Power Aware Page Allocation

Alvin R. Lebeck, Xiaobo Fan, Heng Zeng, Carla Ellis
Department of Computer Science
Duke University
Durham, NC 27708 USA
{alvy,xiaobo,zengh,carla}@cs.duke.edu

Abstract

One of the major challenges of post-PC computing is the need to reduce energy consumption, thereby extending the lifetime of the batteries that power these mobile devices. Memory is a particularly important target for efforts to improve energy efficiency. Memory technology is becoming available that offers power management features such as the ability to put individual chips in any one of several different power modes. In this paper we explore the interaction of page placement with static and dynamic hardware policies to exploit these emerging hardware features. In particular, we consider page allocation policies that can be employed by an informed operating system to complement the hardware power management strategies. We perform experiments using two complementary simulation environments: a trace-driven simulator with workload traces that are representative of mobile computing and an execution-driven simulator with a detailed processor/memory model and a more memory-intensive set of benchmarks (SPEC2000). Our results make a compelling case for a cooperative hardware/software approach for exploiting power-aware memory, with down to as little as 45% of the Energy•Delay for the best static policy and 1% to 20% of the Energy•Delay for a traditional full-power memory.

1. INTRODUCTION

One of the major challenges of the post-PC environment—encompassing ubiquitous mobile, embedded, and wireless devices—is the need to reduce the energy consumed in their operation, thereby extending the lifetime of the batteries that power them. Power consumption is an issue that extends well beyond the realm of battery-powered mobile devices to any computing platform in which the production of heat or fan noise is a consideration (e.g., medical applications). Energy efficiency of computers is also desirable from the economic and environmental points of view.

Sustained exponential growth in processor performance and memory density means that embedded processors and handheld devices can soon have performance characteristics comparable to today's workstations. This increased performance is usually accompanied by increased power consumption. Memory is a particularly important target for efforts to address the energy efficiency issue. Instructions invoking memory operations have a relatively high power cost, both within the processor and in the memory system [43]. Intel's guidelines for mobile power [19] indicate that the target for main memory should be approximately 4% of the power budget (e.g. an average 1.3W for 96MB) for year 2000 laptops. This percentage can dramatically increase in systems with low power processors (e.g., Transmeta Crusoe [15]), displays [35], or without hard disks. Recent studies [10] show that memory system behavior can produce variations in energy consumption of 100% for a pocket computer with the above characteristics versus 9% for a conventional laptop with a high power processor.

Since many small devices have no secondary storage and rely on memory to retain data, there are power costs for memory even in otherwise idle systems. The amount of memory available in mobile devices is expanding with each new model to support more demanding applications (e.g., multimedia) while the demand for longer battery life also continues to grow significantly.

Hardware components, such as memory chips, are becoming available that offer power management features. In particular, we consider power-aware DRAM chips that support several different power modes: active, standby, nap and powerdown in order of decreasing power consumption but increasing access time. Our goal in this work is to determine how to exploit these emerging hardware features for the most effective main memory power management.

Specifically, we ask two basic questions:

1. How can the various power modes available in state-of-the-art DRAM devices be utilized? We consider both static and dynamic hardware policies for determining the power state. Our dynamic scheme uses the time between DRAM chip accesses to determine power state transitions.

2. What is the effect of code and data placement within such power-aware memory chips? Thus we consider page allocation strategies that complement the ability of the hardware to adjust power modes.

Our work is based on the premise that a cooperative hardware/software approach will offer expanded opportunities for energy efficiency. A primary contribution of this paper is a quantitative study that explores the interaction of virtual memory page allocation with dynamic hardware policies to orchestrate the use of power modes provided in emerging DRAM devices.

We measure the energy savings within the memory system and any additional delay in execution time resulting from these power management strategies, expressed in terms of an Energy•Delay metric. Using trace-driven simulation with a simplified processor and memory system model we evaluate our ideas for a set of productivity applications as a workload representative of mobile laptop and handheld devices. We also use an execution-driven simulator with a more detailed processor and memory model to evaluate a set of programs from the integer SPEC2000 suite that place higher demands on the memory system than the available traces.

Our results show the following:

- Among static policies in which every power-aware DRAM chip in the system resides in the same base power mode between accesses, choosing the nap mode as the base achieves the lowest Energy•Delay product for our workload (15% to 40% of staying in active mode).

- Power-aware page allocation by an informed operating system coupled with dynamic hardware policies can dramatically improve energy efficiency of memory. Power-aware allocation allows a 6% to 55% improvement in Energy•Delay over the best static hardware policy.

- Power-aware page allocation when used with static hardware policies can improve Energy•Delay by up to 30%.

- Dynamic hardware policies without informed OS support (i.e., using random page allocation) do not improve energy efficiency as measured by Energy•Delay.

In the next section, we describe the power-managed memory technology upon which this study is based and present related work. Section 3 describes the policies that determine which power mode each chip should be in. Then, in Section 4, we discuss simple page allocation strategies that exploit the power management features of the hardware. Section 5 presents our experimental methods and results are presented in Section 6. Finally, we conclude in Section 7 and describe future work.

2. BACKGROUND AND RELATED WORK

2.1 Rambus RDRAM

Memory technology has developed to respond to the needs of mobile computer designers to limit power consumption in the face of increasing demand for performance. One concrete example is Direct Rambus DRAM (RDRAM)[40]. The Direct Rambus technology delivers high bandwidth (1.6GB/sec per device), using a narrow bus topology operating at a high clock rate. As a result, each RDRAM chip can be activated independently. RDRAM offers four power modes: active, standby, nap, and powerdown. Because of the narrow topology, each chip can be independently set to an appropriate power state. Conventional DRAMs generally require multiple active chips to achieve high bandwidth. Whereas we could apply the ideas presented in this paper to these conventional memory systems, it would sacrifice bandwidth. By adopting the RDRAM model, we can concentrate on the interactions of page allocation with the power modes without the concern for a tradeoff between bandwidth and energy consumption.

An RDRAM device must be in active mode while performing a read or write transaction. Active mode consumes the most power. A chip that is not servicing a memory request can be in any of the lower power states. However, these states incur additional delay for clock resynchronization. Standby is fast and uses 60% of the power of active mode. Greater power savings can be achieved by using nap mode (10% of the power of active) with an additional resynchronization time required to transition to the active state in order to service a memory request. Powerdown mode has the minimal power consumption (1% of active), but a significant delay for clock synchronization (100 times that needed by nap mode) to enter the active state. Figure 1 shows the power states and their relative power costs as well as the possible transitions and relative transition times into active mode.

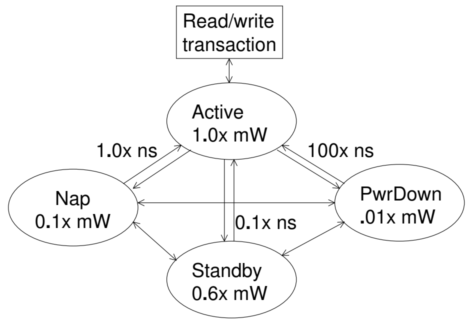


Figure 1: RDRAM Power States

The challenge for the laptop or handheld designer is to utilize these modes effectively. It is not only the availability of these power states but the ability to transition between them dynamically on a per-chip basis that gives the RDRAM its potential for power management.

2.2 Power-Aware DRAM Model

Rather than trying to model the full complexity of the Direct RDRAM specifications, we incorporate the essential features (i.e., multiple, independently controlled memory chips with multiple power states) into an abstract model of a Power-Aware DRAM (PADRAM). We can choose parameters that are consistent with the power modes, the relative power costs, and relative resynchronization times given in Figure 1 so our results are relevant to RDRAM; however, we do not claim to have precise numbers for the power consumed in each state and state transition of any particular RDRAM implementation. We make simplifying assumptions about the power consumption during state transitions and we concentrate only on the transition times that impose additional latency on a memory request.

We focus on improving the energy consumption of main memory, ignoring the energy used by all other system components (including processor and cache). In our studies, the processor and cache affect memory energy efficiency by influencing execution time and miss rate (the number of DRAM accesses).

2.3 Related Work

Architectural studies have examined the impact of software structure on power consumption [6, 34, 44]. Other architectural studies investigated processor design [4, 33, 39], focused specifically on the memory hierarchy [14, 17, 21, 36], or examined ways to optimize DRAM refresh counts [38]. RDRAM was clearly designed to enable designers to create pools of devices in various power states, as it is stated in various documentation [40]. However, to our knowledge, ours is the first quantitative study to explore the interaction of page allocation with dynamic hardware policies to orchestrate the use of power modes being provided in emerging PADRAM-class memory devices.

A novel aspect of our work is the cooperative hardware/OS approach to exploit PADRAM features. Previous OS-level studies focusing on power management include work on scheduling for low power processor modes [31, 32, 46], spindown policies for disks and alternatives [1, 7, 8, 9, 16, 25, 30, 47], and managing wireless communication [18, 24, 42]. A consortium of companies has developed a specification [20] 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. Recent work with Odyssey [37, 11] demonstrates how system support for application-aware adaptation can benefit energy efficiency. Common themes that appear in these power management strategies are the identification / prediction of idleness in the activity patterns of a component and techniques that attempt to change those activity patterns. A particularly valuable approach is based on the "ski rent-to-buy" problem formulation for competitive algorithms [22, 25].

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 [2, 23]. Other recent work has studied page placement aimed at improving TLB performance [41] or NUMA multiprocessor memory access [27, 28, 45, 12, 3]. Each of these problems bears some resemblance to the issues we face since they all attempt to exploit the flexibility available in mapping virtual to physical pages.

3. HARDWARE POWER MANAGEMENT

This section explains various hardware policies for controlling PADRAM 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 Static Policies

The static schemes we investigate correspond to placing all PADRAM chips in a single power state. We note that for an access to occur, the PADRAM 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 PADRAM 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 PADRAM 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.

3.2 Dynamic Policies

To obtain higher performance and energy efficiency we must relax the constraint that each PADRAM 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.

Dynamic power state management exploits locality of reference to individual PADRAM chips. Reference locality is determined in part by the algorithm/data structures and in part by the mapping of program virtual addresses to physical addresses. The next section discusses how the operating system can influence energy efficiency through physical page allocation. Source code and data structure transformations for improving energy efficiency is an important and interesting topic, but is beyond the scope of this paper.

4. PAGE ALLOCATION

An important contribution of this paper is to re-examine virtual memory page allocation policies in light of new PADRAM technology. Previous page allocation studies ignored which actual DRAM chips contained the allocated page frame. In contrast, our work focuses 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 PADRAM chips.

To determine the benefits of power aware page allocation (see Section 6) we compare random and the well-known sequential first-touch placement policies. Our first policy randomly chooses a PADRAM 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 PADRAM chip before moving on to the next. This scheme minimizes the number of PADRAM chips utilized for a given application. Therefore, the hardware can automatically place unused PADRAMs 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 PADRAM chip. We also assume that physical addresses are not interleaved across PADRAM chips. We can interleave at the word, cache line, or page granularity within the PADRAM 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 PADRAM 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. Of course, this could be repeated, but we leave multiple "corrections" as future work.

In Section 6, 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.

5. METHODOLOGY

To evaluate energy efficiency, we use the Energy•Delay product [13]. This metric captures our goal of achieving high performance (seconds) while minimizing energy consumption (joules). Although total system energy consumption is important, it is highly dependent on specific design choices (e.g., processor, display type, wireless network interface, etc.). Therefore, we concentrate only on PADRAM energy consumption, and ignore the energy consumed by all other system components.

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.

Table 1 shows the parameter values we use to determine energy consumption and DRAM access delay. These values were obtained from the Rambus Direct RDRAM manual [40] and from an EE Times article [26] on Rambus. At the time of writing, RDRAM vendor data sheets did not contain sufficient information on power consumption. It is important to note that precise values will vary from vendor to vendor. The values we use match the relative values provided by Rambus. While the values for a particular power state are taken from the literature, we approximate the power consumption associated with a transition between two power states as the average of the power consumed in the two states. The total energy consumption depends on the time for the transition to complete, also shown in Table 1.

5.1 Trace-Driven Simulation

The trace-driven simulator processes instruction and data address traces and uses a simplified PADRAM 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. Results for a 4-way associative L2 were qualitatively similar to the direct-mapped cache, therefore we omit them. We also model the individual PADRAM 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 PADRAM 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.

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

For our trace-driven studies 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 [29], include instruction and data accesses for several popular applications typical of those used on laptops today. Table 2 provides information on the applications we use. The first six benchmarks are from the NT traces.

5.2 Execution-Driven Simulation

To overcome the limitations of trace-driven simulation, we augmented the SimpleScalar execution-driven simulator [5] with a PADRAM model based on the detailed timing and power specifications of Rambus RDRAM. SimpleScalar

Power State/Transtion	Power	Time
Active	300mW	60ns
Standby	180mW	-
Nap	30mW	-
Powerdwn	3mW	-
Standby $\rightarrow$ Active	240mW	+6ns
Nap $\rightarrow$ Active	165mW	+60ns
Powerdwn $\rightarrow$ Active	152mW	+6000ns

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

	Benchmark	Description	Instructions Executed (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
Execution 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 2: Benchmarks

models a dynamically scheduled processor using a Register Update Unit (RUU) and a Load/Store Queue (LSQ). 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. As above, although specific values for a 4-way L2 cache differ from the direct-mapped cache, overall results are qualitively similar and thus omitted. Each cache can have up to 16 outstanding misses. The processor executes Compaq Alpha binaries.

Our PADRAM memory model uses the values from Table 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 PADRAM 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 (See Section *6.4.1*). 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 2). These three were chosen because they exhibited the highest data cache miss ratios. For all benchmarks, we use the reference input data set.

6. EXPERIMENTAL RESULTS

This section presents our results on power management for PADRAM. We begin with analysis of static hardware power state policies and their interaction with page allocation (Section 6.1). This is followed in Section 6.2 by analysis of the effects of page allocation on dynamic hardware power management policies described in Section 3. 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.

6.1 Static Power State Policies

In this section we evaluate the static policies that uniformly place all PADRAM chips in the same power state. We begin by evaluating PADRAM 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 2 shows the Energy•Delay product for the four static policies (*active*, *standby*, *nap*, and *powerdown*) normalized to the *active* policy for each program. Table 3 shows the absolute values for runtime, energy, and Energy•Delay product. From Figure 2 we see that placing all PADRAM 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 accessess. From Tables 2 and 3 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.

6.1.1 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 shows the Energy•Delay of sequential allocation normalized to the Energy•Delay of random allocation. From Figure 3a 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 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 3b) 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.

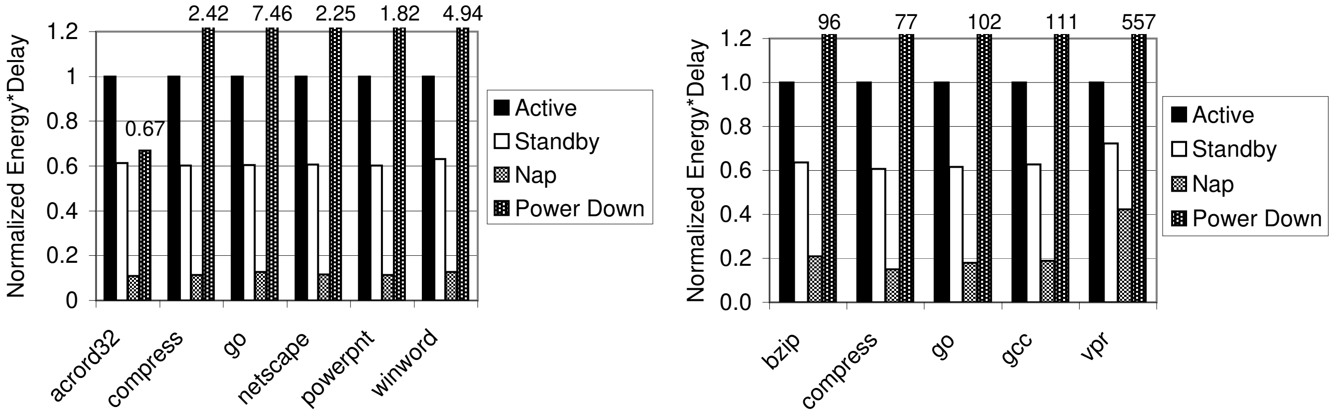
6.2 Dynamic Power State Management

We now examine more sophisticated hardware support for PADRAM power management. By dynamically determining each chip's power state based on recent references, we hope to improve overall energy efficiency. Figure 4 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 4 shows the raw data for the static nap and powerdown schemes along with the best dynamic scheme.

We determined 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. We use that bound to guide the choice of threshold values to explore. Appendix A provides details on our threshold computation. Our analysis determined that there was very little benefit for remaining in *standby*, and that the *active* to *nap* threshold should be on the order of 100's of nanoseconds, while the *nap* to *powerdown* threshold should be on the order of 10,000ns. Our trace-driven simulation results show that thresholds of 100ns/5,000ns produce the best overall Energy•Delay.

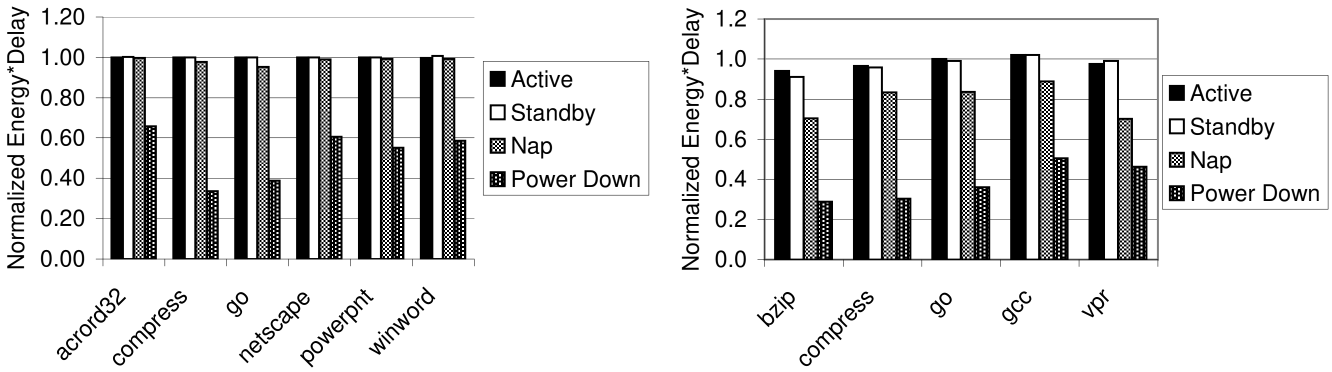
From Figure 4 we see that the combination of power aware page allocation and dynamic hardware policies can produce Energy•Delay values that are 50% to 94% of the static *nap* policy. Five of the six benchmarks achieve 80% or lower.

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 alloca-



a.) NT Traces b.) SPEC Benchmarks

Figure 2: Static Base Power State Policies and Random Page Allocation, Energy•Delay normalized to *active* policy.



a.) NT Traces b.) SPEC Benchmarks

Figure 3: Benefits of Sequential Page Allocation for Static Policies, each Energy•Delay result normalized to same policy with Random Allocation

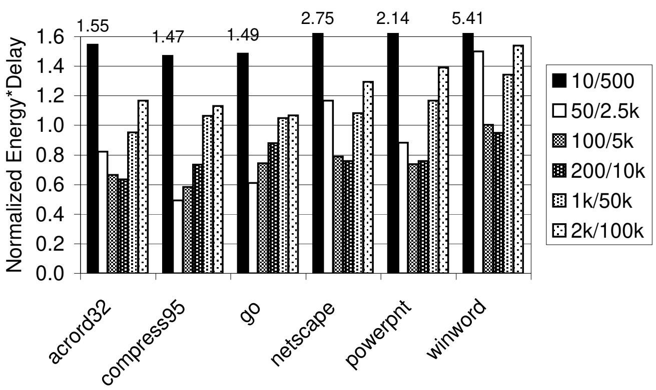


Figure 4: Dynamic Power Management and Sequential Page Allocation (NT Traces), normalized to Static *nap* policy. Each threshold combination for transitioning from *active* to *nap* (x ns) and from *nap* to *powerdown* (y ns) is represented as x/y.

tion. 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*. Section 6.4.1 discusses aspects of our RDRAM model that produce the 0ns threshold. 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*

Benchmark	Policy	DRAM Accesses	Run Time (ms)	Energy (mJ)	Energy•Delay
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: Raw Data for Static Policies with Random Allocation, Energy•Delay is defined in terms of joules×seconds

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.

6.3 Frequency-based Page Placement

The primary goal of page placement thus far has been to cluster all pages into the minimum number of PADRAM chips. In this section, we present preliminary 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 5 shows the Energy•Delay for both the frequency and sequential first-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

Benchmark	Policy	Run Time (ms)	Energy (mJ)	Energy•Delay
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 4: Raw Data for Static Nap and Power Down and Best Dynamic Policy with Sequential Allocation, Energy•Delay is defined in terms of joules×seconds

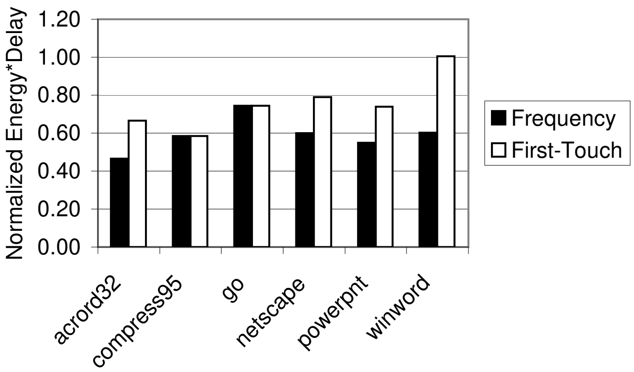


Figure 5: Frequency vs. Sequential First-Touch Page Allocation (NT Traces) for Dynamic Policy, thresholds of 100ns/5,000ns, normalized to sequential first-touch static *nap*.

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 period. 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.

6.4 Alternative DRAM Architectures

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

6.4.1 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 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 PADRAM 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.

6.4.2 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.

7. CONCLUSION

In this paper, we have built a compelling case for cooperative hardware/software policies that can exploit the power management features offered by new PADRAM memory devices, such as the Rambus RDRAM, to dramatically improve the Energy•Delay 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.

There are many opportunities left for future work with the PADRAM model of memory devices and especially with the interaction between hardware and software management. Following our belief that energy conservation should become a "first class" design goal for higher levels of system design, many of our plans explore ways to give the OS more explicit control over PADRAM power modes. This may even eventually extend into API features that allow some degree of application-level direction of memory power states. Sequential first-touch is a simple page allocation scheme. We may consider other "page coloring" techniques and further explore the movement of pages between chips to improve initial placements based on observed access patterns.

We note that our clustered page allocation has other power-related side-effects. It can also be used to reduce DRAM refresh rates. By compacting physical pages into the minimum number of internal memory banks, we can potentially eliminate refresh for entire DRAM banks in which there are no active pages.

The threshold values in our dynamic policy are an important parameter. Unfortunately, using the same threshold value for all programs and all PADRAM chips may not produce the best results. Thus, another possible direction we are exploring is a dynamic policy that attempts to adap-

[1] A DRAM page is one row of an internal DRAM bank.
[2] RDRAM has 32 internal banks, a maximum of 16 can be accessed in parallel.

tively determine the best threshold values for each chip.

Our dynamic policies have concentrated on the transition into lower power states. Policies that support pre-transitioning into higher power states, in anticipation of imminent access in a manner analogous to prefetching, may also have a role to play in improving the Energy•Delay metric of some applications.

Acknowledgments

We thank Amin Vahdat for comments on early drafts of this paper. This work supported in part by DARPA Grant DABT63-98-1-0001, NSF Grants CDA-97-2637, CDA-95-12356, EIA-99-72879, EIA-99-86024, NSF CAREER Award MIP-97-02547, Duke University, and equipment donations from Intel and Microsoft. The views and conclusions contained herein are those of the authors and should not be interpreted as representing the official policies or endorsements, either expressed or implied, of DARPA or the U.S. Government.

8. REFERENCES

[1] M. Baker, S. Asami, E. Deprit, J. Ousterhout, and M. 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.

[2] B. N. Bershad, D. Lee, T. H. Romer, and J. B. Chen. Avoiding conflict misses dynamically in large direct-mapped cach es. In *ASPLOS6*, pages 158–170, October 1994.

[3] 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.

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

[5] D. C. Burger, T. M. Austin, and S. Bennett. Evaluating future microprocessors-the simplescalar tool set. Technical Report 1308, University of Wisconsin–Madison Computer Sciences Department, July 1996.

[6] J. L. da Silvia, F. Catthoor, D. Verkest, and H. 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.

[7] F. Douglis, R. Caceres, B. Marsh, F. Kaashoek, K. Li, and J. 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.

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

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

[10] 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.

[11] J. 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.

[12] R. Fowler and A. 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.

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

[14] N. B. I. Hajj, C. Polychronopoulos, and G. 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.

[15] T. Halfhill. Transmeta breaks x86 low-power barrier. *Microprocessor Report*, February 2000.

[16] 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.

[17] P. Hicks, M. Walnock, and R. 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.

[18] T. Imielinski, M. Gupta, and S. Peyyeti. Energy Efficient Data Filtering and Communications in Mobile Wireless Computing. In *Proceedings of Usenix Symposium on Location Dependent Computing*, April 1995.

[19] Intel Corporation. Mobile Power Guidelines 2000. ftp://download.intel.com/design/mobile/intelpower/mpg99r1.pdf, December 1998.

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

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

[22] A. R. Karlin, M. S. Manasse, L. Rudolph, and D. D. Sleator. Competitive snoopy caching. *Algorithmica*, (3):79–119, 1988.

[23] R. E. Kessler and M. D. Hill. Page placement algorithms for large real-index caches. *ACM Transactions on Computer Systems*, 10(4):338–359, 1992.

[24] 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.

[25] 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.

[26] D. Lammers. IDF: Mobile Rambus spec unveiled. *EETimes Online*, February 1999. //www.eetimes.com/story/OEG19990225S0016.

[27] R. LaRowe and C. Ellis. Experimental comparison of memory management policies for NUMA multiprocessors. *ACM Transactions on Computer Systems*, 9(4):319–363, Nov. 1991.

[28] 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.

[29] D. C. Lee, P. J. Crowley, J.-L. Baer, T. E. Anderson, and B. 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.

[30] K. Li, R. Kumpf, P. Horton, and T. Anderson. A Quantitative Analysis of Disk Drive Power Management in

Portable Computers. In *USENIX Association Winter Technical Conference Proceedings*, pages 279–291, 1994.

[31] J. Lorch and A. 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.

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

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

[34] H. Mehta, R. M. Owens, M. J. Irwin, R. Chen, and D. Ghosh. Techniques for Low Energy Software. In *Proceedings of the International Symposium on Low Power Electronics and Design*, pages 72–75, August 1997.

[35] MicroOptical Corp. *Eyeglass Display*, 1999. http://www.microopticalcorp.com/.

[36] E. Musoll, T. Lang, and J. 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.

[37] B. Noble, M. Price, and M. Satyanarayanan. A programming interface for application-aware adaptation in mobile computing. *Computing Systems*, 8(4):345–363, 1995.

[38] T. Ohsawa, K. Kai, and K. 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.

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

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

[41] T. H. Romer, W. H. Ohlrich, A. R. Karlin, and B. 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.

[42] M. Stemm and R. 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.

[43] V. Tiwari, S. Malik, and A. Wolfe. Compilation Techniques for Low Energy: An Overview. In *Proc. of the 1994 IEEE Symp. on Low Power Electronics*, pages 38–39, October 1994.

[44] V. Tiwari, S. Malik, and A. 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.

[45] 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.

[46] M. Weiser, B. Welch, A. Demers, and S. Shenker. Scheduling for Reduced CPU Energy. In *USENIX Association, Proceedings of First Symposium on Operating Systems Design and Implementation (OSDI)*, November 1994. Monterey CA.

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

APPENDIX

A. DETERMINING THRESHOLD VALUES

For dynamic power state management, we benefit by staying in lower power state, while paying the cost for transitioning back to *active* on the next access. If the chip can stay in the lower power state long enough, the benefit could be greater than the cost. 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. Based on these times, we pick our idle time threshold for transitioning to a lower power state. 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 transition. 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 values from our RDRAM configuration ($P_p = 3mW, P_n = 30mW, P_s = 180mW, P_a = 300mW, P_{p\to a} = 152mW, P_{n\to a} = 165mW, P_{s\to a} = 240mW, T_{p\to a} = 6000ns, T_{n\to a} = 60ns, T_{s\to a} = 6ns, P_0 = P_a = 300mW$), we have $T_p > 9131ns$, $T_n > 103ns$, $T_s > 27ns$.

As we can see from the above lower bound, 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 $10^4 ns$.