \sigma_n\} \text{ that minimizes } E(\sigma)$$

The power manager just picks the threshold that incurs the least energy. However, there are several limitations with this approach.

- First, historical data is always accumulated and never aged out [47, 2]. Therefore, the variables for accumulating statistics could overflow or have to be reset periodically. More importantly, due to the non-stationarity of access patterns, older samples become less relevant to the recent distribution and should be retired. This non-stationarity could be the result of user behavior change, process switching, or phase change inside one program. It could happen in large or small time scale. Figure 4.5 and 4.6 illustrate this non-stationary behavior for memory and a wireless card, respectively. Each curve in Figure 4.5 represents a one-million instruction snapshot taken at different points of a program's execution (e.g. xB-yM denotes a starting point of x billion and y million instructions). Each curve in Figure 4.6 represents a one-hour wireless trace of a consecutive ten hour period.

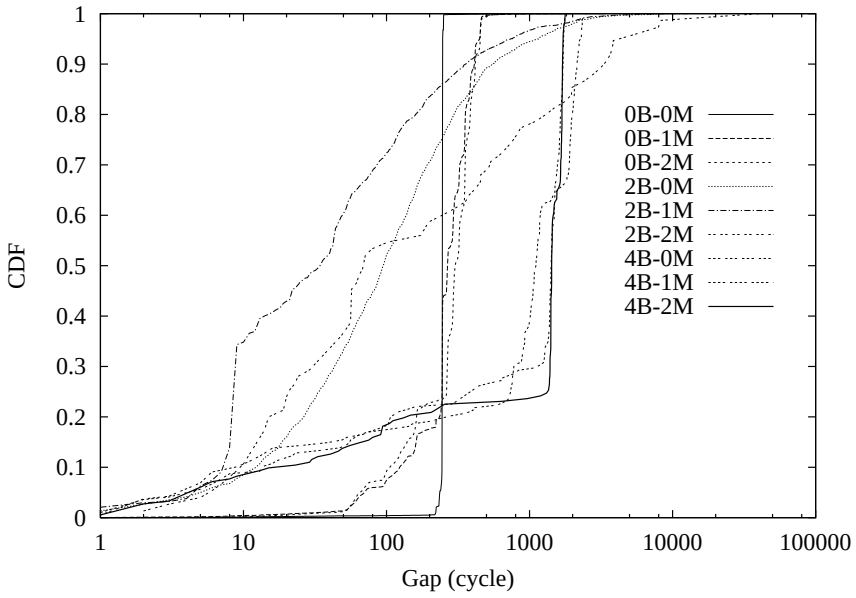


Figure 4.5: Non-stationarity: memory gap

- Second, to limit the number of counters (N_{g_i}), the range for choosing the optimal gap ($[0, \infty)$) is split into a limited number of buckets [2, 39], and gaps falling into the same bucket are treated equally. This introduces error in finding

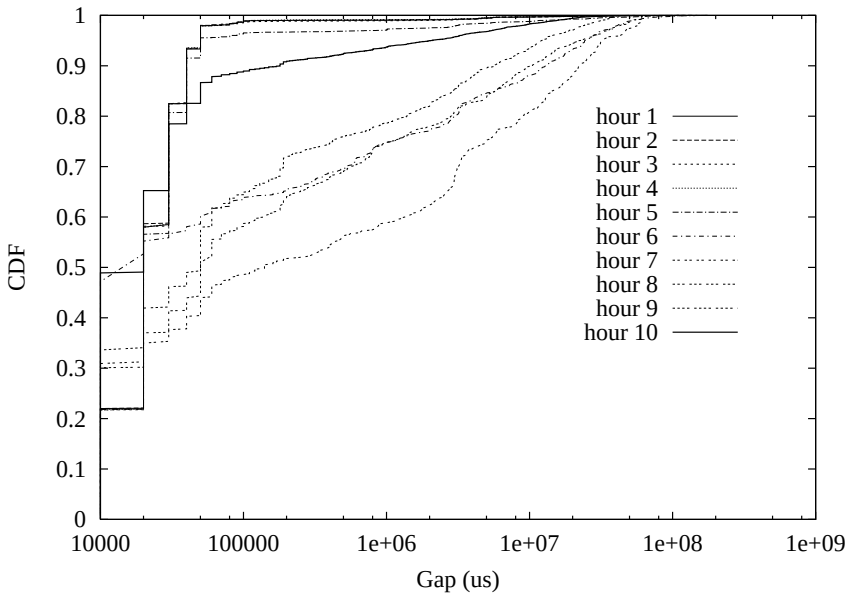


Figure 4.6: Non-stationarity: wireless gap

the best threshold among $\{\sigma_1, \sigma_2, ..., \sigma_n\}$.

- Third, none of the above approaches provide a convincing method for determining the range for the candidate thresholds. They either pick a hueristic value or make an assumption on the biggest gap possible.

To address the above problems with the available empirical approach, we present our adaptive algorithm in the next section.

4.3 Proposed Strategy

In this section, we present our strategy for estimating access patterns and efficiently computing the optimal threshold.

4.3.1 Data Structure and Algorithm

Because the data update and the decision making have to be done in real time, the simplicity of the data structure and the efficiency of the algorithm become the major

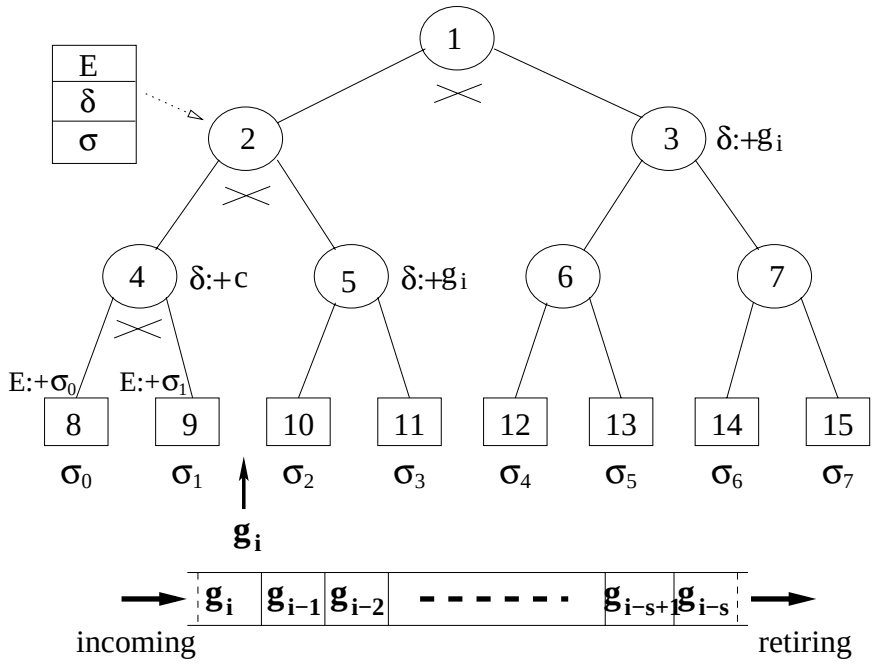


Figure 4.7: Data Structure of the Online Algorithm

consideration. As shown in Figure 4.7, we use a window to store the recent s number of gaps. The window is simply implemented as a circular buffer with the header pointing to the position where the oldest gap is retired and the upcoming gap is stored. We adopt a binary comparison tree to maintain and update the energy cost associated with each candidate threshold and output the optimal threshold. Three variables (E, δ, σ) are allocated in each node (internal node and leaf) of the tree. On each internal node, variable σ and E store the best threshold among its descendant leaves and the energy incurred by this threshold. On the leaves, σ's are just initialized to an ascendingly sorted list of candidate thresholds, i.e. σ_0–σ_7 in Figure 4.7, and E's are the energy cost values updated on the fly. As can be seen from Equation 4.4, an incoming gap, e.g. g_i, incurs an additional energy cost of $c + \sigma_i$ for each threshold σ_i less than g_i, where the transition would be made; and an additional energy cost of g_i for all the thresholds no less than g_i, where no transition would be made. For example in Figure 4.7, suppose the incoming gap g_i falls within $(\sigma_1, \sigma_2]$, then the E

values of σ_0 and σ_1 will be incremented by $c + \sigma_0$ and $c + \sigma_1$ respectively; and those of σ_2–σ_7 will be incremented by g_i. After the E values of all leaves are updated, the first layer of internal nodes can proceed to choose from its two children the one with smaller E value to update its own E and σ value. Then the next layer of internal nodes can proceed. This is done until the root is updated with the best threshold.

To minimize the cost of computation and comparison, we introduce a δ value on each node to store the common energy cost for its descendant leaves wherever possible, so the leaves don't have to update with this cost if their ancestor has done so. To output the best threshold, comparison is needed only at the node where any of its children has been updated. Note the δ value should be counted when making comparisons. For the case of Figure 4.7, node 8 and 9 increment their E value by σ_0 and σ_1 respectively; node 4 increment its δ by c; node 3 and 5 increment its δ by g_i. Then only three comparisons are sequentially made at node 4, 2 and 1.

The update of the tree on the retirement of the oldest gap g_{i-s} (assuming a window size of s) is performed in the same fashion. After these two rounds of updates—admitting g_i and retiring g_{i-s}, the power manager uses σ value of the root as the threshold for the next gap. The correctness of this algorithm is guaranteed by the following induction:

1. We can initialize the tree to make all nodes well-maintained. By "well-maintained" we mean the σ value of the node is the best one among its descendant leaves.

2. If the tree before the current round of updates is well-maintained, then after the current round we check each node from bottom up. All leaves are well-maintained. If the internal node has no updated children, it is already well-maintained since all its descendant leaves must have had a common δ value change which does not affect its σ, which is well-maintained by assumption;

otherwise, by definition, a comparison is made and its σ is updated correctly.

4.3.2 Discussion

Implementation and Complexity

Figure 4.8 is the c-style pseudo-code for updating the tree on the admission of gap g. We use an array to implement the tree so that most operations can be efficiently performed. For instance, node i's two direct children are simply $2i$ and $2i+1$, and its parent is $i/2$.

Locating g in the sorted list of candidate thresholds takes $O(\log n)$ time, where n is the number of thresholds. Then the leaves are divided into a higher portion, where σ's are no less than g, and a lower portion, where σ's are less than g. The first *for* loop adds g to the δ value of the common ancestors of the higher portion. Pointer i starts from the first leaf and leaps through the internal nodes. By the *while* loop it finds the most common ancestor covering the current branch. Then it moves to the sibling (i ++) until it is the rightmost node ($i = 2^k - 1 \Leftrightarrow i \,\&\, (i+1) == 0$). In the same way, the second loop adds c to the common ancestors of the lower portion. Both loops take $O(\log n)$ time. The third loop adds its own σ to the leaves of the lower portion. Whenever ready, comparison and update of the ancestors are done at the same time. This loop takes $O(n)$ time.

Retiring the oldest gap is implemented in a similar way. Overall, the complexity of outputing one threshold is $O(n)$. The operations involved in the algorithm are simple: *add*, *bitand*, *bitshift*, *branch*, and *load/store*. The space requirement is basically the two arrays with a total capacity of $n + s$, where s is the window size. Therefore, this algorithm can be efficiently implemented even in hardware. This helps to make it a general power management strategy which can be used for a variety of devices.

```
Algorithm gap_admission(tree, n, g)
begin
Find k such that g ∈ (tree[k].σ, tree[k+1].σ];
for (i = k + 1; i < 2n; i ++)
  while ((i & 1) == 0) i >>= 1;
  tree[i].δ += g;
  if ((i & (i+1)) == 0) break;
for (i = k; i >= n; i --)
  while ((i & 1) == 1) i >>= 1;
  tree[i].δ += c;
  if ((i & (i−1)) == 0) break;
for (i = k; i >= n; i --)
  tree[i].δ += tree[i].σ;
  for (j = i; (j & 1) == 0; )
    j >>= 1;
    if (tree[2j+1].E + tree[2j+1].δ < tree[2j].E + tree[2j].δ)
      tree[j].E = tree[2j+1].E + tree[2j+1].δ;
      tree[j].σ = tree[2j+1].σ;
    else
      tree[j].E = tree[2j].E + tree[2j].δ;
      tree[j].σ = tree[2j].σ;
end
```

Figure 4.8: Algorithm Pseudo-code: Input is $(tree, n, g)$, where $tree$ is the array-implemented binary comparison tree starting from $tree[1]$ to $tree[2n]$, n is the number of leaves and g is the incoming gap. At the end $tree$ is well-maintained and the root $tree[1]$ records the optimal threshold $tree[1].\sigma$.

Determining Parameters

Several parameters have to be determined to make the algorithm work: the number of the candidate thresholds n, the window size s, and the range of the candidate thresholds. Within a fixed range, the more candidate thresholds, the finer granularity for choosing the optimal, and the more effective the power management can be. From the above discussion, however, the computation cost and space cost are linear to n. Therefore, a general guideline for choosing n is that under the budget of computation and space, choose an n as large as possible.

For a stationary or time-independent gap distribution, the larger the window, the better the distribution can be learned. For a non-stationary distribution, the window cannot be arbitrarily large since the old information doesn't apply to the current distribution anymore. From experiments (i.e. Figure 4.5 and 4.6), we know that the gap distribution changes slowly. We also know that only the space cost is linear to the window size s and the computation cost does not depend on s. Therefore, we usually choose a window size which is "sufficiently large". Although we haven't had a rigorous analysis on the window size, we present an experimental sensitivity analysis in Section 4.4.

The other important parameter we have to determine is the range of the candidate thresholds. Theoretically, depending on the gap distribution, the optimal threshold can be arbitrarily large. For example, due to its memoryless property, an exponential distribution with a mean less than c incurs the least energy when the threshold equals to the largest gap so that no transition is taken. This complicates the selection of candidate thresholds because the largest gap is usually unknown at the time when the algorithm is implemented, and to be able to always capture the optimal threshold, an infinite range of $[0, \infty)$ has to be used. Fortunately, for an arbitrary gap distribution,

choosing a threshold within a finite range does not cost a lot more than doing so within the an infinite range. In this section, we present our analysis which gives an upper bound on the relative error when using a finite threshold range instead of $[0, \infty)$.

Theorem 4.3.1. *Let G be a gap distribution, $G = \{(g_i, N_i)\}$, where N_i is the count of gap g_i. Let $E(\sigma, G) = \sum_{g_i \leq \sigma} g_i N_i + (c + \sigma) \sum_{g_i > \sigma} N_i$, and $E_{min}(R, G) = Min\{E(\sigma, G) | \sigma \in R\}$. Choose $r \in \mathbb{N}$, for any distribution G on $[0, \infty)$,*

$$\frac{E_{min}([0, rc], G) - E_{min}([0, \infty], G)}{E_{min}([0, \infty], G)} < \frac{1}{2^{r-1}}$$

Proof. For a distribution G, assume the optimal threshold $\sigma_m > rc$, meaning $E(\sigma_m, G) = E_{min}([0, \infty], G)$.

We only have to consider the case where no gaps in G are greater than σ_m. Because for the case where there exist gaps greater than σ_m, after removing these gaps, σ_m is still the optimal and the relative error between $E_{min}([0, rc], G)$ and $E_{min}([0, \infty], G)$ can only increase.

We split the default range $[0, \infty)$ into $r+1$ regions: $[0, c)$, $[c, 2c)$, $\cdots$, $[(r-1)c, rc)$, $[rc, \infty)$. Let $g_i^{(k)}$ and $N_i^{(k)}$ denote the ith gap and its count falling in the kth region. We pick a threshold σ'_m such that $\sigma'_m \in [(r-1)c, rc)$ and no gap exists in (σ'_m, rc). Now we have

$$\begin{aligned} E(\sigma'_m, G) - E(\sigma_m, G) &= \sum_i (c + \sigma'_m - g_i^{(r)}) N_i^{(r)} \\ &< c \sum_i N_i^{(r)} \end{aligned}$$

$$
\begin{aligned}
E(\sigma_m, G) &= \sum_i g_i^{(r)} N_i^{(r)} + \sum_i g_i^{(r-1)} N_i^{(r-1)} + \cdots \\
&\quad + \sum_i g_i^{(1)} N_i^{(1)} + \sum_i g_i^{(0)} N_i^{(0)} \\
&> rc \sum_i N_i^{(r)} + (r-1)c \sum_i N_i^{(r-1)} + \cdots \\
&\quad + c \sum_i N_i^{(0)}
\end{aligned}
$$

For a notation convenience, let $N^{(k)} = \sum_i N_i^{(k)}$. The above inequalities can be rewritten as

$$E(\sigma'_m, G) - E(\sigma_m, G) < cN^{(r)} \tag{4.5}$$

$$E(\sigma_m, G) > rcN^{(r)} + (r-1)cN^{(r-1)} + \cdots + cN^{(1)} \tag{4.6}$$

Now if we know the relation between $N^{(n)}$ and $N^{(k)}$, by Inequality 4.5 and 4.6 we can compute the upper bound. Knowing σ_m is the optimal threshold, we have

$$E(\sigma_m, G) < E((r-2)c, G) \Rightarrow N^{(r-2)} > N^{(r)}$$

$$E(\sigma_m, G) < E((r-3)c, G) \Rightarrow N^{(r-3)} > 2N^{(r)} + N^{(r-1)}$$

$$E(\sigma_m, G) < E((r-k)c, G) \Rightarrow$$

$$N^{(r-k)} > (k-1)N^{(r)} + (k-2)N^{(r-1)} + \cdots + N^{(r-k+2)}$$

Solving the above recursion, we have

$$N^{(r-k)} > 2^{(k-2)} N^{(r)} \tag{4.7}$$

Combining Inequality 4.5,4.6 and 4.7 completes the proof. □

By Theorem 4.3.1, we know that the relative error of using range $[0, rc)$ decreases exponentially with r. For example, an r value of 5 can bound the error to be less than 6%. However, an r value of 1, as used in [39], is not large enough to bound the error. For example, a uniform gap distribution of $(0.5c, 1.1c)$ can lead to an error of 25% if a threshold range of $[0, c]$ is used.

4.4 Evaluation

In this section, we use simulations to evaluate our power management strategy.

4.4.1 Methodology

To evaluate the generality of our algorithm, we test it on three hardware devices: memory, disk and wireless card. For memory, we use execution-driven simulation. We modified the SimpleScalar [9] simulator to include a detailed memory model. The two-state power model shown in Table 4.1 is based on the Infineon 256Mbit Mobile-RAM [36]. The processor model is based on Intel's XScale [54]. It has a 32KB one-level on-die cache with 32B block size and 32-way associativity. The workloads are from SPEC [1] benchmark suite.

We use trace-driven simulations to evaluate the power management on hard disk and wireless card. Disk access traces were collected at HP lab [83]. The traces we used in our experiments include one 7-day trace, one 3-day trace and one 2-day trace. The parameters of the disk power model, shown in Table 4.1 are taken from the data sheet of the IBM Ultrastar server hard disk [34].

Traces for wireless card are collected by Kotz et al. on the Dartmouth campus network [44]. They are basically tcpdump files with all wireless traffic recorded and mixed together. According to the source/destination MAC address, we separate the trace files into streams. Each stream relates to a specific wireless card and thus can be used for our purpose. The number of accesses varies from several hundred to one million and the stream lifetime varies from seconds to hours. The wireless card power model shown in Table 4.1 are measured by Anand et al. for an Orinoco Silver 802.11b wireless card [2].

Device	p_{act}	p_{high}	p_{low}	p_{down}	t_{down}	p_{up}	t_{up}	c
Memory	195mW	60mW	1.8mW	30mW	7.5ns	60mW	7.5ns	11.4ns
Disk	39W	22.3W	4.15W	4.15W	15s	34.8W	26s	49.9s
Wireless	2.70W	1.21W	0.19W	1.19W	260ms	1.04W	230ms	490ms

Table 4.1: Power Model of Memory, Disk and Wireless Card. P_{active} is the power consumption for servicing requests. We need this value to compute the active energy consumption for servicing requests, because the total energy reported in this section includes this part of energy. Note that this part of energy stays as a constant for all strategies.

4.4.2 Comparison of Strategies

To evaluate the effectiveness of our power management strategy, we compare the following strategies:

- **Oracle**. With perfect knowledge about the future gaps, it compares the upcoming gap with c and always makes the right decision. Although it is unrealistic, it serves as an upper bound of the energy saving.

- **Tree**. Our tree/window-based algorithm with 128 candidate thresholds, a threshold range of $[0, 4c]$ and a window size of 1024.

- **Histo**. This is the histogram-based algorithm similar to those in [39, 2] except that to make it comparable, we also configure it to have 128 candidate thresholds and a window size of 1024.

- **Pred**. It is the single-value prediction algorithm with exponential averaging. As suggested in [33], we use a preserving factor of 50%, a timer of c for re-estimating gap periodically and a ratio of 2 for limiting prediction increase rate.

- **2-comp**. It uses c as a fixed threshold to trigger power state transitions. According to competitive analysis [41, 38], it incurs energy cost at most as twice

as the oracle. Note this only counts energy consumed during gaps (idle time). Since during active time all strategies consume the same amount of energy, the previous guarantee still holds.

- **Aggressive**. It uses 0 as a fixed threshold and always transitions to low power state during each gap.

- **Unmanaged**. The default operation mode without power management. It stays in high power and never makes transition. It can also be considered as a fixed threshold strategy with infinity threshold.

- **Varth**. It adaptively changes the threshold value depending on how the previous gap compares to c [14]. It doubles the threshold if the previous gap is less than c and halves it otherwise. When comparing gap with c, a significance factor of 10% is used to prevent threshold fluctuation, and $4c$ is used as an upper limit of the threshold range.

Figure 4.9 shows the results of applying the above strategies on three types of devices. Y-axis is the total energy normalized to *oracle*. The total energy includes energy consumed in both idle time and active time. Since the active time energy stays constant across all strategies, the energy saving would be greater if only the idle time energy is counted. The first observation is that *tree* performs consistently well on all kinds of platforms and under different workloads. On disk and wireless card, it consumes at most 7.1% more energy than *oracle*, and is always closer to optimal than the best of other strategies. On memory, the advantage of *tree* becomes less pronounced. This is because the transition cost c (11.4ns) is fairly small compared to the average gap of all the benchmarks (77ns–1296ns). For example, *bzip*, with the least difference between *oracle* and other threshold-based strategies, has an extremely low cache miss rate (0.2%) and hence results in an average gap of 1296ns.

So a 0-threshold (aggressive) strategy can perform almost as good as the optimal. Nevertheless, *tree* still remains the best online strategy.

Across all three graphs, the *unmanaged* always consumes 3.3–8.5 times energy as the *oracle*. Therefore, a tremendous amount of energy wasted in idle time can be potentially saved by making effective power transitions. Using our power management strategy the energy consumption can be reduced to 13%–33% of the unmanaged mode.

4.4.3 Sensitivity Analysis of Window Size

In this and the next section, we run simulations to examine how sensitive our strategy is to the window size and the range of the candidate threshold.

Figure 4.10 shows the energy consumption of wireless card when the window size varies from 128 to infinity. Infinity is actually implemented as no window so the incoming gaps keep being absorbed by the tree and never get retired. We can see generally our power management strategy is not sensitive to window size when it is reasonably large. However, infinite window size consumes more energy. This justifies our choice of a window-based approach because of its ability of adapting to the phase change of the resource access patterns. The uniformity of memory energy consumption across different window sizes is because the sufficiently long warm-up stage and relatively short simulation time have not fully exposed the non-stationarity of the memory access patterns.

4.4.4 Sensitivity Analysis of Threshold Range

With the aid of Theorem 4.3.1, we know that the range of the candidate thresholds can be related to the transition cost c: a value of rc can be used as an upper limit of the candidate thresholds, given a tolerable relative error of energy consumption

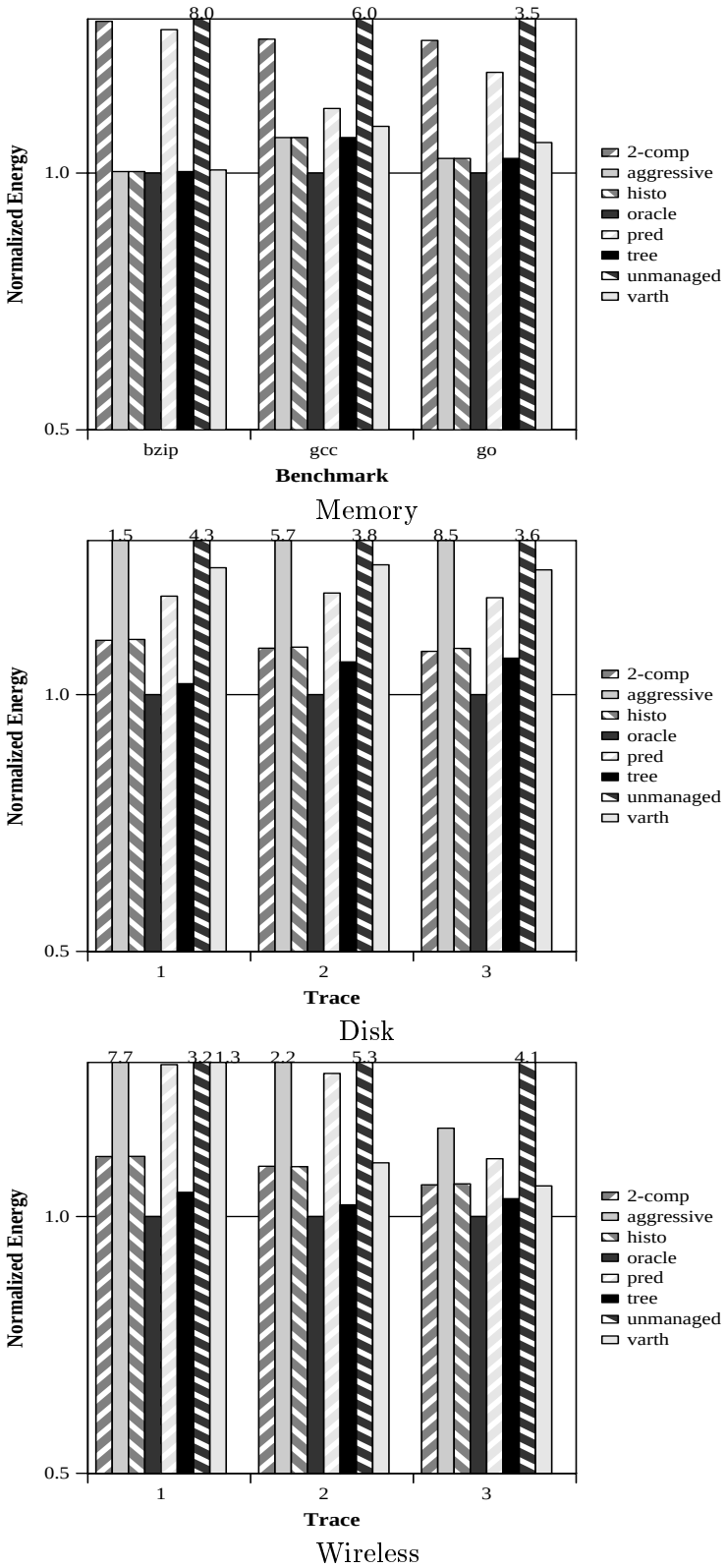


Figure 4.9: Comparison of Power Management Strategies

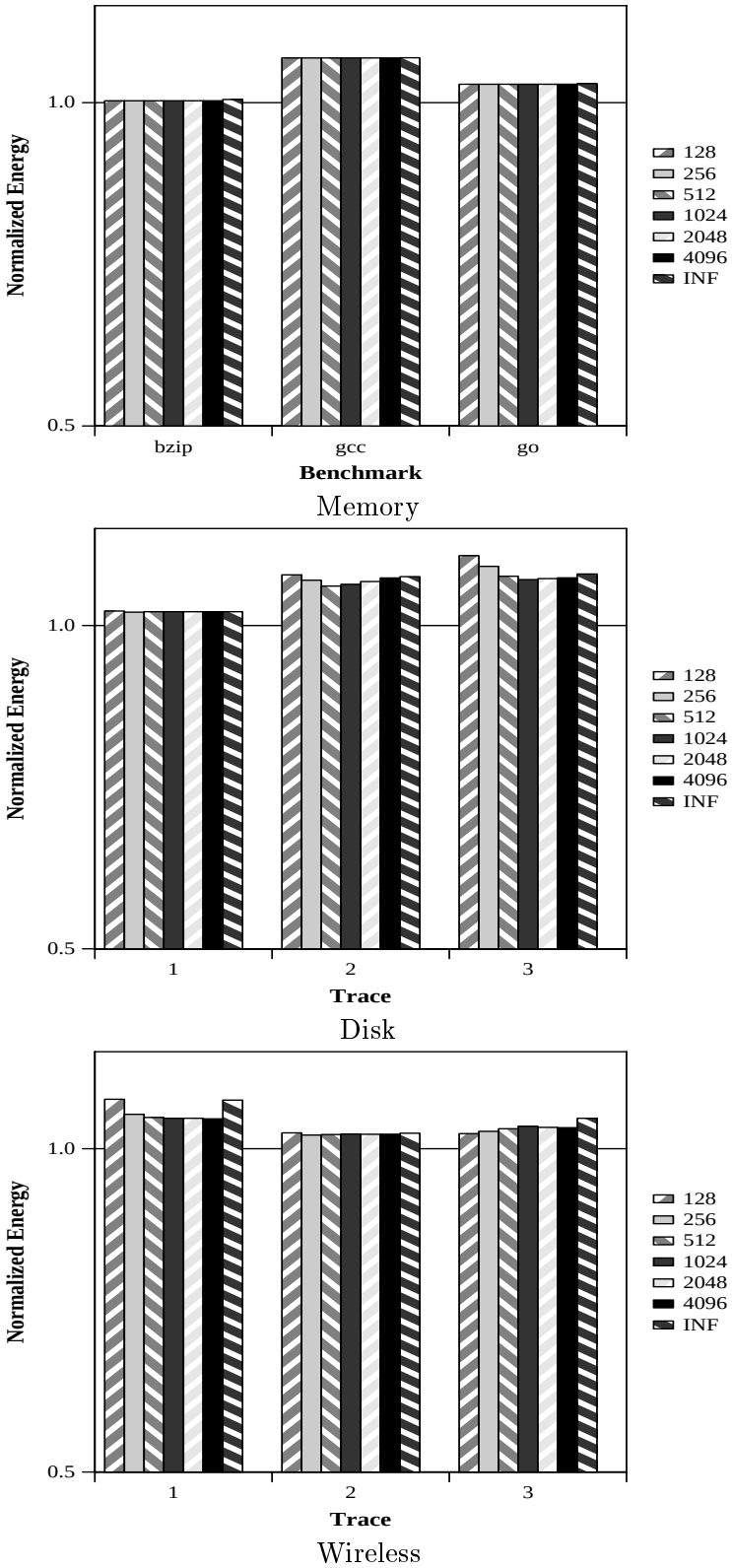


Figure 4.10: Sensitivity Analysis of Window Size

between a global optimal threshold, chosen from $[0, \infty)$, and a suboptimal threshold, chosen from $[0, rc]$, when the worst-case access pattern happens. To investigate the adequate range of the candidate thresholds, we fix other parameters (window size 1024 and 128 candidate thresholds) and vary the threshold range from c to $5c$. As we can see from Figure 4.11, the energy consumption stabilizes when the range is above $3c$.

Again, memory benchmarks do not appear to be sensitive to different threshold ranges. This is also due to the relatively very big gaps (77ns-1296ns vs. $c = 11.4$ns), so that almost all the time the optimal threshold just stays at zero and larger ranges do not make a difference. Nevertheless, to deal with all kinds of access patterns, we still need a sufficiently large range (i.e. $3c$) for choosing the optimal threshold so that the maximum error to the global optimal is bounded at a certain level.

4.4.5 Performance Impact

So far we have been focusing on minimizing energy consumption. In this section, we investigate the performance impact of the power management strategies.

Intuitively, the performance penalty incurred by a power management policy cannot be arbitrarily high if it minimizes the energy consumption. This is simply because energy is the integral of power over time. Next I derive an upper bound on the performance penalty for our power management algorithm.

Using the same notations as in Section 4.2.2, we use $t = \int_0^\infty g f(g) dg$ to denote the average length of all idle periods for the default performance-oriented policy (unmanaged). The extra per-gap latency incurred by using a power management policy with a transition threshold of σ is $\Delta t = t_{up} \int_\sigma^\infty f(g) dg$. Assume the energy

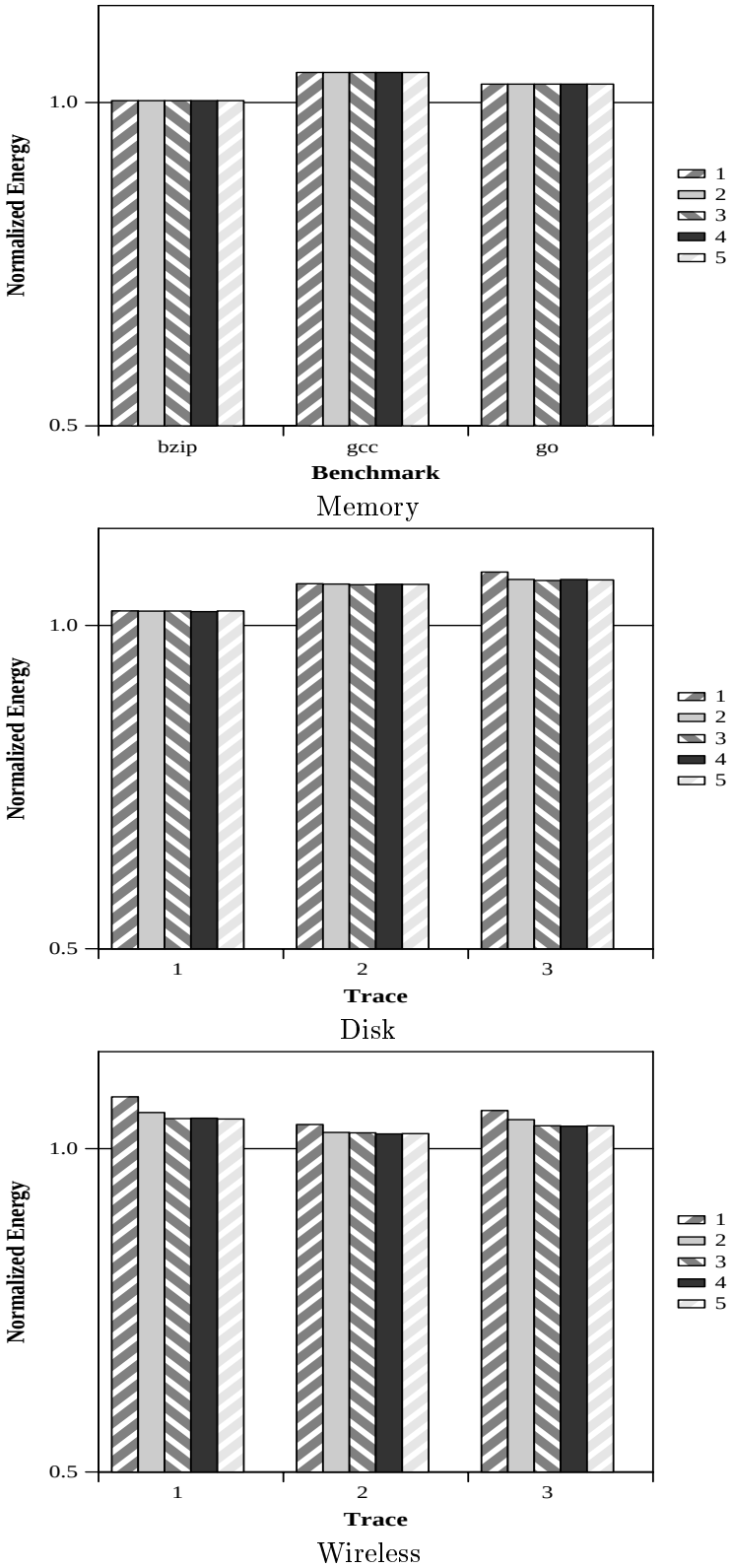


Figure 4.11: Sensitivity Analysis of Threshold Range

consumption of using our optimal threshold σ is

$$E(\sigma) = \int_0^{\sigma} p_{high} g f(g) dg + \int_{\sigma}^{\infty} (p_{high}\sigma + p_{down}t_{down} + p_{up}t_{up}) f(g) dg$$
$$+ \int_{\sigma}^{\infty} (g - \sigma - t_{down}) p_{low} f(g) dg$$

and the energy consumption of making no transitions (unmanaged) is

$$E(\infty) = p_h \int_0^{\infty} g f(g) dg$$

Now we have the following deduction:

$$E(\sigma) < E(\infty)$$
$$\Rightarrow\ p_{up}t_{up} \int_0^{\infty} f(g) dg < (p_h - p_l) \int_{\sigma}^{\infty} (g - \sigma) f(g) dg$$
$$\Rightarrow\ \Delta t < \frac{p_h - p_l}{p_{up}} \int_{\sigma}^{\infty} (g - \sigma) f(g) dg$$
$$\Rightarrow\ \frac{\Delta t}{t} < \frac{p_h - p_l}{p_{up}}$$

Plug in the values from Table 4.1, we get the upper bounds of the relative latency increase: 0.97, 0.52 and 0.98 for memory, disk and wireless card, respectively.

Figure 4.12 shows the performance impacts of different power management strategies. Y values are the latencies normalized to the unmanaged policy. Because of the extreme long gaps of the memory benchmarks, the relative latencies show no difference accross all strategies. For disk and wireless card, aggressive strategy usually incurs significantly high latency. Although our strategy mostly incurs highest latency among the rest, the performance penalty is well within an acceptable level (5%).

4.5 Conclusions

This chapter presents a general and efficient power management strategy for non-scalable devices. By identifying the variety and non-stationarity of the device access

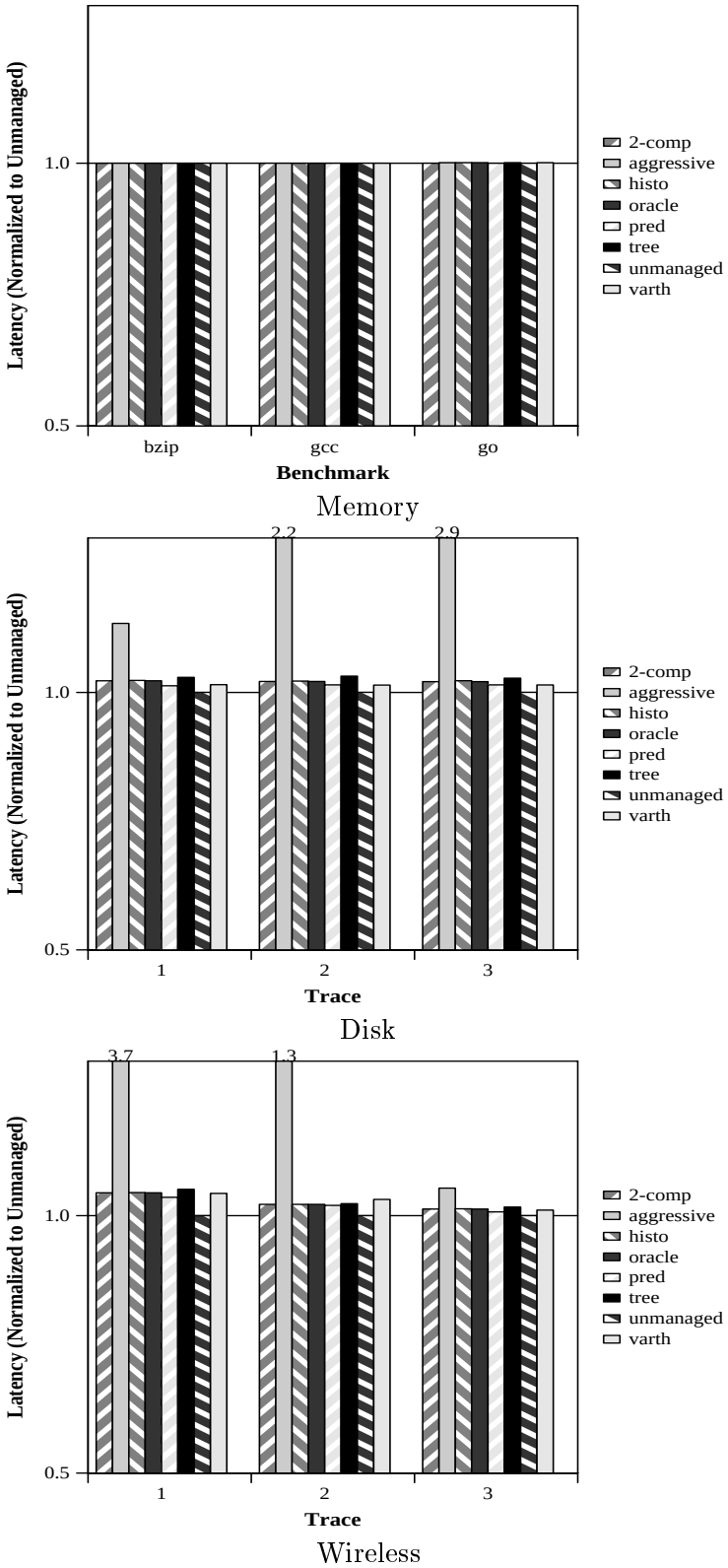


Figure 4.12: Performance Impact of Power Management Strategies

patterns, we propose a probability-based empirical approach. Our strategy uses a window to dynamically monitor the recent idle time distribution, and uses a tree to efficiently update the energy costs and output the optimal threshold. The simulation results show that our strategy adapts to various devices and workloads, and approaches to the optimal with a distance of at most 7.1%. Against the default non-transition mode, our strategy achieves a total energy saving of 67%–87%. Through sensitivity analysis, we show that our strategy is not sensitive to the window size of a certain range, and justify our choice of a window-based approach. We also verify our guideline for choosing appropriate threshold range.

There are several limitations with this work, which we plan to address as future work.

1. While doing optimization, our strategy takes into account only the energy consumed at the individual device under power management. However, the increased latency might raise the energy consumption of other affected devices. Although we can claim that applying power management on all major power-consuming devices can partly solve this problem since the extra idle time imposed on other devices thus won't incur too much energy, we want to develop a systematic power management strategy which can optimize the energy consumption globally.

2. We only give a loose upper bound on the performance penalty. Although the simulation results show our strategy leads to very satisfying performance, there is no guarantee for the performance requirement stricter than the upper bound. We therefore plan to augment our algorithm to factor in the performance requirement while choosing the optimal threshold so that a certain range of performance level can be guaranteed.

3. The window size is still determined in a heuristic way. We like to investigate the scheme for dynamically adjusting the window size depending on how frequently the gap distribution changes. And we speculate that a good adjustment scheme can even save more energy than a static window size.

Chapter 5

Interactions of Power Aware Memory Systems and Processor Voltage Scaling

A real power aware system can not be designed in an isolated way. Preliminary work with focus on one subsystem (i.e. processor, memory, disk, etc.) provides valuable design references, but is just the first step toward a system-wide solution. In this chapter, we take processor and memory, which we believe are major system targets for energy optimization, as an example to study the interactions of system components in light of energy and performance tradeoffs. To fulfill this task, we leverage the results of previous studies on processor and memory energy optimization.

5.1 Introduction

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 employed to exploit 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 [90, 75, 74, 46, 27, 22, 77, 26], 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 [27, 22, 21, 61, 62, 63, 77]. Thus, the impact of memory has been considered to be a complicating factor for the straightforward application of DVS.

In this chapter, we identify a positive synergy between voltage/frequency scaling of the processor and newer 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 [9] 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, we make 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 chapter is organized as follows. Section 5.2 describes our research roadmap and methodology, and Section 5.3 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 5.4. We conclude in Section 5.5.

5.2 Roadmap and Methodology

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.

5.2.1 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 [51, 18, 19] 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.

5.2.2 Methodology

We use a modified version of the PowerAnalyzer [70] 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 1.1. We use two memory chips with a total capacity of 64MB. The variable voltage processor we simulate is based on Intel's XScale [54]. The voltage and frequency values used in our evaluations range from 50MHz and 0.65V to 1000MHz and 1.75V as shown in Table 5.1. The power costs given in the table are approximate; the actual CPU power consumption is derived in the simulator from processor and cache activity. 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 5.2.2, 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

Voltage (V)	**Frequency** (MHz)	**Power** (approximate mW)
0.65	50	15
0.70	100	33
0.80	200	80
1.00	400	222
1.20	600	434
1.40	800	732
1.75	1,000	1,300

Table 5.1: Variable Voltage Processor Values

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

Table 5.2: Mobile-RAM Power State and Transition Values

of the refresh on the *gap* is minor since the refresh cycle is several orders of magnitude longer than the *gap* (ms vs. ns).

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 [77]. We have performed experiments using four applications from the MediaBench suite [52, 5]: 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

CPU Freq (MHz)	CPU Pwr (mW)	Exec Time (ms)	Avg Gap0 (ns)	CPU Eng (mJ)	CPU Residue (mJ)	Mem Eng (mJ)	Standard		Naive	
							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**

Table 5.3: DVS with Standard and Naive Powerdown Memory

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%.

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

Table 5.3 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 5.3 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).

5.4 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.

5.4.1 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 5.3 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.

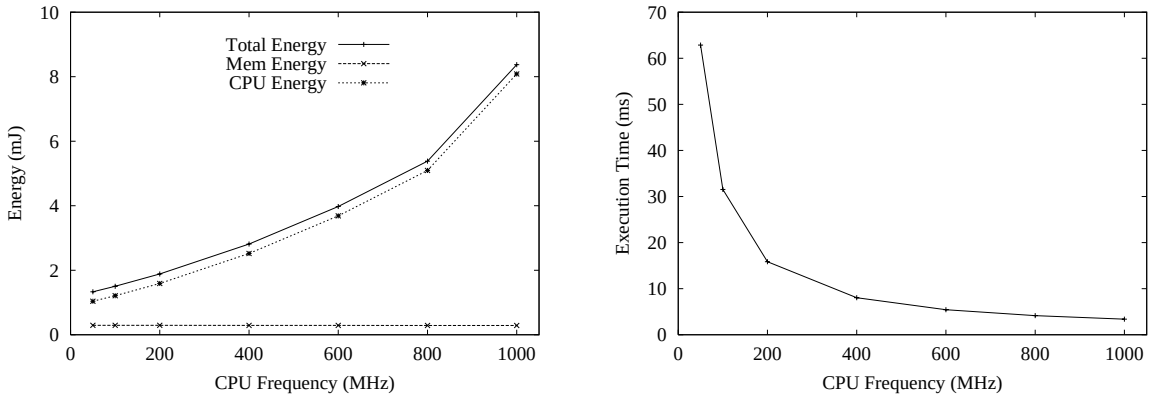
This result conflicts with conventional assumptions used in many DVS algorithms which have been concerned only with CPU energy. *Taking into account the energy used by memory with even minimal power management capabilities, it is no longer best to stretch execution to consume the entire period.* In fact, 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.

5.4.2 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.

Figure 5.1 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 5.4.2, 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



a) Energy b) Task Busy Time

Figure 5.1: DVS and Power Aware Memory: MPEG Decode–Aggressive Policy

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 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.

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). Figure 5.2 illustrates these results by showing energy consumption versus frequency. One line is for

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

Table 5.4: DVS and Power Aware Memory: MPEG Decode

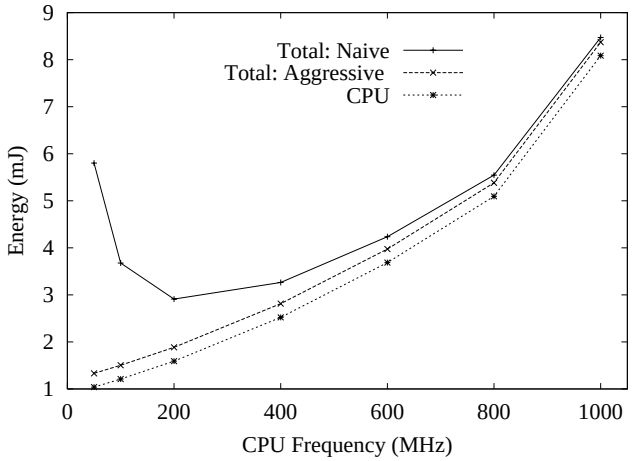


Figure 5.2: DVS and Memory Controller Policies

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.*

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.

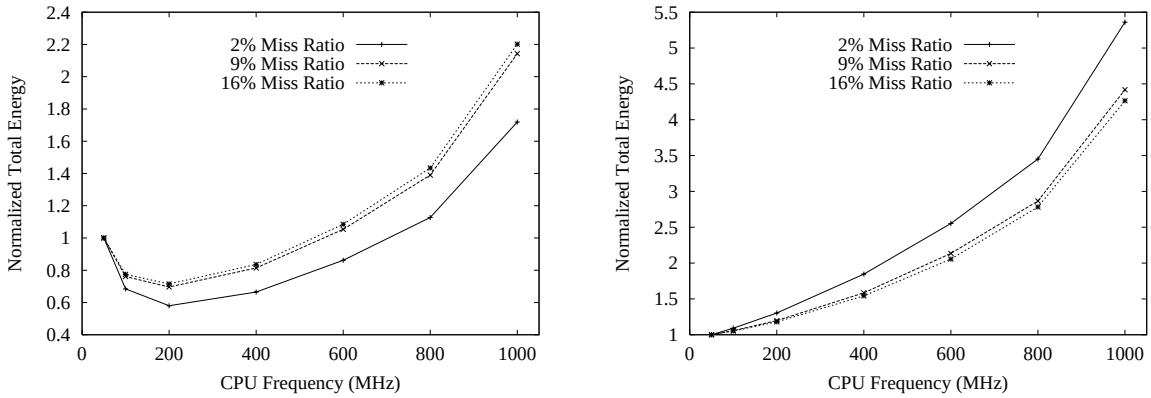
In particular, cache behavior can have a dramatic effect on execution time for some programs. MPEG has a very low data cache miss ratio; however, several researchers have identified embedded applications that incur miss ratios from 5% to 15% depending on cache configuration. Bishop et al [5] show that 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.

5.4.3 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. Figure 5.3a) and 5.3b) show the total energy normalized to the 50MHz value for the naive and aggressive policy, respectively.

From Figure 5.3 we see similar trends for each workload. For the naive policy (Figure 5.3a), energy initially decreases, then increases dramatically as processor power increases. For the aggressive policy (Figure 5.3b), energy increases as frequency increases. We note that for the aggressive policy, the total energy increases more



a) Naive Policy b) Aggressive Policy

Figure 5.3: Miss Ratio Effects on DVS: Normalized to the 50MHz value.

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 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.

5.4.4 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 [54] 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. Figure 4.1 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:

$$T_{act} = t_{access} N_{misses} \tag{5.1}$$

$$T_{L \to act} = t_{L \to act} N_{misses} \tag{5.2}$$

$$T_L = \frac{1}{f}(N_{insts} - N_{misses}) \tag{5.3}$$

$$T_{exec} = T_{act} + T_{L \to act} + T_L \tag{5.4}$$

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:

$$E_{cpu} = T_{exec} P_{cpu}(f) + T_{residue} P_{leakage} \tag{5.5}$$

$$E_{mem} = T_{act} P_{act} + T_L P_L + T_{L \to act} P_{L \to act} + T_{residual} P_{powerdown} \tag{5.6}$$

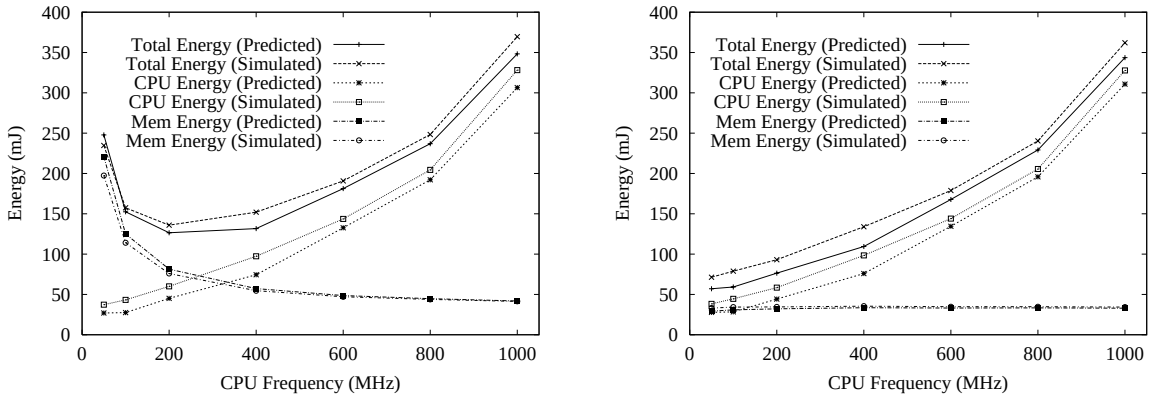
$$E_{total} = E_{cpu} + E_{mem} \tag{5.7}$$

Note all parameters required to solve the above equations are either available from the hardware specifications (t_{access}, $t_{L \to act}$, P_{act}, P_L, $P_{L \to act}$, $P_{powerdown}$, $P_{cpu}(f)$, $P_{leakage}$) 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. Figure 5.4 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



a) Naive b) Aggressive

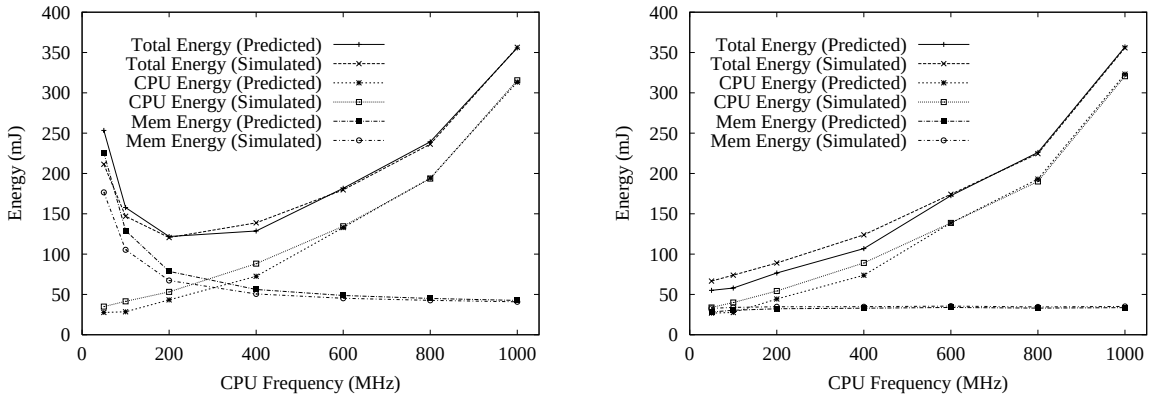
Figure 5.4: Energy Prediction – Inorder Processor

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 [54] 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 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 Figure 5.5).

5.5 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 isola-



a) Naive b) Aggressive

Figure 5.5: Energy Prediction – Out-of-order Processor

tion – 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 execu-

tion 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.

Chapter 6

Conclusion

The first major part of this work builds a compelling case for cooperative hardware/software policies that can exploit the power management features offered by new memory technologies to dramatically improve the energy efficiency of main memory. We use trace-driven simulations of a set of personal productivity applications and execution-driven simulation of integer SPEC2000 benchmarks to evaluate static and dynamic hardware policies that determine the power states of each memory chip. We show that statically assigning the nap mode as the base power mode for all memory chips in a system is a successful strategy, achieving an Energy•Delay of 15% to 40% of static active mode. We show that power aware page allocation can improve energy efficiency by up to an additional 30%. Using power-aware page allocation in conjunction with hardware policies that dynamically adjust the power mode of each individual memory chip based on thresholds of inactivity can provide 6% to 55% improvement in Energy•Delay over the best static hardware policy and offers 99% to 80% improvement over a traditional full-power memory system with random page placement.

To further improve the above dynamic hardware policy, we present a general and efficient power management strategy for a class of non-scalable devices. By identifying the variety and non-stationarity of the device access patterns, we propose a probability-based empirical approach. Our strategy uses a window to dynamically monitor the recent idle time distribution, and uses a tree to efficiently update the energy costs and output the optimal threshold. The simulation results show that our strategy adapts to various devices and workloads, and approaches to the optimal with

a distance of at most 7.1%. Against the default non-transition mode, our strategy achieves a total energy saving of 67%–87%. Through sensitivity analysis, we show that our strategy is not sensitive to the window size of a certain range, and justify our choice of a window-based approach. We also verify our guideline for choosing appropriate threshold range.

Finally we show there is a synergistic effect between power-aware memory control and dynamic voltage scaling (DVS) of the processor. 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. We propose a technique for DVS to choose appropriate processor voltage/frequency setting with the consideration of memory's contribution on both execution time and energy consumption.

Despite our work on the interaction between processor and memory, no comprehensive research has been done on the global power optimization in the system level. It is especially important for energy-balanced systems (i.e. mobile system, sensor networks, etc.), where each component (i.e. processor, memory, network, etc.) consumes energy in the same order of magnitude. A locally optimal policy might not save energy globally. For instance, if the latency introduced by a disk spin-down is in the critical path, the additional energy incurred on other components might be greater than that saved from the disk. Therefore I plan to investigate a global energy optimization policy which takes into account the power characteristics of all major

components and their access dependencies. It works as a central module coordinating the local optimizers. A challenge is to develop a model that captures the interactions among the related components and the tradeoff between performance and energy consumption, and can be used to devise an optimizing algorithm.

Besides stand-alone systems, I am also interested in the energy optimization problem for distributed systems, i.e. ad-hoc networks, sensor networks, and even large-scale computing clusters, etc. The challenge for this class of problems is that no global information is available for deploying a central manager although a global optimization is still desired. First I need to define unified metrics to describe the overall system performance and energy efficiency. Then I intend to develop a model that formulates the optimal relationship between performance and energy efficiency so that system designers can use it to direct their designs and evaluate their implementations.

Bibliography

[1] *SPEC: Standard Performance Evaluation Corporation*, September 2000. `http://www.spec.org/`.

[2] M. Anand, E. Nightingale, and J. Flinn. Self-Tuning Wireless Network Power Management. In *Proceedings of the 9th Annual International Conference on Mobile Computing and Networking*, 2003.

[3] Mary Baker, Satoshi Asami, Etienne Deprit, John Ousterhout, and Margo Seltzer. Non-volatile Memory for Fast, Reliable File Systems. In *Proceedings of the 5th International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 10–22, October 1992.

[4] Brian N. Bershad, Dennis Lee, Theodore H. Romer, and J. Bradley Chen. Avoiding Conflict Misses Dynamically in Large Direct-Mapped Caches. In *Proceedings of the 6th International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 158–170, October 1994.

[5] B. Bishop, T. Kelliher, and M. Irwin. A Detailed Analysis of MediaBench. In *1999 IEEE Workshop on Signal Processing Systems*, November 1999.

[6] W. Bolosky, M. Scott, and R. Fitzgerald. Simple But Effective Techniques for NUMA Memory Management. In *Proceedings of the ACM Symposium on Operating Systems Principles*, pages 19–31, December 1989.

[7] A. Borodin and R. El-Yaniv. *Online Computation and Competitive Analysis.* Cambridge University Press, 1998.

[8] Thomas D. Burd and Robert W. Brodersen. Processor Design for Portable Systems. *Journal of VLSI Signal Processing*, 13(2/3):203–222, August/September 1996.

[9] Doug C. Burger, Todd M. Austin, and Steve Bennett. Evaluating Future Microprocessors-the SimpleScalar Tool Set. Technical Report 1308, University of Wisconsin–Madison Computer Sciences Department, July 1996.

[10] J. S. Chase, D. C. Anderson, P. N. Thakar, and A. M. Vahdat. Managing Energy and Server Resources in Hosting Centres. In *Proceedings of the 18th ACM Symposium on Operating Systems Principles*, October 2001.

[11] E. Y. Chung, L. Benini, A. Bogliolo, and G. D. Micheli. Dynamic Power Management for Non-Stationary Service Requests. In *Proceedings of the Design Automation and Test Europe*, 1999.

[12] Julio L. da Silvia, Francky Catthoor, Diederik Verkest, and Hugo De Man. Power Exploration for Dynamic Data Types through Virtual Memory Management Refinement. In *Proceedings of the International Symposium on Low Power Electronics and Design*, pages 311–316, August 1998.

[13] V. Delaluz, M. Kandemir, and I. Kolcu. Automatic Data Migration for Reducing Energy. In *Proceedings of 38th Design Automation Conference*, June 2002.

[14] V. Delaluz, M. Kandemir, N. Vijaykrishnan, A. Sivasubramaniam, and M.J. Irwin. DRAM Energy Management Using Software and Hardware Directed Power Mode Control. In *Proceedings of 7th International Conference on High-Performance Computer Architecture*, January 2001.

[15] Fred Douglis, Ramon Caceres, Brian Marsh, Frans Kaashoek, Kai Li, and Joshua Tauber. Storage Alternatives for Mobile Computers. In *Proceedings of the First Symposium on Operating Systems Design and Implementation (OSDI)*, pages 25–37, November 1994. Monterey, CA.

[16] Fred Douglis, P. Krishnan, and Brian Bershad. Adaptive Disk Spin-down Policies for Mobile Computers. In *2nd USENIX Symposium on Mobile and Location-Independent Computing*, April 1995. Monterey CA.

[17] Fred Douglis, P. Krishnan, and Brian Marsh. Thwarting the Power Hungry Disk. In *Proceedings of the 1994 Winter USENIX Conference*, pages 293–306, January 1994.

[18] 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*, 2001.

[19] X. Fan, C. S. Ellis, and A. R. Lebeck. Modeling of DRAM Power Control Policies using Deterministic and Stochastic Petri Nets. In *Proceedings of the Workshop on Power Aware Computing Systems*, 2002.

[20] K. I. Farkas, J. Flinn, G. Back, D. Grunwald, and J. Anderson. Quantifying the Energy Consumption of a Pocket Computer and a Java Virtual Machine. In *Proceedings of the 2000 ACM Sigmetrics Conference on Measurement and Modeling of Computer Systems*, June 2000.

[21] K. Flautner and T. Mudge. Vertigo: Automatic Performance Setting for Linux. In *Symp. on Operating Systems Design and Implementation (OSDI)*, 2002.

[22] Krisztin Flautner, Steve Reinhardt, and Trevor Mudge. Automatic performance setting for dynamic voltage scaling. In *The seventh annual international conference on Mobile computing and networking 2001*, pages 260–271, 2001.

[23] Jason Flinn and M. Satyanarayanan. PowerScope: A Tool for Profiling the Energy Usage of Mobile Applications. In *Workshop on Mobile Computing Systems and Applications (WMCSA)*, pages 2–10, February 1999.

[24] R.J. Fowler and A.L. Cox. The Implementation of a Coherent Memory Abstraction on a NUMA Multiprocessor: Experiences with PLATINUM. In *Proceedings of the ACM Symposium on Operating Systems Principles*, pages 32–44, December 1989.

[25] Ricardo Gonzalez and Mark Horowitz. Energy Dissipation in General Purpose Microprocessors. In *Proceedings of the IEEE International Symposium on Low Power Electronics*, October 1995.

[26] Kinshuk Govil, Edwin Chan, and Hal Wasserman. Comparing algorithm for dynamic speed-setting of a low-power CPU. In *Proceedings of first annual international conference on Mobile computing and networking*, November 1995.

[27] Dirk Grunwald, Philip Levis, Keith I. Farkas, Charles B. Morrey III, and Michael Neufeld. Policies for Dynamic Clock Scheduling. In *4th Symposium on Operating System Design and Implementation*, October 2000.

[28] Nikolaos B. I. Hajj, Constantine Polychronopoulos, and George Stamoulis. Architectural and Compiler Support for Energy Reduction in the Memory Hierarchy of High Performance Microprocessors. In *Proceedings of the International Symposium on Low Power Electronics and Design*, pages 70–75, August 1998.

[29] D. Helmbold, D. Long, and B. Sherrod. A Dynamic Disk Spin-Down Technique for Mobile Computing. In *Proc. of the 2nd ACM International Conf. on Mobile Computing (MOBICOM96)*, pages 130–142, November 1996.

[30] Patric Hicks, Matthew Walnock, and Robert M. Owens. Analysis of Power Consumption in Memory Hierarchies. In *Proceedings of the International Symposium on Low Power Electronics and Design*, pages 239–242, August 1997.

[31] Z. Hu, S. Kaxiras, and M. Martonosi. Let Caches Decay: Reducing Leakage Energy via Exploitation of Cache Generational Behavior. *ACM Transactions on Computer Systems*, 20(2):161–190, May 2002.

[32] H. Huang, P. Pillai, and K. G. Shin. Design and Implementation of Power-Aware Virtual Memory. In *Proceedings of USENIX Annual Technical Conference*, June 2003.

[33] C.-H. Hwang, C. Allen, and H. Wu. A Predictive System Shutdown Method for Energy Saving of Event-Driven Computation. In *Proceedings of the IEEE/ACM International Conference on Computer Aided Design*, 1996.

[34] IBM Corporation. *IBM Hard Disk Drive: Ultrastar 36ZX*. `http://www.storage.ibm.com/hdd/ultra/ul36zx.htm`.

[35] Tomasz Imielinski, Monish Gupta, and Sarma Peyyeti. Energy Efficient Data Filtering and Communications in Mobile Wireless Computing. In *Proceedings of Usenix Symposium on Location Dependent Computing*, April 1995.

[36] Infineon Technologies. *HYB39L256160AC/T 256Mbit 3.3V Mobile-RAM*, 2002. `http://www.infineon.com/cmc_upload/documents/079/004/DS_256M_LS_33V_200%2-_12_20.pdf`.

[37] Intel Corporation, Microsoft Corporation, and Toshiba Corporation. Advanced Configuration and Power Interface Specification. `http://www.teleport.com/~acpi`, December 1996.

[38] S. Irani, S. Shukla, and R. Gupta. Competitive Analysis of Dynamic Power Management Strategies for Systems with Multiple Power Saving States. In *Design Automation and Test Conference*, 2002.

[39] S. Irani, S. K. Shukla, and R. K. Gupta. Adaptive Probability-Based Power Management Strategies. Technical Report UCI-ICS-01-58, Matsushita Information Technology Laboratory, September 2001.

[40] Toni Juan, Tomas Lang, and Juan J. Navarro. Reducing TLB Power Requirements. In *Proceedings of the International Symposium on Low Power Electronics and Design*, pages 196–201, August 1997.

[41] A. Karlin, M. Manasse, L. McGeoch, and S. Owicki. Randomized Competitive Algorithms for Non-uniform Problems. In *First Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 301–309, 1990.

[42] Anna R. Karlin, Claire Kenyon, and Dana Randall. Dynamic TCP Acknowledgement and Other Stories about e/(e-1). In *ACM Symposium on Theory of Computing*, pages 502–509, 2001.

[43] Richard E. Kessler and Mark D. Hill. Page Placement Algorithms for Large Real-Index Caches. *ACM Transactions on Computer Systems*, 10(4):338–359, 1992.

[44] David Kotz and Kobby Essien. Analysis of a Campus-wide Wireless Network. In *Proceedings of the 8th Annual International Conference on Mobile Computing and Networking*, pages 107–118, September 2002.

[45] R. Kravets and P. Krishnan. Power Management Techniques for Mobile Communication. In *Proc. of the 4th International Conf. on Mobile Computing and Networking (MOBICOM98)*, pages 157–168, October 1998.

[46] 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.

[47] P. Krishnan, P. Long, and J. Vitter. Adaptive Disk Spin-Down via Optimal Rent-to-Buy in Probabilistic Environments. In *Proceedings of the 12th International Conference on Machine Learning*, pages 322–330, July 1995.

[48] Sven O. Krumke. Online Optimization: Competitive Analysis and Beyond. `http://citeseer.nj.nec.com/594237.html`.

[49] Rick LaRowe and Carla Ellis. Experimental Comparison of Memory Management Policies for NUMA Multiprocessors. *ACM Transactions on Computer Systems*, 9(4):319–363, November 1991.

[50] R. P. LaRowe Jr., C. S. Ellis, and L. S. Kaplan. The Robustness of NUMA Memory Management. In *Proceedings of the ACM Symposium on Operating Systems Principles*, pages 137–151, October 1991.

[51] Alvin R. Lebeck, Xiaobo Fan, Heng Zeng, and Carla S. Ellis. Power Aware Page Allocation. In *Proceedings of Ninth International Conference on Architectural Support for Programming Languages and Operating System (ASPLOS IX)*, November 2000.

[52] 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), December 1997.

[53] Dennis C. Lee, Patrick J. Crowley, Jean-Loup Baer, Thomas E. Anderson, and Brian N. Bershad. Execution characteristics of desktop applications on Windows NT. In *Proceedings of the 25th Annual International Symposium on Computer Architecture*, pages 27–38, June 1998.

[54] Steve Leibson. XScale (StrongARM-2) Muscles In. *Microprocessor*, September 2000.

[55] Kester Li, Roger Kumpf, Paul Horton, and Thomas Anderson. A Quantitative Analysis of Disk Drive Power Management in Portable Computers. In *USENIX Association Winter Technical Conference Proceedings*, pages 279–291, 1994.

[56] J. R. Lorch and A. J. Smith. Improving Dynamic Voltage Scaling Algorithms with PACE. In *Proceedings of the 2003 ACM SIGMETRICS Conference*, 2001.

[57] Jacob Lorch and Alan J. Smith. Reducing Processor Power Consumption by Improving Processor Time Management in a Single-User Operating System. In *Proc. of the 2nd ACM International Conf. on Mobile Computing (MOBICOM96)*, pages 143–154, November 1996.

[58] Jacob Lorch and Alan J. Smith. Scheduling Techniques for Reducing Processor Energy Use in MacOS. *Wireless Networks*, 3(5):311–324, October 1997.

[59] Y. Lu and G. DeMicheli. Adaptive Hard Disk Power Management on Personal Computers. In *Proceedings of the Great Lakes Symposium on VLSI*, 1999.

[60] Srilatha Manne, Artur Klauser, and Dirk Grunwald. Pipeline Gating: Speculation Control For Energy Reduction. In *Proceedings of the 25th Annual International Symposium on Computer Architecture*, pages 132–141, June 1998.

[61] T. Martin. *Balancing batteries, power and performance: System issues in CPU speed-setting for mobile computing.* PhD thesis, Carnegie Mellon University, 1999.

[62] 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.

[63] 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.

[64] Huzefa Mehta, Robert M. Owens, Mary J. Irwin, Rita Chen, and Debashree Ghosh. Techniques for Low Energy Software. In *Proceedings of the International Symposium on Low Power Electronics and Design*, pages 72–75, August 1997.

[65] Micron Technologies. *256Mb and 512Mb: ×16 TwinDie MOBILE SDRAM*, 2003. `http://www.micron.com/products/dram/mobile/part.aspx?part=MT48V32M16S2F%G-8`.

[66] Pat Morin. Online Algorithms and Game Theory.

[67] 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.

[68] Enric Musoll, Tomas Lang, and Jordi Cortadella. Exploiting the Locality of Memory References to Reduce the Address Bus Energy. In *Proceedings of the International Symposium on Low Power Electronics and Design*, pages 202–207, August 1997.

[69] B. Noble, M. Price, and M. Satyanarayanan. A Programming Interface for Application-aware Adaptation in Mobile Computing. *Computing Systems*, 8(4):345–363, 1995.

[70] University of Michigan and University of Colorado. The SimpleScalar-Arm Power Modeling Project. http://www.eecs.umich.edu/ tnm/power/.

[71] Taku Ohsawa, Koji Kai, and Kazuaki Murakami. Optimizing the DRAM Refresh Count for Merged DRAM/Logic LSIs. In *Proceedings of the International Symposium on Low Power Electronics and Design*, pages 82–87, August 1998.

[72] G. A. Paleologo, L. Benini, A. Bogliolo, and G. D. Micheli. Policy Optimization for Dynamic Power Management. In *Proceedings of the Design Automation Conference*, 1998.

[73] T. Pering, T. Burd, and R. Brodersen. Dynamic Voltage Scaling and the Design of a Low-Power Microprocessor System. //www.cs.colorado.edu/ grunwald/LowPowerWorkshop/agenda.html, June 1998.

[74] T. Pering, T. Burd, and R. Brodersen. Voltage Scheduling in the lpARM Microprocessor System. In *Proceedings of International Symposium on Low Power Electronics and Design*, 2000.

[75] Trevor Pering, Thomas D. Burd, and Robert W. Brodersen. The Simulation and Evaluation of Dynamic Scaling Algorithms. In *Proceedings of the International Symposium on Low Power Electronics and Design*, August 1998.

[76] Padmanabhan Pillai and Kang 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.

[77] Johan Pouwelse, Koen Langendoen, and Henk 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.

[78] Q. Qiu and M. Pedram. Dynamic Power Management Based on Continuous-Time Markov Decision Processes. In *Proceedings of the Design Automation Conference*, 1999.

[79] Q. Qiu, Q. Wu, and M. Pedram. Stochastic Modeling of a Power-Managed System: Construction and Optimization. In *Proceedings of the International Symposium on Low Power Electronics and Design*, 1999.

[80] K. Rajamani and C. Lefurgy. On Evaluating Request-Distribution Schemes for Saving Energy in Server Clusters. In *Proceedings of IEEE International Symposium on Performance Analysis of Systems and Software*, March 2003.

[81] Rambus. *RDRAM*, 1999. `http://www.rambus.com/`.

[82] Theodore H. Romer, Wayne H. Ohlrich, Anna R. Karlin, and Brian N. Bershad. Reducing TLB and Memory Overhead Using Online Superpage Promotion. In *Proceedings of the 22nd International Symposium on Computer Architecture*, pages 176–187, June 1995.

[83] C. Ruemmler and J. Wilkes. UNIX Disk Access Patterns. In *Proceedings of the 1993 Winter USENIX Conference*, January 1999.

[84] Samsung Electronics. *4M×16Bit×4 Banks Mobile SDRAM in 54FBGA*, 2004. `http://www.samsung.com/Products/Semiconductor/DRAM/MobileSDRAM/256Mbit/%K4M56163PE/ds_k4m56163pe-r_bg_f-r01.pdf`.

[85] T. Simunic, L. Benini, P. Glynn, and G. D. Micheli. Dynamic Power Management for Portable System. In *Proceedings of the MOBICOMM 2000*, 2000.

[86] Mark Stemm and Randy Katz. Measuring and Reducing Energy Consumption of Network Interfaces in Hand-Held Devices. In *Proceedings of 3rd International Workshop on Mobile Multimedia Communications (MoMuC-3)*, September 1996.

[87] 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.

[88] Vivek Tiwari, Sharad Malik, and Andrew Wolfe. Power Analysis of Embedded Software: A First Step Towards Software Power Minimization. *IEEE Transactions on Very Large Scale Integration*, 2(4):437–445, December 1994.

[89] B. Verghese, S. Devine, A. Gupta, and M. Rosenblum. Operating System Support for Improving Data Locality on CC-NUMA Compute Servers. In *Proceedings, Architectural Support for Programming Languages and Operating Systems*, pages 279–289, October 1996.

[90] Mark Weiser, Brent Welch, Alan Demers, and Scott Shenker. Scheduling for Reduced CPU Energy. In *USENIX Association, Proceedings of First Symposium on Operating Systems Design and Implementation (OSDI)*, November 1994. Monterey CA.

[91] John Wilkes. Predictive Power Conservation. Technical Report HPL-CSP-92-5, Hewlett-Packard Labs, February 1992.

[92] W. Yuan and K. Nahrstedt. Energy-Efficient Soft Real-Time CPU Scheduling for Mobile Multimedia Systems. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles*, 2003.

[93] H. Zeng, C. Ellis, A. Lebeck, and A. Vahdat. ECOSystem: Managing Energy as a First Class Operating System Resource. In *Proceedings of Tenth International Conference on Architectural Support for Programming Languages and Operating System (ASPLOS IX)*, October 2002.

Biography

Xiaobo Fan was born in Taiyuan, Shanxi, China, on November 20, 1973. He was awarded the Bachelor and Master degrees in Computer Science from Tsinghua University, Beijing, China, in 1996 and 1998 respectively. For his undergraduate school career, he won two Motorola Scholarships, one Excellent Student Scholarship. At Duke University he was supported by Duke Fellowship for one year, and won the Best Second Year Project award and Best Ph.D. Preliminary Exam award.

Publications

Xiaobo Fan, Carla S. Ellis, Alvin R. Lebeck. The Synergy Between Power-aware Memory Systems and Processor Voltage Scaling. In *Proceedings of the Workshop on Power-Aware Computer Systems (PACS'03), Dec. 2003.*

Xiaobo Fan, Carla S. Ellis, Alvin R. Lebeck. Modeling of DRAM Power Control Policies Using Deterministic and Stochastic Petri Nets. In *Proceedings of the Workshop on Power-Aware Computer Systems (PACS'02), Feb. 2002.*

Xiaobo Fan, Carla S. Ellis, Alvin R. Lebeck. Memory Controller Policies for DRAM Power Management. In *Proceedings of the International Symposium on Low Power Electronics and Design (ISLPED'01), pages 129-134, Aug. 2001.*

Alvin R. Lebeck, Xiaobo Fan, Heng Zeng, Carla S. Ellis. Power Aware Page Allocation. In *Proceedings of the Ninth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS IX), November 2000.*

Abhijit Vijay, Carla S. Ellis, Xiaobo Fan. Experiences with an Inbuilding Location Tracking System: Uhuru. In *Proceedings of the IEEE Int'l Symp. on Personal, Indoor, and Mobile Radio Communications (PIMRC), Sept. 2003.*

Xiaobo Fan, Chuang Lin, Jianping Wu, Ke Xu. Performance Modeling and Analysis of a Distributed Router. In *Chinese Journal of Computers, Vol.22 No.11 Nov. 1999.*

Xiaobo Fan, Carla S. Ellis, Alvin R. Lebeck. General and Efficient Power Management for Non-scalable Devices. Under Submission, 2003.