Managing Memory for Power, Performance, and Thermal Efficiency

Matthew E. Tolentino

Dissertation submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy
in
Computer Science and Applications

February 18, 2009
Blacksburg, VA

Keywords: Memory Management, Operating Systems, Control Theory, Energy Efficiency

Managing Memory for Power, Performance, and Thermal Efficiency

Matthew E. Tolentino

ABSTRACT

Extraordinary improvements in computing performance, density, and capacity have driven rapid increases in system energy consumption, motivating the need for energy-efficient performance. Harnessing the collective computational capacity of thousands of these systems can consume megawatts of electrical power, even though many systems may be underutilized for extended periods of time. At scale, powering and cooling unused or lightly loaded systems can waste millions of dollars annually.

To combat this inefficiency, we propose system software, control systems, and architectural techniques to improve the energy efficiency of high-capacity memory systems while preserving performance. We introduce and discuss several new application-transparent, memory management algorithms as well as a formal analytical model of a power-state control system rooted in classical control theory we developed to proportionally scale memory capacity with application demand. We present a prototype implementation of this control-theoretic runtime system that we evaluate on sequential memory systems. We also present and discuss why the traditional performance-motivated approach of maximizing interleaving within memory systems is problematic and should be revisited in terms of power and thermal efficiency. We then present power-aware control techniques for improving the energy efficiency of symmetrically interleaved memory systems. Given the limitations of traditional interleaved memory configurations, we propose and evaluate unorthodox, asymmetrically interleaved memory configurations. We show that when coupled with our control techniques, significant energy savings can be achieved without sacrificing application performance or memory bandwidth.

Acknowledgements

It is only because of the support of my colleagues, friends and family that this dissertation is now complete. I owe my gratitude to Professor Kirk W. Cameron, graduate advisor, colleague, and friend. His incessant encouragement, insightful perspective, similar approach to deadlines, and positive attitude inspired me to strive to achieve that which I wasn't convinced was possible. Thank you, Kirk.

I owe my thanks to the members of my committee: Gary Beihl, Professor Ali Butt, Professor Dimitris Nikolopoulos, and Professor Calvin Ribbens. Their advice, suggestions, and flexibility with my rather erratic schedule have been incredibly helpful. Thank you.

Without the enthusiastic encouragement and support of my friend Dave Patterson I may not have continued graduate studies beyond the University of Washington. The talks we've had during the hard times as well as those "on the lift" between runs have been invaluable. With graduate studies nearly behind me, I'm looking forward to some epic heli-skiing as well as taking on High Rustler.

I've worked with some really great people at Virginia Tech. I've shared some great experiences and conversations while travelling, hiking, and climbing with Joseph Turner. Perhaps we'll continue some of those conversations soon; not in an office, but en-route to a summit. I've enjoyed my conversations with Rong Ge and Xizhou Feng as their perspectives have been consistently helpful and insightful. I always look forward to discussing ideas with Hari Pyla as I greatly respect his perspective, capabilities, and incredible drive to succeed. I'm also thankful to Hung-Ching Chang for working with me, even at a distance, to collect power and thermal measurements.

I’m also thankful to my Intel colleagues for listening to my ideas as well as my frustrated rants as I’ve juggled my seemingly schizophrenic identities as an academic researcher and professional engineer.

Finally, this journey would not have been possible without the support of my family. I’m deeply appreciative of my in-laws, Maria and Richard, who have been consistently supportive during this journey. They have shown me how families are supposed to support and help each other during difficult times. Of course, none of this would have been possible without the support of my dear wife, Amalie. This has been difficult for both of us, but I couldn’t have finished this journey without your love, support, encouragement, and endurance. I’m happy to report the number of pages I have left to complete is finally … zero. Thus, our lives must no longer remain in a holding pattern and it’s on to the next adventure (after a quick heli-skiing trip). Finally, having to take time away from William and Dylan to focus on research and this dissertation has been difficult. Know that your love and patience has made this endeavor easier to endure.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Rapid advances in computing performance and capacity over the past several decades have ushered in a new era of data-centric computing that is changing how we live, work, and communicate. To accommodate this trend, modern server designs are incorporating multi- as well as many-core processors and rapidly scaling memory capacity. However, increasing the computational capabilities of servers is not free since the additional capacity increases power consumption. When deployed in aggregate at data-center scale, operating and cooling these servers can consume megawatts of electrical power even though applications may only utilize a fraction of the available capacity. This inefficient gap between application demand and capacity availability amounts to millions of dollars of wasted operational expenditures annually.

Given the rapid increases in memory capacity and density, which are propelling power consumption higher, this dissertation proposes system software and architectural techniques to transparently improve the energy efficiency of high-performance, high-capacity memory systems. We introduce new operating system memory management algorithms, memory power-state control systems rooted in formal control theory, and unorthodox memory architecture adaptations that significantly improve memory energy efficiency while preserving application performance.

1.1 Current Challenges

The power consumption of server-class systems has increased substantially in recent years. According to a 2007 survey funded by AMD, the power consumption of volume and mid-range servers increased 16% and 51% respectively from 2000 to 2005 [110]. These types of servers account for 99% of the more than 1.5 million servers sold in the United States annually.

Increased server power density is cause for concern. Volume and mid-range servers consume from 100 to 200 watts per cubic foot and this number is growing. The IDC and others predict that the energy costs for servers will soon exceed hardware acquisition costs [51] . In the short term, the heat produced from increased power densities requires elaborate, costly cooling technologies both inside and outside the unit. In the long term, elevated average internal and ambient temperatures may reduce the mean-time-between-failure (MTBF) of these systems.

High power system components can be replaced with low power equivalents from mobile devices. Unfortunately, microelectronic devices operating at lower power typically provide less functionality and speed. For example, Transmeta Crusoe processors operate at significantly lower power than Intel Xeon and AMD Opteron processors. However, both Xeon and Opteron processors outperform Crusoe on many server workloads [62].

An alternative approach is to provide high power (and performance) on demand. The basic approach is to intelligently schedule component power modes to meet performance and energy constraints. Various techniques to save energy have been proposed using power-aware processors [17, 25, 26, 35, 65, 67, 98, 107, 156, 165], disks [17, 118, 172], and interconnects [123]. Recent work has shown that workload determines the proportion of the power budget consumed by server components with CPU, memory, and disk accounting for the majority share in most cases [17, 25, 114, 115].

Traditionally, server power has been dominated by the microprocessor. However, the trend towards multi-core has resulted in processor power leveling off close to 100 watts per CPU. Meanwhile, memory transistor density, functionality, and scalability have continued to increase. As a result, volume servers with 64Gbyte memory capacity composed of 4Gbyte DIMMs that consume 10 watts each are now commonplace. Figure 1.1 shows that should these trends continue, volume and mid-range servers may soon exceed 32Gbytes of main memory per processor core and memory power will dominate.

Reducing memory power in server-class systems is challenging. For example, in mobile systems users may opt for lower performance for extended battery life. In server systems, reduced performance may cause a company to lose money, customers, or both. Hence, memory power reduction techniques in servers must control and limit performance loss. Ideally, instead of placing devices (i.e. memory) in a low power state, we would like to turn the memory completely off without losing data or performance. This is extremely challenging since

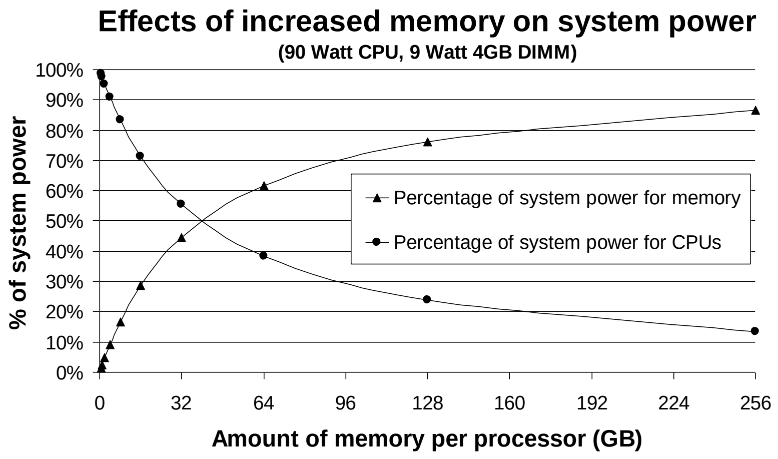


Figure 1.1. CPU and memory percentages of total system power as memory-per-processor increases.

techniques that offline memory challenge traditional operating system assumptions about an unchanging physical address space and require significant changes to memory management structures and algorithms.

There has been significant previous work to reduce memory power consumption by augmenting memory controllers with fine-grained, access-based policies [42-45, 48, 57-59, 80, 87-90, 92, 113, 117, 131, 134, 173]. While such techniques have been shown to yield significant energy savings, we wanted to explore ways to conserve memory energy that leverage the capabilities of existing or upcoming server deployments and avoid the costs of silicon redesign and consequent validation. Previous software-based memory energy studies have avoided these costs [45, 82, 88]. However, the proposed techniques are not scalable to servers with large memories (>1Gbyte) and tens of thousands of processes, typical of large-scale server deployments.

1.2 Research Objectives and Approaches

The objective of this dissertation is to improve the power and performance efficiency of memory systems. Ideally, memory energy consumption should be proportional to workload memory demand, even though application memory demand may fluctuate significantly. Server workloads often underutilize system resources over time resulting in system-level and component-level slack [12, 18, 107, 114]. For example, DVFS scheduling exploits slack during execution to reduce processor voltage and frequency to save power [25, 26, 59, 63, 67, 156, 164]. Similarly, significant energy savings have been achieved by aligning memory device power-state transitions with process scheduling [43-45, 88, 90]. These savings were realized by exploiting slack between memory accesses for devices that did not contain pages referenced by scheduled processes. Other research has shown that page allocation algorithms can have a direct effect on energy consumption [57, 59, 113, 155, 167, 168]. The common theme among these different approaches is the effectiveness of software-directed hardware component power-state transitions to exploit slack in demand for energy efficiency. In this work, our approach is twofold. First, we develop and evaluate system software techniques to realize performance and energy efficiencies in today's memory systems. Second, we propose and evaluate architectural enhancements to improve the energy efficiency of future memory systems. Thus, our work focuses on operating system memory management algorithms and techniques, device power-state control systems, and adaptation of traditional, performance-motivated memory system architectural techniques to improve memory power and performance efficiency.

Our first objective is to adapt system software to reduce the energy consumption of memory on currently available commodity hardware. In earlier work, we extended the operating system to support hot-pluggable memory for fault resiliency [150]. We leverage this prototype system, but extend the page frame allocation and management algorithms for power-management. This is challenging since techniques that change memory capacity and device power states are not supported by traditional operating systems. Further, there has been limited previous work to develop or evaluate allocation-based techniques to reduce memory energy across the spectrum of available memory technologies.

Given the underlying mechanisms to change memory device power states, our next objective is to develop an adaptive, stable control system to efficiently manage transitioning devices between multiple power states. Systemic memory demand and utilization can vary significantly based on workload. Thus, naively switching memory devices between power states runs the risk of degrading application performance. Previous work relied on a priori knowledge of workloads, static policies, or workload-specific heuristics [43-45, 88, 90, 113, 117, 131]. However, these approaches lacked the scalability, flexibility, and resiliency to handle unexpected demand variations. Our challenge is to formulate an analytically provable, stable control system to transition memory devices between multiple power states. Moreover, to minimize overhead and avoid application performance degradations, we need a control system with quantifiable transient response characteristics.

Third, we profile and analyze the power, performance, and thermal efficiency of architectural techniques designed to increase bandwidth and reduce access latencies. Server memory systems exploit architectural techniques such as interleaving, that distribute contiguous cache line accesses among multiple memory devices to improve bandwidth, and hence application performance. However, by altering memory device access patterns, memory interleaving changes the power and thermal characteristics of the memory system [13, 100, 106]. Our objective is to identify and characterize the power, performance, and thermal efficiency of these techniques within the memory system of a current system. This is challenging since gleaning insight into the power and thermal characteristics of the memory system requires extensive board, memory controller, and memory device instrumentation as well as specialized firmware and monitoring software to correlate these effects to specific interleaving dimensions.

Our final objective is to combine our power-aware memory management algorithms, control-theoretic system for managing device power state transitions, and our new-found insight into the power and performance effects of interleaving into a dynamic runtime control system capable of reducing the energy consumption of complex, highly-interleaved memory topologies. Previous work to improve the energy efficiency of memory has largely ignored the complexities of highly-interleaved memory systems. Improving the energy efficiency of interleaved systems is challenging as power-state transitions for memory regions must be coordinated across multiple devices and more importantly, not reduce achievable bandwidth. Reduced bandwidth can degrade application and system performance. Our approach is to develop interleaving-aware algorithms and a control system that exploits the use of asymmetrically interleaved configurations to improve memory energy efficiency while preserving application performance and system bandwidth.

1.3 Research Contributions

This dissertation introduces system software, memory power-state control systems, and memory system architectural enhancements to improve the energy efficiency of memory systems. More specifically:

- We address the problem of extending the operating system to support onlining and offlining memory devices for systems with dense memory topologies. Inspired by network traffic flow research, we propose several OS-level page allocation and management *shaping* techniques to proactively and reactively direct allocations to a minimal number of devices. We extend the Linux operating system to support these shaping techniques and structural enhancements on *real* systems. Our experiments using a simple history-based heuristic for controlling memory device power-state transitions show our techniques yield up to 60% memory energy savings with less than 1% performance loss.
- We develop an analytic model of a feedback control system rooted in formal control theory to dynamically scale online memory capacity while minimizing performance loss. We detail each of the control system components, formulate the system transfer function,

and then describe the derivation of control gains through stability analysis. We develop an implementation of our control system model, which we integrate with our prototype operating system into a complete runtime system we call Memory Management Infrastructure for System Energy Reduction, or Memory MISER. We then compare the energy and performance using our control system relative to several alternative policies. Experiments on an 8-node cluster of servers show our dynamic control system conserves memory energy up to 56.8% with no performance degradation for scientific codes that utilize the entire cluster. For multi-user workloads, we achieve memory energy savings of up to 67.94% with no performance degradation. Normalizing to total system energy consumption, our power-aware memory approach reduces energy between 18.81% and 39.02%.

- We profile and analyze why the performance-driven, "more is better" interleaved memory design assumption is problematic and should be revisited. Our results indicate that for bandwidth-sensitive benchmarks such as STREAM, memory interleaving in a single dimension yields up to 35% average bandwidth improvement and reduces energy consumption by 13%. However, this improved bandwidth results in higher memory device access frequency which results in a 25% increase in memory temperature. For the same benchmarks, further increases in interleaving dimensionality result in little to no performance or energy efficiency gains but still increase temperature nearly 25%. For other benchmarks less sensitive to memory bandwidth, we found interleaving dimensionality often does not significantly improve bandwidth or energy efficiency while temperatures increase nearly 25%. The additional heat resulting from higher component operating temperatures must be exhausted from the system chassis. This increases the need for additional cooling, elevates the cost of the system and, in the case of powered cooling, can increase chassis energy consumption. We conclude that the effects of interleaving on energy and thermals must be considered in future memory designs.

- We develop a control-theoretic runtime system to reduce the power of interleaved memory systems while preserving application performance and memory bandwidth. We propose, develop, and evaluate new device power-state transition algorithms to reduce memory power within interleaved memory systems by tracking system-wide memory demand as well as bandwidth utilization. We propose and evaluate the power and performance impact of several novel asymmetric interleaving schemes that our runtime system exploits for energy efficiency while preserving application performance. Our results show that combining our memory power-management techniques with asymmetrically interleaved memory system improves EDP by up to 58%.

1.4 Organization of this Dissertation

The organization of this dissertation is as follows. Chapter 2 presents background and discusses related work. We first introduce the concepts of power-aware memory and then discuss previously proposed software and hardware techniques to reduce memory energy consumption. This is followed by an introduction to a range of previously developed control policies and how those policies have been used within a variety of disparate control systems. Finally, we review the mechanics of memory interleaving and discuss previous approaches to adapt interleaving schemes to improve memory system performance.

Chapter 3 presents the structural changes and page allocation shaping techniques we developed and implemented in our prototype operating system to improve memory energy efficiency. We introduce three alternative shaping techniques, *reactive, proactive, and hybrid*, and evaluate the performance and energy efficiency of each using a simple heuristic control policy on a server system.

Since our simple history-based heuristic policy missed many opportunities to transition memory devices into lower power states and required workload-specific tuning [151], we investigated alternative control policies. Chapter 4 introduces an analytic model of a feedback control system rooted in classical control theory that we developed to improve memory energy efficiency. We formally describe the elements of our control system, show the derivation of control gains through stability analysis, and describe the implementation we used in concert with our prototype operating system. We describe the deployment and evaluation of our complete system on a single server as well as an 8-node cluster of servers.

Even though interleaving is a widely accepted technique for increasing peak theoretical bandwidth in servers, the energy and thermal efficiency of interleaving has not been characterized in previous work. Chapter 5 quantifies the power, performance, and thermal effects of varying dimensionality within complex, highly-interleaved memory systems. This chapter illustrates that simply increasing interleaving does not automatically lead to performance improvements for many applications and in some memory configurations, may actually degrade power and thermal efficiency.

Chapter 6 describes the control-theoretic runtime system we developed to reduce the energy consumption of interleaved memory systems. This chapter shows how our techniques improve the energy efficiency of traditional symmetrically-interleaved memory configurations. We also show that using our control system on unorthodox, asymmetrically interleaved memory systems further improves energy efficiency over standard symmetric configurations while preserving application performance and memory bandwidth.

Finally, this dissertation concludes with Chapter 7, in which we summarize conclusions and discuss directions for future work.

Chapter 2

Background and Literature Survey

2.1 Power-Aware Memory

A collection of memory devices (e.g. DIMMs) in a system is often referred to as main memory. The collective capacity of all devices refers to the capacity of main memory. Typical high-end servers support many DIMMs of gigabyte capacities. Since DRAM densities continue to double every 18 to 24 months, future high-end systems in the next decade may support petabytes of memory capacity.

In the absence of proprietary hardware, conventional memory DIMMs cannot usually be dynamically onlined or offlined at runtime [94]. There have been Memory technologies such as RDRAM [1] offer multiple power modes at various granularities yet have gained limited market share in the server community and additionally lack software support. Emerging technologies [72] integrated in the Intel, AMD, and Sun server roadmaps such as the Fully-Buffered memory architecture (FB-DIMMs) combine the benefits of commodity DDR DRAM with the power mode transitions and online and offline device control of RDRAM.

The techniques proposed in this thesis do not rely on a specific memory technology. Rather, the systems and methodologies we develop are intentionally general and may be used on any memory technology or architecture that supports multiple power states.

2.2 Techniques to Reduce Memory Power

There have been numerous approaches to reduce the energy consumption of memory systems, which we classify into two categories. One approach is to leverage the malleability and workload insight of system software to change how memory devices are accessed to improve energy efficiency. An alternative approach is to modify the underlying memory architecture to dynamically exploit intermediate power states, alleviating the need to modify system software or applications. This section reviews previous work in both of these areas and discusses limitations that motivate our approach.

2.2.1 Software Approaches

There has been considerable recent research to improve the energy efficiency of systems using the operating system. The improvements demonstrated in previous work have included: efficiently adapting processor power modes to meet application computational requirements using minimal energy [14, 16, 17, 25, 26, 35, 59, 63, 65, 98, 107, 115, 147, 149, 156, 158, 164, 165], adapting the power states of network interface adapters based on network traffic [120, 121], tuning memory management algorithms to take advantage of power managed memory devices [44, 88, 113], optimizing storage-bound I/O accesses to minimize hard drive accesses [50, 86, 121, 172], and systemic approaches to elevate power-management to a first-class operational constraint [52, 135, 167, 168].

Historically, processors have consumed the most total system power of the components within a system. So, there has been significant focus on reducing the energy consumption of processors. However as previously discussed, over the past 5 years advances in process technology coupled with the widespread trend towards multi-core processors has resulted in lower per-processor power consumption [20]. Meanwhile, system memory density and capacity has continued to increase to meet the performance needs of increasingly rich data-centric applications [11, 24]. These trends have driven recent memory management research to improve the operational efficiency of memory systems.

Memory management has been an active area of research for decades. Historically, the primary goal has been to efficiently utilize limited memory resources to improve application performance. As memory power consumption has scaled proportionally with increases in capacity, energy consumption has emerged as an important factor in efficient memory management over the past ten years. The remainder of this section highlights recent software-based approaches to improve the energy efficiency of emergent, high-capacity memory systems.

In *Power-Aware Page Allocation*, Lebeck et al studied the impact of static and dynamic memory controller policies coupled with page allocation algorithms for automatically transitioning memory into lower power states to reduce power consumption [113, 155]. Lebeck proposed and evaluated several virtual memory page allocation policies to identify the impact of different page allocation algorithms on various hardware power-state policies. They showed that using a sequential first-touch policy, their techniques effectively aggregated pages onto a minimal set of memory devices as they were accessed. This enabled unused devices to transition into lower power-states according to the embedded power-state transition policy. They also studied an allocation policy that clustered frequently accessed pages onto a minimal set of memory devices over time. The idea was to use software placement of pages into frames to increase the probability that embedded power-state policies would transition devices into deeper power-states, thereby reducing wasted energy. Interestingly, the best reported results were achieved using a combination of software and hardware techniques in concert, as the authors found that hardware policies alone were insufficient to significantly improve energy efficiency as measured by Energy-Delay Product (EDP) [69]. Further, their results showed that using power-aware page allocation with the simplest static power-state transition policies they were able to improve energy efficiency up to 30%, as measured by EDP.

Rather than focusing on allocation techniques to reduce memory power consumption, an alternative approach proposed by Delaluz, et al, in *Scheduler-Based DRAM Energy Management*, directed the transition of memory device power states at context-switch granularity within the operating system scheduler [45]. They leveraged the OS-maintained knowledge of which page frames were mapped to each process to control the power-state transitions of memory banks. During a context switch, memory banks that were not referenced by the process about to be scheduled would be turned off to save power. Conversely, the memory banks for the next scheduled process would be turned on at context switch time to avoid latency-induced performance penalties. Using these scheduling techniques, Delaluz et al realized up to 40% energy savings in the memory system for several applications. In a subsequent study, Delaluz proposed a runtime library to conserve energy in a single application environment by collocating or migrating frequently accessed data structures to the same bank within a multi-bank memory system [42].

In *Dynamic Tracking of Page Miss Ratio Curve for Memory Management*, Zhou et al proposed a metric for analyzing the page demand of applications within an operating system that could be used to direct memory system energy management [171]. They developed a system to track page accesses within the OS, which they then used to determine how many pages to allocate to processes. Even though tracking at the granularity of page hits and misses incurred 7% - 10% overhead, they were able to identify the minimal set of pages necessary to meet application demand based on access frequency. Consequently, unneeded memory devices were turned off to conserve energy. Based on an evaluation of these techniques in simulation, Zhou realized EDP improvements of 27% to 58% over hardware-only power-management policies.

One of the innovations borne out of classical memory management research was virtual memory [29, 30, 46, 47, 99]. Originally conceived for use on the Atlas machine in the 1950s, virtual memory is still widely supported by most modern microprocessors [39, 46, 99]. Virtual memory abstracts system memory details by providing a large, consistent address space. These virtual address spaces provide access protections and enable the data used by executing processes to reside across memory and secondary storage. Despite the advantages, virtual memory is not free. An access to a virtual address must first be translated to a physical address, typically through a lookup table. Many modern processors have a memory management unit (MMU) that performs the translation of virtual to physical addresses via a finite-state machine. Advanced memory controllers then must translate the physical address to a device address within the memory system. Depending on the topology of the memory system, this translation can require several levels of decoding which eventually map to Dynamic Random Access Memory (DRAM) cells [100]. Consequently, accessing memory within a virtual address space often requires several levels of translation before performing the hardware access. Second, supporting separate virtual address spaces for every process imposes a spatial overhead to store the translation tables referenced by the MMU. Third, context switching between processes requires the processor to flush the previous process's page tables and translation look-aside buffer (TLB), which for some architectures, is an expensive operation [100, 133].

Leveraging classical elements of virtual memory, Huang et al proposed an OS-centric approach that combined elements of Lebeck's allocation scheme [113] and Delaluz's DRAM scheduling techniques [45] in *Design and Implementation of Power-Aware Virtual Memory* [88]. They designed and built a prototype power-aware virtual memory implementation that leveraged OS-level NUMA memory management infrastructure to reduce the energy footprint on a per-process basis [88]. Using Linux support designed for NUMA architectures [70], Huang et al mapped the logical NUMA node abstraction traditionally used to articulate distance-based access latencies for memory regions [5, 19], to track allocations mapped to specific RDRAM memory devices [1]. For every process, they tracked which memory devices (or nodes) held page frames allocated by that process. Similar to Delaluz [45], when a process was about to be scheduled, the memory devices that held page frames allocated to the process were brought into a high-power state to minimize access latency. Memory devices that did not hold allocated page frames for the scheduled process were transitioned into lower-power states to conserve energy. As an improvement over prior work, periodic page migration was used to further reduce the number of memory devices required by aggregating read-only shared libraries and reducing the number of devices required for individual processes. Using this approach, Huang et al realized 34% to 89% energy savings on a 16-device RDRAM memory system.

There are several common themes amongst all of these software approaches. First, the inclusion of power-aware software techniques can have a dramatic impact on memory energy consumption. Lebeck, and in a subsequent retrospective position paper by Vahdat, concluded that even with static hardware policies, software-approaches that change how memory is used can improve energy efficiency [14, 113, 155]. Second, the goal of all previous work was to maximize the time devices spend in low power states. The primary differences between the approaches were the prediction and control mechanisms used to manage transitions between power states. Third, it has been shown that operating systems have a direct impact on memory system performance [36]; So even though complex mechanisms such as page migration may be used to improve energy efficiency, these techniques must be used carefully to avoid application performance penalties.

2.2.2 Hardware Approaches

As memory system capacity continues to scale, the increased memory power consumption has motivated the incorporation of power-aware techniques not just in software, but in hardware architectures. In contrast to software techniques, power-aware hardware approaches have the advantage of low-latency access to high-fidelity access frequency information and fine-grained control capabilities. Additionally, power-aware mechanisms (e.g. those that realize energy efficiencies) implemented in hardware can lead to efficiency improvements that do not require costly changes to complex system software [127]. What these approaches generally lack is higher-level context and insight that limits their efficiency. For example, the mapping of accesses to applications could be used to predict future accesses and utilization. Despite this challenge, there has been considerable previous work within the architecture community to improve the energy efficiency of memory systems. Similar to processor-based DVFS scheduling, most of these approaches exploit memory devices that support intermediate power states.

Lebeck et al compared the use of static memory controller thresholds and dynamic, predictive thresholds to determine when to transition memory devices into one of four power states (active, standby, nap, and power down) under several memory management algorithms [113]. In subsequent work, they developed an analytical model to predict the duration of idle periods, but found such prediction did not outperform simply transitioning devices to a lower power state immediately after detecting idleness [58].

Li et al proposed several performance-guaranteeing control algorithms to bound performance losses caused by fluctuations in workload memory accesses and improve application performance over the memory power-state control heuristics proposed by Lebeck [113, 117, 118]. They proposed two sets of control algorithms, static and dynamic, which they applied to power-aware memory and disk storage systems. Their static algorithm avoided manual threshold identification and tuning by periodically adjusting thresholds online while guaranteeing performance by bounding degradation within a limit. Motivated by formal optimization techniques, their dynamic control heuristic alleviated the epoch-based thresholds of their static scheme, instead specifying a single device power state configuration across an epoch. They were able to eliminate an 800% performance degradation they observed when reproducing Lebeck's threshold-based techniques. In terms of energy, their static policy outperformed Lebeck's techniques for most workloads (although, in several cases their static policy consumed 36% to 46% more energy) and their dynamic policy resulted in up to 68% energy savings. Park et al later extended the performance-guaranteeing control algorithms from the realm of sequential applications and applied them to memory energy management under multithreaded applications on multiprocessor systems [132].

Pandey et al employed similar performance-directed techniques to increase memory device utilization while the high-power active power-state by increasing concurrent DMA traffic on servers with data-centric, I/O intensive workloads [131]. Their approach was motivated by the disparity between I/O and memory speeds results in long DMA transfers. This wastes energy since devices are forced to remain in an active, high-power state for the duration of the DMA transfer. They proposed two approaches. The first was a technique to direct DMA transfers originating from I/O devices on distinct buses to the same memory devices. Second, they proposed a technique to align DMA transfers based on logarithmic page popularity. Using these two approaches, they reduced memory energy by up to 38% for I/O intensive workloads.

Huang et al proposed a combined hardware-software approach that involved augmenting the memory controller with a context-aware power management unit to track memory access behavior for every process in a system [90]. Using the OS to provide process context information (e.g. which process is currently running) the memory controller was designed to track memory access history for each process. Using this information, the memory controller was able to transition previously unreferenced memory devices into lower power states to conserve energy. Huang subsequently proposed a more general technique to reshape DRAM-bound memory traffic within the memory controller to increase the duration of idle periods [87]. By lengthening idle periods, memory devices remained in lower power-states more often thereby reducing power consumption. In their simulation environment, they saved an additional 35% to 38% energy.

AbouGhazaleh et al proposed alleviating L3 processor caches and integrating smaller, high-speed SRAM caches into memory modules [4]. By changing the distribution of memory within the topology, they sought to reduce miss rates within the cache hierarchy, reduce the latency and power costs of large centralized L3 caches, and increase DRAM bank idle periods to save energy. Using this approach, AbouGhazaleh realized EDP improvements of up to 84%.

Another approach being pursued by ZettaCore [141] as described by Venkatesan et al is to replace conventional DRAM capacitors with “charge-storage” molecules capable of functioning as a capacitor [157]. Due to the unique molecular composition, accesses can be performed at very low voltages. Unlike traditional DRAM, ZettaRAM is designed to operate at multiple voltages, enabling power-savings to be traded off with performance. Using a hybrid write policy where memory writes are performed at different voltages based on performance requirements (fetches are performed at higher voltage resulting in lower-latency, while writebacks are completed more slowly, at a lower voltage), performance degradation is bound within 1% with up to 34% energy savings.

Aggarwal et al proposed a mechanism to reduce speculative broadcasted DRAM accesses on shared-memory multiprocessor (MP) systems using region coherence arrays [6]. Motivated by the fact that many DRAM accesses constitute wasted energy in MP systems due to satisfaction by cache to cache transfers, the authors developed an extension to region coherence arrays to identify unnecessary accesses within cache-coherent topologies. By reducing superfluous speculative DRAM accesses, they achieved a reduction of 16% to 21% in average EDP within the memory system.

Ghogh and Lee proposed a policy to minimize wasted energy in the memory system by reducing the number of DRAM refresh operations issued by the memory controller [68]. DRAM cells need to be periodically read and rewritten to maintain the integrity of data in the storage capacitors due to access transistor leakage [100]. However, these refresh operations consume the same power as row accesses and limit achievable bandwidth. Ghogh and Lee embedded timeout counters within the memory controller to track the last access or refresh operation for every row of each bank in a memory module.

2.2.3 Limitations of Previous Work

Even though techniques embedded in hardware have fine grained control and transaction granularity access information, the lack of system-level insight to correlate memory controller transactions to applications limits the effectiveness of these approaches. For example, a memory controller monitor can observe individual command and data transactions traversing the memory system, but there is no way for it to predict if a memory intensive application has been invoked and is about to start inundating the system with memory accesses. The lack of this insight leads to power-state transition mispredictions that impact performance. Second, many memory controller studies have focused on scheduling memory requests (or reordering of queued requests) within the memory controller. The idea is to exploit slack within memory banks, enabling more frequent transitions into lower power states. While such scheduling approaches are important, these studies did not consider the applicability or impact of the policies in the context of interleaved memory topologies. Since interleaving is commonly used to provide a significant boost in memory system bandwidth, evaluations must consider this important architectural feature.

Previously proposed software approaches have several inherent limitations. First, the system-software studies by Delaluz [45] and Huang [88] did not address the scalability of their approaches on highly parallel systems. Their systems used memory device tracking and control on a per-process basis. While they were able to hide device power-state transition latencies within context switch latencies, this is not necessarily practical on multiple socket systems with many-core processors that are all executing multiple applications in parallel. For this class of system, a runtime system that does not rely on per-process access characteristics is needed for scalability. Second, in Huang's system, page migration was used to reduce the number of memory devices required to remain in a high-power state. This incurs overhead that could be avoided by proactively allocating pages based on system-level memory demand. Finally, the prediction mechanisms used were heuristic based and would require costly retuning for different workloads. While these techniques were shown to work well in their evaluation environment, it is unclear how these might scale to more advanced memory topologies running complex server workloads.

2.3 Power-State Control Systems

The introduction of configurable dynamic power-states within hardware components enables the overall reduction of power consumed by computing systems. In most cases, performance is proportional to power consumption, so reducing power consumption reduces performance. In mobile systems users may opt for lower performance to realize extended battery life. However, in commercial servers where throughput and latency are critical, performance degradations may cause a company to lose money, customers, or both. Similarly, in high-performance clusters used in scientific research, reducing power naïvely at the expense of performance may significantly delay the time-to-completion of large-scale applications, resulting in reduced energy efficiency over the period of execution.

Reducing runtime power consumption, while maintaining application performance, requires efficient and precise control systems to adeptly transition between multiple component power-states. Such control systems are often referred to as control policies [14]. The general goal of control policies is to accurately maximize the effectiveness of the system under control for specific objectives with minimal overhead [7, 9, 109, 116]. Numerous control policies have been proposed to manage the power-states of various hardware components including processors, memory, network interconnects, I/O devices, as well as combinations of such components [3, 4, 12, 14-18, 21, 25, 26, 32-35, 38, 42-45, 48, 49, 52-54, 57-61, 63, 67, 68, 74, 80, 82, 83, 87-93, 96-98, 105, 107, 114].

Among these approaches, the control policies used in previous studies can be generally classified as one of two types: heuristic-based and control-theoretic. Each of these two classes of control policies may be composed of different underlying constructs that trade-off operational overhead, effectiveness, predictability, stability, and consequent to these, complexity. This section discusses and contrasts previous work in applying different incantations of these two classes of control policies to conserve energy on systems with power-aware components, with a focus on memory systems.

2.3.1 Heuristic-Based Policies

Heuristic-based control policies are commonly used in control systems because they are intuitive, flexible, and hence simple to formulate and implement quickly. Given a limited range of parameters, such policies are also usually easy to adapt or periodically retune to account for new conditions or environments. Because heuristic-based policies are typically designed specifically for the system under control, they often yield good performance and control results under a bounded range of conditions. As a result, heuristic-based policies are commonly used in control systems across many engineering domains [9, 116].

There are also numerous drawbacks of using heuristic-based control policies. The most common problem is the need to retune control parameters for variations in environmental conditions [9, 116]. In many cases, changing the parameters of the system under control requires an extensive re-evaluation of new control parameters, such as timeout or threshold values, to achieve reasonable performance. For example, while evaluating system shutdown methods, Hwang and Wu observed variance in energy savings using different threshold values due to differences in the actual shutdown overhead [93]. Li et al [117] found substantial performance variance using the memory system control heuristics proposed by Lebeck et al [113] by simply executing a different set of applications. Achieving energy and performance parity required extensive parameter retuning and eventually led to the design of a new heuristic [117]. Further, the improved heuristic control policy was subsequently retuned yet again to account for parallel execution [132].

The underlying problem demonstrated by this evolution is that even slight changes in the parameter space of the controlled environment (such as executing different applications) can cause undesirable consequences, such as performance degradations. Because most heuristic-based policies lack mathematical rigor, the performance and stability effects are difficult to identify or quantify without extensive empirical verification and validation in different environments [7, 9, 14, 116, 128]. Still, due to the wide-spread use of heuristic-based control policies to improve the energy efficiency of computing systems in the literature, we review the use of both static and dynamic policies in the following sections.

2.3.1.1 Static Heuristic Policies

Heuristic-based polices are generally classified as one of two types, static or dynamic. Static policies are usually simple and fixed over some interval. A common example of a static heuristic-based policy used in power management is the timeout policy [14-16, 18, 25, 48, 63, 64, 105, 114, 117, 131]. The timeout policy is used to transition components into lower power states after a predefined period of inactivity. The benefit of static policies lies in the limited information necessary for control. Timeouts typically use a single input, such as time since last access, as the sole control point, ignoring other factors. This simplicity minimizes overhead yielding good operational control system performance; however, this lack of predictive sophistication limits the flexibility of the policy to adapt to future changes.

In the case of power management-oriented control systems, timeout policies save energy at the expense of performance. More specifically, static policies do not predict future activities; they are constructed to blindly perform some action upon meeting a predefined condition. As a result, substantial performance degradations may be incurred when workload characteristics change rapidly. In a power-aware system, a timeout control policy may be configured to automatically transition a hardware component into a lower-power, high-latency state after a certain time period [44, 58]. Should this transition be immediately followed by a period of high activity, a performance penalty proportional to the time necessary to return the component to the higher power state necessary to service the activity will be incurred. Depending on the power-state transition latency for a given hardware device, this may severely degrade performance.

Numerous studies have employed static, heuristic-based policies to conserve energy. Lebeck used several variants of a simple access-based timeout control policy to control power-state transitions for a Power-Aware DRAM model of memory devices [113]. The use of static policies was motivated by the need for low-overhead and simplicity given the proposed implementation in hardware. In their evaluation, they realized significant power savings at the expense of increased delays in application time-to-completion. Fan et al showed that using a static policy of immediately transitioning memory devices into a lower power state were more effective on cache-based systems than using sophisticated prediction [58]. Similarly, Pisharath et al used a static policy to transition memory banks into the lower-power, standby mode after every memory access in non-cached systems [134]. In their evaluation using memory-intensive TPC-H database queries, memory bank energy consumption was reduced by 55% at the cost of performance degrading by 28%. Ghosh realized up to 25% energy savings for DRAM systems using a timeout counter for each bank-row pair of memory devices, avoiding sending unnecessary refresh operations to DRAM that had been recently accessed [68].

2.3.1.2 Dynamic Heuristic Policies

Although static heuristic-based policies are conducive to hardware implementation due to low-overhead and simplicity, the potentially severe performance impact motivates the use of dynamic heuristic-based control policies. One of the differences between static and dynamic heuristic-based control policies is prediction. Dynamic heuristics often incorporate simple prediction mechanisms to estimate when to exert control signals. Adding predictive capabilities improves control response and performance at the expense of additional spatial and execution-time overhead.

Many predictors used to control hardware power-states are history-based [26, 44, 58, 85, 93, 98, 123, 139, 148]. The basic idea is that past activity history provides insight into near-future activity. For example, off-cache memory accesses may occur at some detectable frequency throughout execution on workloads with strong memory access locality. In such cases, a history-based predictor could detect the trend of the time between memory accesses and determine whether a sufficient period of inactivity is likely to occur that would benefit from a power-state transition. In contrast to simple timeout policies, a history-based predictor would avoid the performance penalty of one long period of inactivity occurring within a sequence of rapid accesses. In this case, the timeout policy would cause a performance-degrading power-state transition, whereas the history-based predictor would identify the periodic of inactivity as an outlier within the sequence and avoid the transition. Thus, dynamic history-based prediction improves energy efficiency while mitigating performance penalties relative to static policies.

A common problem with history-based predictors is mispredictions. These occur when previous history does not correlate well to near future activity, such as workload fluctuations [26, 44, 58, 85, 93, 98, 123, 139, 148]. Thus, dynamic history-based policies are often augmented with additional logic to mitigate the impact of mispredictions. Many dynamic policies use thresholds to exert control decisions, where control signals are driven if some parameter exceeds a given threshold. The threshold used in such control systems is critical to performance. While fixed thresholds are generally easy to determine for a particular use case, they often perform poorly when the environment changes. For example, Hwang and Wu found different shutdown threshold values impacted energy savings differently depending on the shutdown overhead [93]. To achieve good performance, they had to choose thresholds based on offline analysis of application traces.

To overcome this inflexibility, several schemes have been developed to improve adaptability. Hembold et al proposed a policy that kept a list of thresholds and assigned weights to each depending on the observed outcome using each threshold, which they derived through offline analysis [75]. Another approach used by Douglis et al, incremented or decremented a threshold based on how it was performing [50]. Within their threshold-based throttling control system, Felter et al proposed the adaptation of throttling thresholds for systems with insufficient decoupling capacitance to tolerate dI/dT fluctuations [60].

For systems in which runtime parameters do not change, in contrast to workload fluctuations on power-aware systems, the simplicity of static heuristic-based policies motivates their widespread utilization in practice. The straightforward improvements offered by dynamic heuristic-based control policies motivate their use when the spatial and computational overheads are tolerable. However, such polices still incur substantial verification and validation costs when environmental parameters change. Since heuristic-based policies lack the mathematical foundation and rigor that yields analytical insight into control system performance and stability during runtime fluctuations. As a result, recent power-aware computing research has turned to formal control systems rooted in classical control theory.

2.3.2 Control-Theoretic Policies

Control theory provides a standard tool set for analytic formulation of robust and flexible control systems with discernable stability and performance characteristics [3, 7, 9, 116, 128]. Techniques based on control theory are widely used across many disciplines ranging from aeronautical engineering to economics, but also in various applications in computing. The widespread use of control theory is motivated by the capability to analytically reason and evaluate control system behavior in terms of performance and stability. As such, when system parameters change, the impact on control response can be identified and quantified without having to resort to searching large parameters spaces for better heuristic values or expensive revalidation.

Recently, it has been suggested that standard controllers, such as the PID, from control systems theory should be applied to control various aspects of computing systems [108]. Karamananolis argues that standard off-the-shelf controllers from control theory should be used instead of ad-hoc heuristics to control configurable system components, thus enabling system researchers and engineers to focus on designing robust, configurable computing systems.

Although the intuitive appeal of heuristics often motivate their use in software and even hardware control systems, formal feedback-driven control systems offer stability, predictability, efficiency, and in many cases implementation simplicity. Consequently, there have been many recent efforts to incorporate models from control theory instead of resorting to the development of custom heuristics. The remainder of this section introduces a common controller that is used extensively in related work and then discusses approaches that have employed control-theoretic systems to manage system performance, power management, and even network traffic.

2.3.2.1 Proportional-Integral-Derivative Controller

Within the spectrum of dynamical control systems, feedback control systems adaptively track a changing variable by altering its response based on the error or difference between the set point and a process variable. A feedback control system transfer function can be expressed generically by the following equation:

$$D(z) = \frac{M(z)P(z)}{1 + M(z)P(z)} \tag{2.1}$$

The numerator reflects the feed-forward transformation of the input signal through the controller and plant transfer functions to the output signal, while the denominator relates the effect of the feedback loop to the output signal. The term *M(z)* describes the system controller, while the term *P(z)* constitutes the plant function. The controller regulates changes within the system by sending a control signal to the plant function, which is responsible for control actuation through domain-specific mechanisms.

A common controller used in dynamical feedback systems is the Proportional-Integral-Derivative, or PID controller [3, 8, 9, 77, 112, 143]. Historically, the flexibility and simplicity of PID controllers have motivated their use in controlling industrial systems, particularly mechanical systems [9]. A PID controller is generically described by the following equation in the continuous time domain:

$$m(t) = K_P e(t) + K_I \int e(t)dt + K_D \frac{de(t)}{dt} \qquad (2.2)$$

m(t) is the output of the controller at time *t* based on the measured error within the system. As shown in equation 2, the output of the PID controller is the algebraic superposition of three terms:

- $K_P\, e(t)$: The first term is proportional to the error. This causes the system to respond to the error value and direction.
- $K_I \int e(t)dt$: The second term is proportional to the integral of the error. This aggregates error over an interval to eliminate the steady-state error.
- $K_D\, e(t)d/dt$: The third term is proportional to the rate of change in the error, which is used to reduce overshoot by directionally damping the response.

The terms K_P, K_I, and K_D are control gains that are analytically determined through stability analysis [8, 9, 116]. Poor selection of these parameters can cause system instability manifested as output signal oscillation – which for performance-constrained systems can lead to catastrophic performance losses. Careful selection of control gains results in a controller with desirable convergent, performance properties.

2.3.2.2 Network Power & Performance Control Systems

Abdelzaher et al used a control-theoretic feedback control model to manage relative delay guarantees between different quality of service levels on web servers [3]. They also proposed the use of middleware, which they called *ControlWare*, to express various quality-of-service guarantees as feedback control problems.

Hollot et al used formal feedback control design methodologies to analyze parameter identification for a model of TCP traffic active queue management control systems (using the random early detection scheme) in terms of performance and stability [79]. Through this analysis, they found they increased stability in terms of response behavior at the cost of performance; moreover, they found several limitations of the random early detection scheme. One such limitation manifested as a performance penalty resultant to the loading of queue levels that caused decreased bandwidth per flow. This analysis prompted the investigation of other active queue management control systems [78].

2.3.2.3 Micro-architectural Control Systems

Joseph et al proposed a micro-architectural mechanism to prevent wide variations in supply voltage supply consequent to changes in current-draw (the *dI/dt* problem) during invocation of power saving modes using a second order linear control model [103].

Motivated by the need to alleviate hot-spots on modern high-performance microprocessors [144], Skadron et al proposed the use of a PID controller to adaptively actuate instruction fetch regulation [143]. By throttling the rate at which instructions are fetched from memory, localized elevated temperatures within on-die functional units were avoided or in the case of emergencies, mitigated. Using a formal feedback control system instead of the ad-hoc heuristics that had been previously proposed [22], they reduced the performance impact of thermal management by 65%.

Lin et al also employed a PID controller to improve the efficiency of dynamically managing thermal emergencies in memory systems [119]. This work sought to alleviate the performance degradations of thermal emergency induced DRAM shutdowns. Using a PID controller, they were able to efficiently regulate memory temperature within an acceptable bound by actuating DVFS power-performance state transitions at the CPU to reduce the traffic at the memory controller. Using simple heuristics, they found very conservative thresholds were required to avoid thermally-induced memory shutdowns, causing premature throttling. By using a PID controller, they were able to regulate memory temperature near the silicon limit, maximizing memory traffic and hence performance.

2.3.2.4 Processor Power-Performance Control

Varma et al developed an online DVFS algorithm based on a variation of the PID controller to reduce power consumption by exploiting ACPI performance states [2] while minimizing performance loss [156]. They achieved up to 75% reduction in power using an implementation of their controller in an operating system, which constituted an improvement of 10% to 50% relative to a heuristic-based DVFS control policy.

Lefurgy et al employed a firmware implementation of a proportional or P-controller to track and adjust processor performance state transitions to control peak power consumption on IBM blade-servers [115]. Since their goal was to control peak power, they evaluated their P-controller relative to an enhanced open-loop controller and an ad-hoc controller, which are claimed to be commonly used to regulate power consumption in industry. They found that application performance improved by 31% to 82% over using the open-loop controller and 17% over the ad-hoc controller. Although the performance of the ad-hoc controller was close in several cases, performance using the P-controller was consistently higher due to the minimal settling time.

Wu et al employed a Proportional-Integral (PI) controller to control DVFS transitions within separate clock domains of a multiple clock domain processor [164, 165]. Modeled as a queue domain network, their online controller tracked and adapted the local domain frequency to service performance demand, using interface queue occupancy as feedback. Relative to heuristic-based controllers, their control-theoretic DVFS controller guaranteed stability and achieved 146% improvement in EDP.

Wang et al modeled the coordination of component-specific, power-performance control systems as an optimal control problem [162, 163]. They constructed a power-performance management framework to adaptively control quality of service goals, as measured by web server response times, across a distributed four server cluster. Despite server failures, Wang realized up to a 55% reduction in power consumption while maintaining consistent response times. Similarly, Rachavendra et al proposed a power management control architecture rooted in control theory to coordinate multiple nested, domain-specific, power-aware feedback control systems specifically to reconcile competing domain policy objectives with system-level objectives [138].

2.3.2.5 Limitations of Previous Work

Although elements of formal feedback control theory have been proposed in many power-management studies, previous memory power management research has used various forms of heuristics to control device state transitions. Because heuristics often require extensive retuning due to minor system changes, this limits the applicability of these approaches for controlling power-state transitions in future memory systems. For example, are the scheduling-based control heuristics proposed by Delaluz [45] and Huang [88] applicable in multi-core or many-core platforms designed to support the execution of thousands of threads with a memory system composed of potentially hundreds of DRAM banks? A definitive answer would require an extensive study since it is difficult to quantitatively reason about the stability and performance of those control systems.

Much of the work discussed in this section demonstrates the benefits of using standard controllers, which have been applied extensively in other engineering disciplines, to adaptively manage power as a function of load. As suggested by Karamanolis, using off-the-shelf controllers alleviates the need to design custom adaptive controllers and permits the systems community to focus on designing and developing configurable systems that benefit from such control [108]. As shown by the previous work described in this section, researchers have successfully incorporated feedback control systems form formal control theory to achieve efficiencies in dynamic hardware management. However, the application of control theoretic systems to realize efficiencies within the memory system remains largely unexplored.

2.4 Memory System Architecture

Despite the rapid advances in processor speed, memory access latencies have not scaled proportionally [102, 133]. In the mid nineties, Wulf and McKee described the system performance limitations caused by this difference as the *memory wall* [166]. There has been extensive hardware architectural research to mitigate the effects of the memory wall. The most common approach has been to integrate additional caches into the memory hierarchy [104, 133, 145], although many additional techniques have also been proposed. Some of these include: embedding the processor in memory [111, 160, 169], enabling architecture-specific application optimizations [31, 104, 122, 125, 159, 161, 169, 174], prefetching data from the memory system into caches [6, 27, 28, 37, 41, 66, 130, 145, 154], and interleaving accesses amongst multiple memory banks [13, 40, 41, 84, 100, 129, 170].

Of these approaches, the increased capacity of modern memory systems and need for low-latency and high-bandwidth has resulted in the widespread deployment of memory interleaving in current systems, particularly in servers. Given this trend, this section reviews the taxonomy and mechanisms of interleaving followed by previou approaches to improve system memory performance using various memory interleaving schemes.

2.4.1 Memory Interleaving

Memory interleaving refers to the mapping of the contiguous physical address space across multiple memory devices within a system. When a cache miss occurs at the last level within the CPU cache hierarchy, a memory request for the address is sent to the memory controller. As shown in figure 2.1, the memory controller translates the physical address into memory controller, channel, rank, bank, and eventually row and column offsets to determine which memory devices should be activated. The memory request is subsequently queued, scheduled, fetched, and returned to the processor cache in the case of a read or written to the corresponding DRAM cells in the case of a write. The physical address space may be interleaved at different granularities within a memory topology depending on the system board and memory device configuration. We refer to each of these granularities as interleaving dimensions. This section reviews the interleaving dimensions commonly used in DRAM-based memory systems.

2.4.1.1 Bank Interleaving

DRAM forms the basic building block of current memory technologies. At the most primitive level, DRAM consists of a collection of cells, each consisting of a transistor-capacitor pair. Cells are aggregated into two dimensional DRAM arrays, and are accessed using row and column indices. As part of an access, a row access strobe (RAS) signal is driven by the DRAM controller. This causes an entire row (or page in DRAM parlance) of the DRAM array to be brought into the sense amplifiers. The column access strobe (CAS) signal is then decoded, which determines which columns within the sense amplifiers should be driven onto the data bus. Sets of DRAM banks are aggregated to form a DRAM device. Since banks within a DRAM device operate independently, multiple banks may be accessed in parallel. Thus, bank interleaving maps addresses across banks to maximize parallelism within DRAM devices and minimize latency of row buffer conflicts.

2.4.1.2 Rank Interleaving

A set of DRAM devices connected in parallel and operating in lockstep comprise a rank. For each rank, the collective width of DRAM bank rows (e.g. sense amplifiers) constitutes the row size from the perspective of the DRAM controller. Thus when the memory controller issues a command to access a specific row within a rank, the same rows within constituent DRAM devices are accessed in parallel.

Rank interleaving maps contiguous cache lines across multiple ranks to reduce row buffer conflicts and minimize the impact of turnaround latencies. This enables rank-level accesses to be parallelized as long as the ranks are not on the same device. A rank-sequential configuration maps cache lines to ranks up to the capacity of available ranks serially. When ranks are interleaved, multiple cache lines may be accessed concurrently.

2.4.1.3 Channel Interleaving

The bus connecting memory controllers with memory devices is referred to as a memory channel. Most channels consist of a multi-drop bus which connects devices that support specific DRAM technologies, such as DDR, DDR2, and DDR3 DRAMs. The capacities of multi-drop bus channels are limited by the number of impedance discontinuities as parallel memory bus speeds increase. Thus, increasing overall system memory capacity requires the costly addition of more memory channels. Other architectures such as fully-buffered DIMMs (FBDIMMs) and RamBus alleviate the multi-drop bus design in favor of narrow, high-speed channels, which allow for greater per-channel capacity. Even using serial, high speed channels, many board designs include multiple channels to mitigate latencies caused by increased distances between the controller and DRAMs.

Channel interleaving exploits the parallelism of multiple channels by mapping contiguous cache lines across the channels. Similar to bank and rank interleaving, this reduces row buffer conflicts and increases parallelism. Coupled with bank and rank interleaving, channel interleaving increases the number of outstanding transactions that may occur in parallel within the memory system.

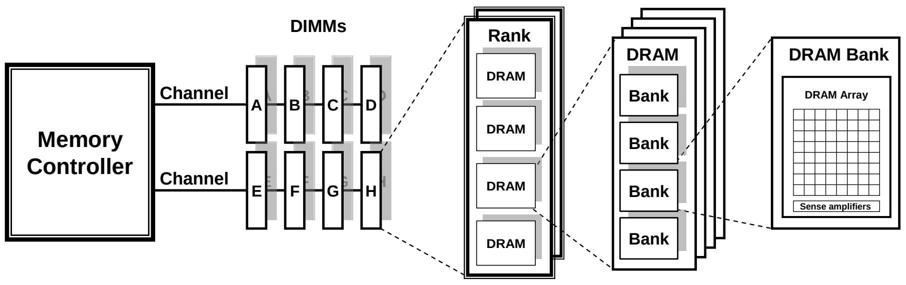


Figure 2.1. Topological view of memory system. Interleaving exploits parallelism at multiple granularities, including DRAM bank, rank, channel, and branch/controller.

2.4.1.4 Controller/Branch Interleaving

The need for increased memory bandwidth is driving the incorporation of multiple memory controllers in server designs. Including multiple controllers increases the available parallelism within the memory system as controllers may operate independently. So, transactions targeting memory residing behind each controller may be accessed in parallel further reducing latency.
One such design is the memory controller hub (MCH) within the Intel 5000 series chipset [137]. The MCH incorporates common request queuing and transaction handling logic that may be scheduled across two controllers, which are referred to as branches. The physical address space is distributed across branches based on the DIMM population within the system. When configured in sequential mode, the addressable memory of one branch constitutes the lower part of the physical address range while the second branch contains the upper range. Lower-level interleaving may still be used within each branch to increase parallelism at the bank, rank, and channel levels. Branch interleaving maps contiguous cache lines across multiple branches or controllers. In a two branch configuration, contiguous cache lines are mapped to unique branches such that even cache lines are mapped to branch 0 and odd cache lines are mapped to branch 1, enabling more cache lines to be accessed in parallel.

2.4.2 Interleaving Approaches in the Literature

Numerous memory system interleaving algorithms have been proposed and evaluated in the literature. As an example of a commercial system, Hotchkiss et al describes the memory topology of an HP commercial server and workstation developed during the mid-1990s [81]. The memory topology uses hierarchical controllers, one master controller and multiple slave controllers. Each slave memory controller may be configured to interleave accesses across multiple banks using a low-order address interleaving scheme.

Zhang et al proposed the use of a permutation-based interleaving scheme to improve performance by reducing row-buffer conflicts and exploiting row-buffer locality [170]. Using their scheme, they reduced processor memory stall time for TPC-C and SPEC 95 floating point benchmarks by 21% to 68% compared to traditional cache line interleaving schemes and 16% to 50% compared to page interleaving schemes. Because they did not use a power or thermal model in their simulations, they did not report the energy or thermal impact of the performance improvements.

Kaplan described the memory interleave scheme used in the BBN TC2000 parallel computer designed and sold by BBN Advanced Computers Incorporated [106]. Unlike previous schemes that imposed platform restrictions, such as requiring power of two banks to interleave sub-block accesses across all banks, Kaplan's method supported a highly configurable interleaving scheme across any number of banks and even partial interleaving within the memory system using simple lookup tables in small SRAMs.

Hsu and Smith studied the impact of several cache line interleaving schemes to increase data locality on vector supercomputers using cached DRAM [84]. They found that interleaving schemes that increase data locality have the side effect of creating hot banks since data used in short computational loops did not have the benefit of residing in processor data caches. They also found that using cached DRAM did not yield a performance benefit when low-order address interleaving was used due to the reduction in spatial locality. In contrast, sequential block interleaving schemes improved performance in their system, at the cost of increased contention on hot banks.

Baskett and Smith developed a model based on a Markov chain to predict access contention within interleaved memory topologies in multiprocessor systems without caches [13]. They showed their model compared well with memory contention observed on synchronous memory access machines at the time, which included the GE 645 and Honeywell 6000 series.

2.4.3 Limitations of Previous Work

Memory interleaving increases performance by exploiting parallelism within the memory system. Although interleaving is a widely accepted technique for improving memory bandwidth and reducing access latencies, it is unclear how interleaving dimensionality or different interleaving schemes impact the power consumption or thermal dissipation of the memory system. In this work, we isolate several interleaving dimensions on real systems and characterize each in terms of power, performance, and thermal efficiency.

Moreover, given the complexity and limited control over interleaving dimensionality, there has been limited previous work to reduce the energy consumption of highly-interleaved memory configurations.

Chapter 3

Memory Management Algorithms for Energy Efficiency

In this chapter, we introduce several operating system page allocation and management shaping techniques to direct allocations to a minimal memory device set. We propose three shaping techniques: proactive shaping which directs pages to frames at allocation time, reactive shaping which periodically changes page to frame mappings, and hybrid shaping which combines elements of both proactive and reactive shaping techniques. We describe the mechanics of these three techniques as well as structural enhancements we made to the Linux operating system to enable memory devices to be transitioned between multiple power states. Finally, we evaluate the efficiency of these techniques by evaluating our power-aware operating system using a simple heuristic-based power-state controller.

3.1 Introduction

Scientific computing platforms are rapidly approaching petascale. Such systems consist of server systems may have thousands or tens of thousands of processors, tens or hundreds of terabytes of memory, and hundreds of petabytes of disk space [10]. The power consumption of petascale systems, therefore, will likely be tens to hundreds of megawatts, requiring specially designed facilities to house, cool, and power these systems. Power and cooling budgets may soon rival the cost of the hardware.

A key design challenge for these emergent server systems is to reduce power consumption while maintaining stringent performance constraints at reasonable cost. Processors typically account for the largest amount of power in a high-performance cluster, yet memory power is significant [17, 114]. For example, the IBM Bluegene at LLNL uses 32 terabytes of main memory that consume approximately 70 kilowatts of peak power. That's a maximum of $1200 per week for memory energy alone excluding cooling costs.

Several methods for reducing the power consumption of processors have been proposed, such as DVS, DFS, and clock gating, [16]. These techniques all rely on the concept of *slack* in processor utilization for a given workload. When processor demand decreases, during I/O or network traffic for instance, the supply voltage or frequency is decreased to conserve power. Numerous studies have shown that clever scheduling of low power modes during computationally slack periods results in reduced energy consumption with minimal performance loss.

As in processor utilization, slack in system memory demand provides opportunities for energy savings in the memory subsystem through power-mode scheduling. The key to conserving energy in memory is to offline memory devices whenever possible without impacting performance. During slack periods, if we minimize the number of online memory devices we reduce the total energy consumption of the system. However, to avoid degrading performance we must be able to adapt to increasing demand by quickly turning on additional memory. Therefore, by dynamically adapting the amount of online system memory according to workload demand, we can minimize the energy consumption of memory.

Previous mechanisms for decreasing the power consumption of the memory subsystem have been hardware-centric and focused primarily on mobile devices [58, 113]. Other approaches have extended the operating system scheduler to manage power state transitions through per-process memory reference accounting [44]. Huang et al. leveraged NUMA memory management infrastructure to reduce memory energy consumption on a per-process basis [88]. Li et al. proposed control algorithms to reduce the energy consumption without hurting performance in memory hierarchies and disks [117].

Emergent memory technology provide systems with the ability to dynamically turn-on (*onlining*) and completely turn-off (*offlining*) memory devices at runtime [72]. Unfortunately, direct application of previous power-aware memory approaches to onlining and offlining memory devices are problematic. First, a device can only be powered off if it contains no allocations. Since many operating systems do not support transparent page migration, this is not typically possible. Second, even with support for directed page migration, the performance-driven allocation policies of the OS may stripe data across devices making offlining impractical since performance penalties will be severe. Third, monitoring memory usage per process to schedule device transitions is not scalable to large-scale server deployments with tens of thousands of processes.

In this chapter, we address the problems of extending the operating system to support onlining and offlining memory devices for systems with dense memory topologies. Inspired by network traffic flow research, we propose several OS-level page allocation and management *shaping* techniques to proactively and reactively direct allocations to a minimal number of devices. We also review the structural changes necessary to enable memory to be onlined and offlined at runtime. We extend the Linux operating system to support these shaping techniques and structural enhancements. Using our kernel implementation on *real* systems, we evaluate the performance impact of our modifications against an unmodified kernel. Experiments using a simple history-based heuristic for controlling the power state transitions of memory devices, our techniques yield up to 60% energy savings in memory with less than 1% performance loss.

3.2 Structural Changes

Collectively, the set of memory devices in a system forms the usable physical address space managed by operating systems. Due to electrical constraints, memory devices (e.g. DIMMs) are usually only added or removed when a system is powered off. So from the OS perspective, the size of the physical address space or aggregate capacity of all memory devices is fixed at boot time. Accordingly, memory related data structures within the kernel have been traditionally designed to manage a fixed memory device set at runtime. To reduce power consumption, we modified these data structures to cope with transient memory devices, enabling devices to be easily onlined or offlined with minimal overhead. This section highlights these changes.

3.2.1 Traditional Page Frame Accounting

Most operating systems use a frame table to track the state of usable page frames within the physical address space. The frame table is generally organized as a contiguous linear array such as the *cmap* in BSD [136], the *Ram Tab* in Nemesis [73], the *resident page structure* in Mach [140], and the *memory map* in Linux [70]. Because memory capacity is not expected to change at runtime, a statically sized frame table is used that covers the usable physical address space [19]. This simplifies the implementation of frame state lookup logic as the page frame number can be used as an index within the frame table.

3.2.2 Mapping Page Frame Sets to Devices

To effectively manage the power states of memory devices we need to track and manage page frame allocations by device. Since devices are mapped into the physical address space and frame tables are used to track frame state, we partition the traditional frame table into sets of frame tables, one for each power-manageable, memory device. For example, in a system that has 8Gbytes of system memory with a power-managed memory granularity of 1Gbyte (i.e. memory device size), the system-level frame table would be composed of eight 1Gbyte page frame sets.

By partitioning the frame table into discrete sets, we accomplish several objectives. First, we gain the capability of tracking page utilization of each memory device capable of being power managed; that is, we can easily discern how many frames are currently allocated or free by simply scanning individual frame tables. Second, we do not waste memory on structures for memory devices that are offline. If we offline a memory device, we can easily free the memory consumed by the associated frame table. Since large frame tables can have a significant memory footprint [70], we minimize the spatial overhead of managing offline memory devices by only allocating sufficient space for online memory devices. This maximizes the memory available for applications in any given memory configuration. Third, each frame table may be dynamically sized to account for memory devices of any capacity or even multiple memory devices with interdependent power states.

3.3 Page Allocation Shaping

To minimize the energy consumption of dense memory topologies, we need to be able to transition memory devices into lower power states. However, devices that satisfy page allocations may not be transitioned into lower power states without incurring significant latencies upon subsequent accesses. Because lower power states cause higher access latencies, the mapping of pages to frames becomes critical to performance.

In this section, we first briefly discuss page frame allocation in several operating systems and identify the challenges involved in transitioning memory devices into low power states. We then propose and compare three approaches to aggregate page allocations to a minimal set of memory devices.

3.3.1 Current Allocation Policies

Most operating systems maintain several lists to track page frame state as memory demand changes. For example, BSD variants use *active, inactive, cached,* and *free* lists [136], Solaris uses *free and cache* lists [124], and Linux uses *active, inactive,* and *free* lists [70]. Page frames traverse the lists according to their state and reference frequency. Using multiple lists for currently allocated page frames allows for further delineation between allocated types and has been the focus of memory management research for decades [23, 30, 39, 73, 101, 140, 171]. As evidenced by the lists used in these operating systems, page frames are fundamentally either *allocated* or *free*; thus for the purposes of our discussion we shall refer to page frames as being in one of these two states.

Allocated page frames are those that are currently in use. These could include frames mapped into the address space of processes as a result of `malloc` allocations, frames used for I/O transfers or to hold file system data, or even those used for device drivers or kernel data structures. Once a frame is allocated it is removed from the pool of free frames and placed onto a list that tracks its state. Allocated frames are returned to the free pool once explicitly freed or remain unreferenced for some interval.

Frames are often not immediately moved to the free list based on the prediction the page will be referenced again in the future. For example, the buffer and page caches retain previously referenced pages in memory rather than flushing data and returning frames to the free lists. By retaining pages in memory, future references are satisfied quickly by simply mapping the frame into the address space of the requesting process. Such in-memory caches improve performance for workloads that read or modify pages repeatedly. However, workloads with minimal file system I/O interaction, such as computationally-intensive scientific codes, do not tax these caches. For these workloads, these caches often consume significant memory and do not yield significant performance benefits.

Controlling the allocation of all free page frames in the system is the responsibility of a *frames allocator* [73]. When a page frame allocation request arrives, the frames allocator determines which page frame shall satisfy the request. As page frames are continually allocated and freed by the frames allocator, a new allocation request may be satisfied from any valid region in the physical address space. Since the location of each allocation is based on the dynamic memory allocation characteristics of all applications executing on the system preceding the arrival of the request, two back-to-back requests may be mapped to different memory devices.

This behavior is evidenced by the binary buddy allocator used in Linux [70]. The buddy allocator maintains blocks of contiguous page frames by power-of-two size. Several lists are used to aggregate blocks of increasingly larger contiguous page frames. When an allocation request arrives, the request size determines which lists will be searched to satisfy the request. If the list with the optimal order is empty the next list of higher order is searched. Assuming the next list is not empty, a free block (e.g. set of frames) is extracted from the list and split in half.

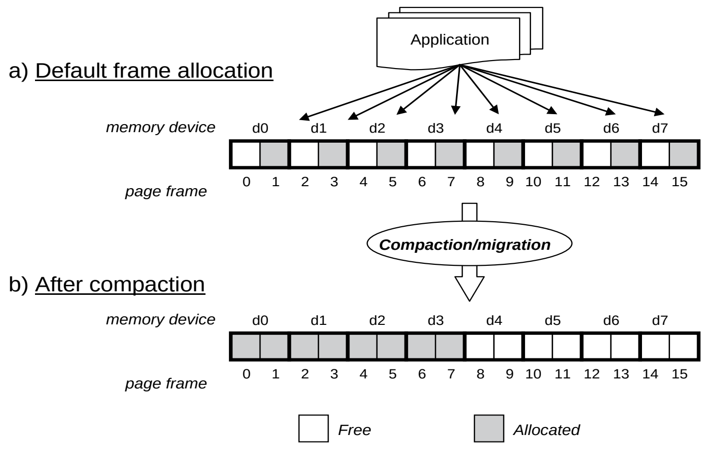


Figure 3.1. The default allocation policy often results in pages distributed throughout all devices as shown in a). After migration, pages are compacted into a minimal device set enabling devices to be transitioned into lower power states as shown in b). After migration, actual page demand (50% of capacity) is satisfied from the minimal

One half is used to satisfy the allocation request and the other half is moved to the next lower order list. As blocks are continuously allocated, partitioned, freed, and moved between lists, contiguous memory regions become fragmented. Since each list is unordered with respect to the address space, allocated frames are selected based solely on request arrival relative to previous allocation and free operations. The effect of this system is that allocated pages are scattered throughout the physical address space. In the worst-case, all memory devices must be retained in a high power state even though only the capacity of a few devices is necessary to satisfy page demand.

Figure 3.1a illustrates this scattering effect. There are 8 memory devices in this system, each containing 2 pages for a total of 16 pages. Consider an application that allocates a total of 8 pages. As a result of page faults, page frames are allocated individually at regular intervals, resulting in the total allocation of 8 frames by the frames allocator. In the pathological case the memory allocated to the application is scattered across the entire physical address space as depicted by the gray page frames. Because of the distributed allocation pattern, all memory devices must be retained online even though page demand requires only half of the system's capacity.

Given this worst case page frame allocation pattern, we observe several potential solutions: 1) we could migrate pages from frames in sparsely populated memory devices to frames in more densely populated devices. By consolidating frame allocation to a subset of memory devices, unused devices could be transitioned into low power states or even offlined. In our above example, this would reduce the energy consumption of memory by 50% and preserve the existing performance without adding complexity to the frames allocator. However, on real systems we must consider allocation requests that may not be easily migrated, such as those used by device drivers for DMA operations. 2) We could dynamically direct page frame allocation requests to specific regions of physical memory based on the intended use of the page frame. For example, if we knew a set of frames were going to be used for DMA, we could allocate frames from a memory device that we will never try to remove. Similarly, we could direct user-level, dynamically allocated application page frames to regions that are more likely to be removed. 3) We could combine the two approaches and proactively direct page frame allocation to specific regions as well as reactively migrate or swap out currently allocated pages in sparsely allocated devices. The remainder of this section discusses each of these alternatives.

3.3.2 Reactive Shaping

One approach to the allocation scattering problem is to preserve the allocation characteristics of the frames allocator, but reactively compact allocated page frames into a subset of memory devices. Figure 3.1 illustrates this approach. Recall the upper half of the figure (3.1a) shows the worst case frame allocation scheme where pages are scattered throughout devices. Figure 3.1b shows page placement after migration. Prior to migration all devices were required to remain in a high-power state; however, after migration four devices (half of system capacity) may be offlined or transitioned into a lower power state.

Migrating pages between devices is achieved through several steps. First, a new page frame is allocated. Then the page to be migrated is locked to prevent further access during migration. The page is then copied to the new frame and all references to the old frame are adjusted to point to the new frame. For example, for pages allocated by an application, the page table entry pointing to the old frame is updated to point to the new frame. The page is then unlocked and the old frame is freed.

Randomly moving pages between devices can be costly when devices contain many allocated page frames. For example, consider a system with two devices, each containing 1000 frames. If 900 frames are currently allocated on device 1 and only 50 frames are allocated in device 2, migrating pages from device 1 to device 2 would be suboptimal. To minimize migration costs the pages on device 2 should be migrated to device 1. Consequently, judicious control of page migration is required to minimize overhead.

To avoid this scenario, we scan each memory device to determine how many allocated frames each device contains. We then sort the devices to form two sets. The first set is composed of devices that contain the fewest allocated frames. We migrate pages from these devices to devices in the second set, composed of devices with the most allocated page frames.

Although theoretically, any page can be simply migrated to a different frame on other device, real-system constraints may prevent some pages from being migrated. For example, pages used for DMA operations or performance-centric regions such as those that contain kernel text may incur significant performance penalties to migrate. Considering the first example, frames allocated by a device driver for DMA operations could be freed by temporarily disabling and subsequently restarting the device. However, if the system or application depends on the device for proper operation, such as a storage controller or network interface card, performance could be severely impacted while the device is being reinitialized. Although migration may be possible for all page frames, performance and reliability constraints often limit whether a page may be pragmatically migrated.

In light of these real-world constraints, we further classify pages in terms of their potential for migration. Many operating systems maintain per-page state information that indicates how the page is currently used. We exploit this information to classify pages in terms of those that are pinned *(P)* and others that are easy to move *(E).* Pages classified as easy-to-move can almost always be moved on demand while pinned pages may never be movable. Generally, pinned pages reduce the opportunity to minimize the number of online memory devices. These classifications also affect our groupings of memory devices as devices that contain pages that may not be moved will not be targets for page migration.

After determining if the set of allocated pages residing on a memory device can be moved, we migrate sets of pages to other areas of the physical address space that map to other memory devices. In essence, we dynamically compact page utilization to a subset of the total number of devices when the number of allocated page frames is less than the total number of page frames.

Figure 3.2a shows how this approach works using the same memory device configuration as figure 3.1. In this example, the frame allocator has allocated page frames across 6 of the 8 devices. Gray frames contain allocated pages and white frames are unused. Allocated pages are further marked as *P* and *E*, for pinned and easy to move respectively. While the frames allocator

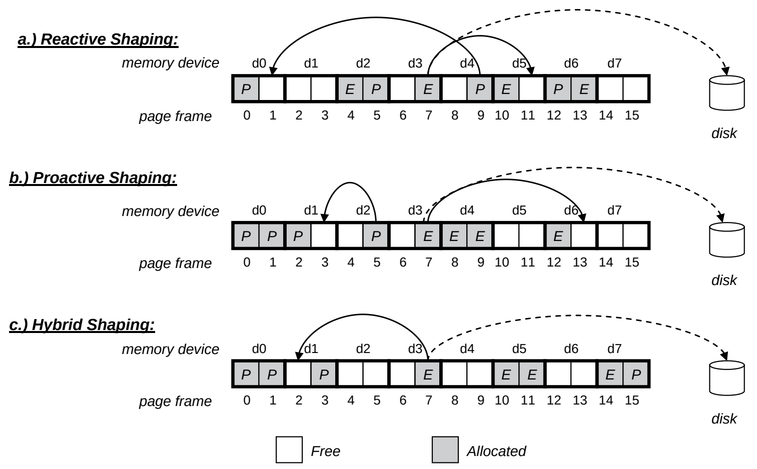


Figure 3.2. Comparison of three shaping approaches. Reactive shaping uses page migration to aggregate pages onto the minimal device set. Proactive shaping avoids migration costs by placing pages on the minimal number of devices at allocation time. Hybrid shaping combines allocation time placement with page migration to aggregate pages onto the minimal device set.

distributed page frame allocations across several memory devices, two memory devices, *d1* and *d7*, have not been used to satisfy any allocations and can be immediately transitioned into a low power state. However, since only 8 of 16 frames are currently allocated, we could optimally turn off 4 of the eight memory devices. We use migration to move the page at frame 7 (an *E* page) to frame 11 as shown by the solid-line arrow, enabling us to turn off device d3. Similarly, we migrate the page at frame 9 to frame 1 (a *P* page) also shown by the solid-line arrow enabling us to turn off device d4. Although we could have migrated our two example pages to any of the available free frames we attempt to move them to devices that have similar allocations. This increases the chance of removing the device later. However, because our approach only moves pages in a reactive manner and does not change how frames are allocated within the physical address space, collocating pages by type may be reversed at the next allocation. For example, if after migrating the page in frame 7 to frame 11, frame 6 is allocated to an *E* page, the page in frame 10 is freed and then populated with a *P* page, then collocating the *E* page in frame 7 wouldn't have been productive.

Instead of migrating pages, we could also free frames by paging pages to disk. As shown in figure 3.2a rather than migrating the page at frame 7 we could have paged it out to disk. We plan to explore that alternative in future work.

3.3.3 Proactive Shaping

An alternative approach to page migration is to proactively direct the allocation of frames from specific devices based on the characteristics of the occupying page. To direct allocations, we modify the frames allocator to manage frames in pools according to page type. As before, we differentiate between *P* and *E* pages, and loosely divide the online device set into two sets; one for *E* pages and one for *P* pages. We also add flags to the allocation call interface to distinguish between the page types. By requiring the requester to specify the page type, the frames allocator can direct the allocation to specific devices. For example, when an allocation request for a *P* page arrives, the frames allocator will allocate a frame from the *P* device set. Similarly, when a request for an *E* page arrives, a frame from the *E* device set will be selected.

An example using proactive shaping is shown in Figure 3.3. We divide the available set of frames into two sets, one for *P* pages and one for *E* pages. As in previous examples, page demand is 50% of capacity, but we determine that only two pages are classified as *P* pages, while the remaining 6 allocation requests are for *E* pages. We aggregate the allocated pages into the two frame sets when they are allocated, such that only the minimal device set is consumed by all allocation requests. As a result, half of the memory capacity in the system may be

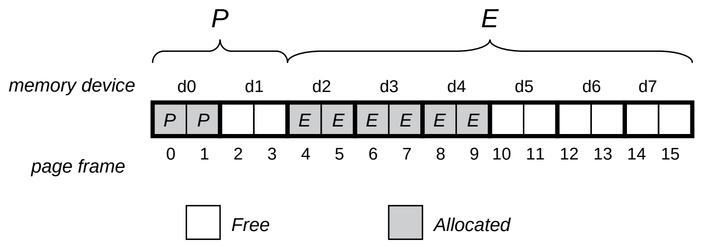


Figure 3.3. Proactive shaping aggregates pages onto a minimal number of devices at allocation time, avoiding migration overhead. Using proactive shaping, actual page demand (50% of capacity) is satisfied from a minimal set of memory devices.

transitioned into lower power states. Since we performed the delineation at allocation time, page migration is not required.

Limitations. Although proactive shaping avoids the overhead of migration, it does have limitations. For example, when unused devices are transitioned into lower power states (such as offline), the number of frames for each type of allocation is reduced. Since we segregate P pages from E pages and offlining devices creates artificial memory limitations, subsequent allocation requests for P pages may be satisfied from the E device set. This condition can lead to fragmentation similar to that originally depicted in Figure 3.1(a).

Figure 3.2(b) shows how this effect manifests. Unlike Figure 3.2(a), we see the *P* pages and *E* pages are aggregated similar to Figure 3.3. However, we also see that device *d2* contains a page that would be better placed on device *d1*; similarly, we observe that there is only a single *E* page on devices *d3* and *d6.* If all these pages were migrated onto common devices, two additional devices could be transitioned into lower power states.

Implementation Details. For historical reasons, physical memory is coarsely grouped by zone in Linux [70]. However, many supporting architectures use only a subset of the available zones. Consequently, for this discussion we consider an architecture that primarily uses a single zone. Each zone uses a buddy system as the frame allocator to manage free page frames. To direct page frame allocation requests to specific memory devices by allocation type, we use two buddy systems: one for backing E pages and one for P pages. The free area list from which a frame is allocated is determined by checking a flag bit passed into the allocation request. Because this requires only one additional bit-wise comparison and branch instruction, the overhead is trivial. We incorporated this allocation-time direction within the interface functions for allocating and freeing sets of page frames. All other aspects of the buddy allocation algorithm remain unchanged.

Even though this approach minimizes the probability pinned pages will prevent memory removal, it does introduce the possibility of a balancing problem between allocation types. For example, if the number of free frames of either type becomes scarce, this could cause allocation failures for the requested type. To prevent this scenario, we allow for large contiguous areas to be transitioned from one buddy system to the depleted buddy system. If a page frame set is transitioned from buddy system for the *E* frame set to the *P* frame set, the capability of turning off the memory device may be compromised due to pinned pages. However, transferring frame sets between the two systems prevents artificial memory shortages solely because of the delineation between memory request types. A side effect of this approach is a lower bound on the amount of memory that may be de-allocated. However, immovable kernel pages account for a small amount of total physical memory and at least one memory device must remain powered on to maintain reasonable performance on any static or dynamic memory system.

3.3.4 Hybrid Shaping

To maximize energy efficiency we propose a third approach called hybrid shaping. Hybrid shaping combines the allocation-time page placement of proactive shaping with the migration capability of reactive shaping. Since proactive shaping directs allocations to devices with minimal overhead, the need for migration is minimized. However, as previously discussed, relying solely on proactive shaping can result in suboptimal allocations across devices over time. Hybrid shaping uses reactive shaping to avoid these inefficiencies by periodically aggregating pages onto a minimal device set.

Figure 3.2(c) shows how hybrid shaping work relative proactive and reactive shaping. In this example, page demand is again 50% of system capacity (8 pages, 16 frames). We observe that allocation time placement has aggregated pages by type onto common devices with the exception of device *d7*. Pragmatically, only the *P* page in device *d1* and the *E* page resident in device *d3* are candidates for migration. Because *E* pages are by definition easier to migrate than *P* pages, the page in frame 7 is moved to frame 2 on device *d1*. Optionally, the page in frame 7 may also be paged out to disk; however, this may incur a performance penalty if the page is in active use. After migration half of system capacity may be transitioned into a lower power state.

3.4 Experimental Results

We implemented all of the extensions discussed in the previous section in a 2.6 x86-64 version of the Linux kernel on a recent Intel 64-bit SMP Xeon processor system with various amounts of memory. For some of the experiments our system contained 3Gbytes. In later experiments we populated the system with 8Gbytes of memory. To evaluate our approach, we modified our operating system extensions slightly to emulate memory topologies by partitioning the physical address space into logical devices. In this way, we can experiment using memory devices of any granularity, within the limits of actual system memory capacity. Using emulation, we can evaluate the impact of dynamically adjusting memory device states using the actual, real-time memory demand of various workloads.

In this section, we present energy and performance results using our hybrid approach. Because hybrid shaping is a combination of proactive and reactive shaping techniques, the proactive page placement is integrated into the page allocation process. However, in our implementation page migration is only triggered when a device is offlined by a system-level controller. So, we implemented a root-privileged, application-level daemon using a simple history-based heuristic to monitor page demand and control the power state of all memory devices.

3.4.1 lmbench Results

For our first evaluation we configured lmbench to run the OS-centric set of benchmarks, including the memory-intensive bandwidth codes, to stress our page allocation and migration modifications. Each run was configured to execute the same codes with the same amount of memory. Additionally, we configured lmbench to use a subset of total memory in the system. We ensure the page demand of lmbench is in-core to evaluate page migration.

Our controller uses observed page demand to predict the number of online memory devices required. Specifically, we retain a configurable number of prior observations (10-20) and use those to predict when to offline devices. To ensure a sufficient number of devices were available to satisfy sudden increases in demand, we retained additional devices online beyond the optimal number of devices. Consequently, the control application aggressively on-lines additional devices when demand increases and off-lines devices only when the number of online devices is greater than any of the observations retained in the recent history buffer.

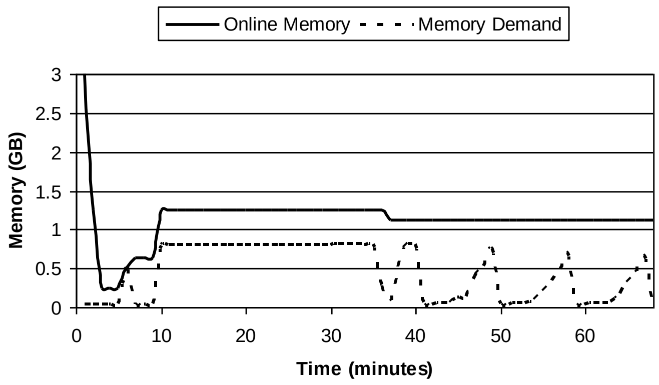


Figure 3.4. Online memory scaling using hybrid page shaping and simple heuristics while running lmbench. The lower line is the actual memory demand observed and the upper line constitutes the online memory device set which closely tracks memory demand and reduces energy consumption of memory.

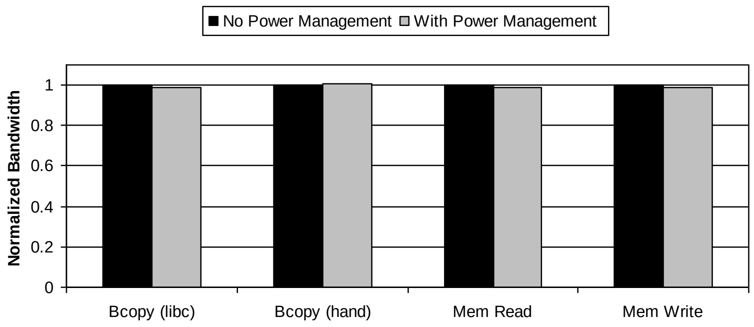


Figure 3.5. Performance result comparison of lmbench codes with our kernel modified to use hybrid page shaping and an unmodified base kernel. For all codes, there is less than 1% difference.

Figure 3.4 plots memory demand and online memory as directed by the controller. For this experiment, we first started the controller, after which we started lmbench. At the beginning of the run all memory devices are online and available. After starting the controller, devices are incrementally off-lined to conserve energy due to low memory demand. When lmbench starts there is a spike in memory demand which causes the controller to online additional memory devices quickly, preventing paging. During the initial execution phase, memory demand remains near constant, but then oscillates towards the end. However, because we are using a simplistic history-based heuristic to control power state transitions, some efficiency is lost as noted by the gap between online memory and actual memory demand. We propose and evaluate control policies in the next chapter. Despite this efficiency gap, retaining additional memory online preserves performance and still reduces energy consumption. For this experiment, using the hybrid shaping, the controller achieved 56.26% energy savings within the memory subsystem, largely attributable to the limited memory demand of lmbench.

We also ran lmbench against an unmodified base kernel to characterize the performance implications of our changes. For nearly all of the specific codes, there was no discernable difference. The only observable differences were in the memory bandwidth codes. Figure 3.5 compares the bandwidth results of the memory intensive lmbench codes of our kernel modified to include hybrid shaping and a base kernel. There was less than a 1% difference as a result of our changes.

3.4.2 SPEC CPU2000 Results

We also experimented using the SPEC CPU2000 benchmarks. In this case, our system was populated with 8Gbytes of memory. Figure 3.6 shows how memory devices are dynamically scaled to meet memory demand of the SPEC benchmarks using our history-based heuristic. Initially, the full 8Gbytes of memory capacity was online and available for use; however, because the SPEC benchmarks do not require significant memory, the controller offlined most of the unnecessary memory devices as also shown in Figure 3.4. Thus, we have omitted the first 5 minutes of the trace in Figure 3.6.

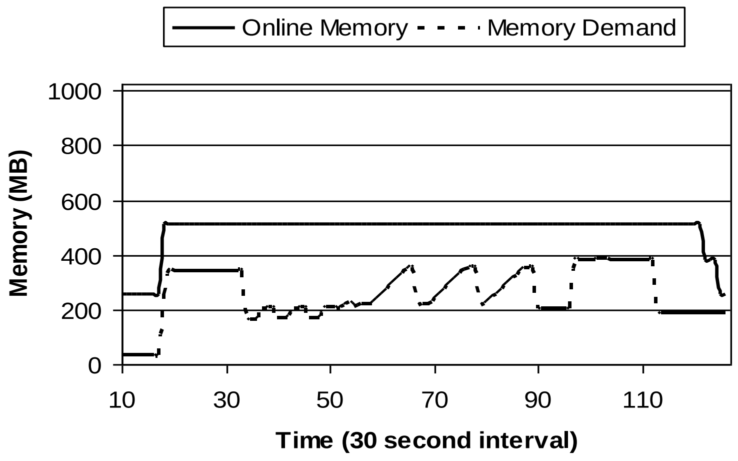


Figure 3.6. Online memory scaling using hybrid page shaping and simple heuristics while running SPEC CPU2000 benchmarks. The lower line shows actual memory demand and the upper line shows online memory which is adjusted based on memory demand.

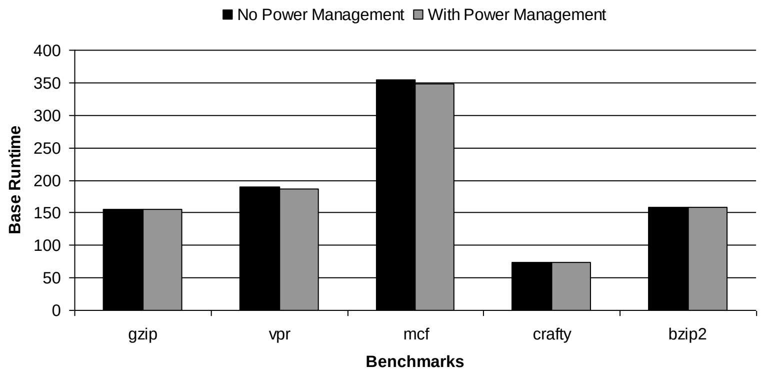


Figure 3.7. Performance result comparison of SPEC CPU2000 codes with our kernel modified to use hybrid page shaping and an unmodified base kernel. For all codes, there is less than 1% difference.

As shown, systemic memory demand increases when SPEC is started and additional devices are onlined to meet the demand. The memory demand of the benchmarks oscillates slightly while executing but never exceeds about 400Mbytes. In this case, a total of about 512Mbytes is more than sufficient to satisfy the demand of all the SPEC benchmarks while executing. Once the benchmarks complete executing, the number of online memory devices is further minimized. This resulted in 81.25% energy savings within the memory subsystem. Our energy savings are significant using SPEC codes due to the low memory demand relative to system capacity. As before, we also ran the SPEC benchmarks on an unmodified base kernel to characterize the performance implications of our changes. Figure 3.7 shows the performance of several SPEC codes using our kernel with hybrid shaping normalized to the base kernel. The performance loss for these codes is maximally about 1%.

3.5 Chapter Summary

In this chapter we quantified the performance and energy for several workloads using a kernel with page allocation shaping techniques and shown that there are significant opportunities to minimize energy consumption on large systems by taking advantage of variable *slack* in system memory demand. We proposed three page shaping techniques: proactive, reactive, and a hybrid variant that combines elements of both. By combining proactive page placement at allocation time and reactive page migration on demand, we achieved significant energy savings (56% for lmbench and 81% for SPEC) with less than 1% performance penalty. We show that applying these techniques in operating systems managing large memory-dense systems can yield significant cost savings by dynamically scaling online memory based on actual memory demand.

Chapter 4

Formal Control Systems for Improving the Energy Efficiency of Sequential Memory Systems

In this chapter we present an analytic model of a feedback control system rooted in formal control theory to dynamically scale online memory capacity while minimizing performance loss. We introduce several models of memory demand that motivates the need for an adaptive, yet stable control system. We then compare our approach with several alternative control policies. We formally describe the control system components; derive the system transfer function, and finally identify control gains through stability analysis. An implementation of our control system that we combined with our previously introduced page allocation shaping techniques to form Memory Management Infrastructure for System Energy Reduction, or Memory MISER is described. We conclude by evaluating Memory MISER in terms of memory energy and application performance under multiple workloads using an individual server as well as an 8-node server cluster.

4.1 Systemic Memory Demand

Systemic memory demand can fluctuate significantly depending on workload. To better understand how applications affect systemic memory demand, we ran commonly used benchmarks in isolation and monitored the ratio of allocated and free memory during execution. Figures 4.1 plots memory demand over time for select codes from the SPEC CPU2000, BioBench, and lmbench benchmark suites; recent studies have similarly characterized the

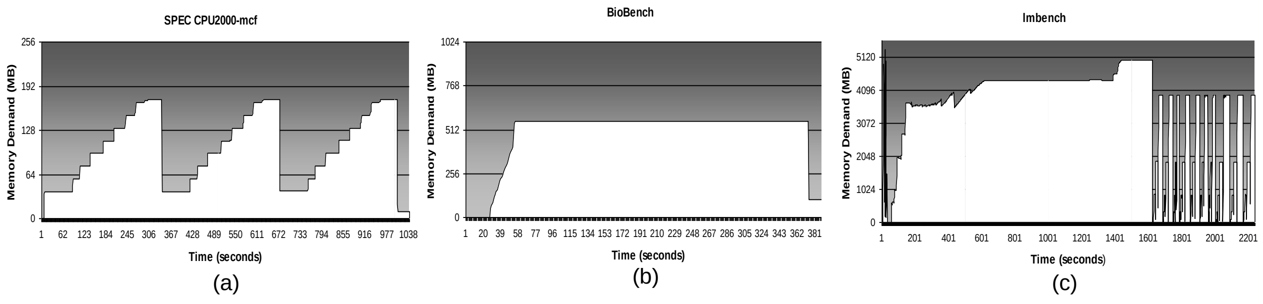


Figure 4.1. Memory demand traces of select (a) SPEC CPU2000, (b) BioBench, and (c) lmbench benchmarks.

memory footprints of the SPEC CPU2006 benchmarks by monitoring the resident set size of each code at runtime [76, 146].

Three distinct patterns emerge in these traces in terms of the rate of change in memory demand which we classify as memory demand signatures. First, we observe some applications cause high volatility in memory demand over short intervals. For example, in the latter half of the lmbench trace (Figure 4.1(c)) memory is repeatedly allocated and freed. We characterize these signatures as *pulsed*, as they cause instantaneous spikes or pulses in demand. In contrast, computationally-intensive applications, such as those used in scientific computing often operate on large, in-core data sets for long periods. Many such applications cause rapid spikes in memory demand upon invocation, but stabilize to steady-state demand at runtime. Such applications retain allocated memory during execution to maximize performance, but rapidly free memory upon completion, such as large-scale applications that handle memory management internally. We define these applications as having *stepped* memory demand signatures. In Figures 4.1(a) and 4.1(b) the memory demand of BioBench and SPEC's mcf exhibit stepped signatures. Other applications have a seemingly random impact on system memory demand. For example, many web-based applications allocate memory proportional to the number of active sessions. Such runtime allocations manifest as random perturbations in memory demand. Accordingly, we classify these applications as having *random* memory demand signatures. Running applications with different memory demands concurrently, as is typical on multicore systems, often results in random memory demand signatures.

4.2 Scaling Online Memory Capacity

If we could proportionally scale the online memory capacity of systems to meet the memory demands of applications at runtime, we could increase the energy efficiency of memory while maintaining performance. However, this is a challenging problem. First, demand is often unpredictable and fluctuates significantly. Memory scaling techniques must be robust and flexible to cope with disparate demand signatures. Second, transitioning memory devices between multiple power states at runtime is not free. There is overhead, so memory scaling techniques must avoid or minimize the overhead so performance is not sacrificed. Third, historically memory demand has been serviced transparently by the operating system. We do not want to introduce techniques that reduce or remove this transparency. These three challenges motivate the need for flexible control techniques that maintain transparency with minimal overhead.

4.3 Control System Policies

The high variance in memory demand requires a flexible control policy capable of responding to rapid changes in demand. Figure 4.2 contrasts the impact of three control policies for balancing performance and energy consumption using a sample workload on a system with 10 memory devices. We identify the first approach shown in Figure 4.2(a) as the *default* control policy. This

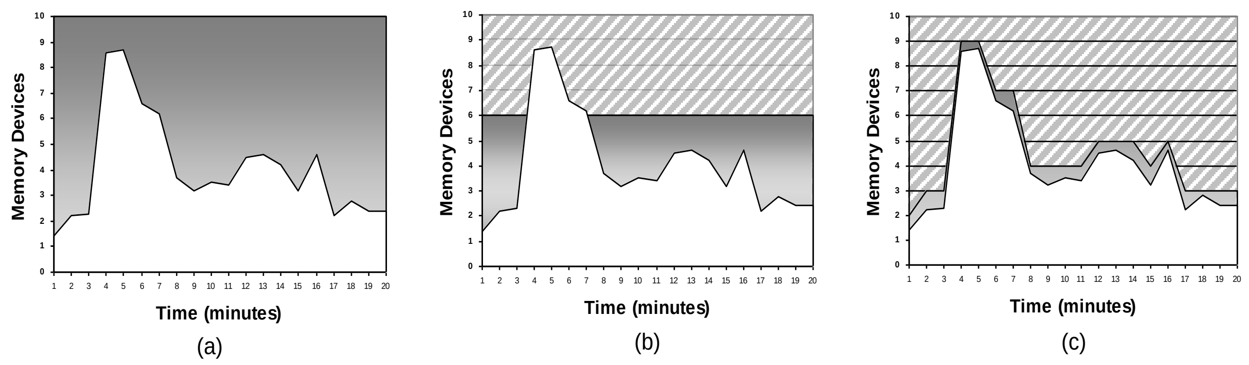


Figure 4.2. Policy comparison for managing memory power and performance. The white area constitutes page demand, the solid gray area is wasted energy, and the hatched area represents conserved energy. The default policy (a) retains all devices online to maximize performance, but wastes significant energy when memory demand is less than full capacity. The static control policy (b) retains a subset of devices online realizing energy savings relative to the default policy, but suffers performance penalties when demand spikes. The dynamic policy (c) tracks memory demand and scales the online memory device set to meet demand, preserving performance and minimizing wasted energy.

policy simply keeps all devices in a high-power state to ensure optimal performance. Since demand peaks at 90% of the available capacity, performance penalties are avoided. However, because memory demand is less than half of system capacity most of the time, several devices go unused. Without the flexibility to offline unused devices, the default control policy wastes energy when demand is less than total capacity.

Unlike the default approach, *static* control policies can reduce system energy consumption. These preserve transparency and minimize control overhead for workloads with static demand, but are often inefficient when demand fluctuates due to inflexibility. Figure 4.2(b) illustrates a static control policy tuned for a specific workload. This policy reduces the online device set to 6 devices, achieving a 40% reduction in memory energy over the default policy. For a workload that requires fewer than six devices, such a policy would yield savings without a significant performance penalty. However, significant performance loss results when demand spikes, causing in-core workloads to execute out-of-core. For example, in Figure 4.2(b) demand exceeds the capacity of the online device set for about 4 minutes, or 20% of the default approaches execution time. This causes in-memory pages to be swapped out to free sufficient frames for new allocations, increasing access latencies and degrading application performance. For such static control policies, workload-specific tuning is often required to ensure performance is preserved. This means that the policy must be updated whenever the application set changes.

The inflexibility of default and static control policies motivates our approach of dynamically scaling the online device set to match memory demand. Figure 4.2(c) illustrates the flexibility of a *dynamic* control system; the online device set is scaled to satisfy demand within memory device granularity. By scaling memory capacity, the performance penalty of causing in-core workloads to execute out-of-core is avoided and wasted energy is minimized. To quantify the differences in this simple example, dynamic scaling achieves 54% memory energy savings relative to the default policy, as well as a 14% improvement over the static control system without any performance penalty.

Dynamic control systems often employ different controllers. Simple, heuristic-based controllers have been effectively integrated into dynamic control systems in various domains [9, 45, 117]. However, after initial attempts using heuristics for memory control, we found this approach required time-consuming, iterative, and application-specific tuning to achieve desired performance and stability. In some cases, changing application parameters requires retuning the heuristic, increasing verification and validation costs. Moreover, unexpected application behavior can lead to instability and poor performance. In contrast, control theory provides a standard tool set for analytic formulation of robust and flexible control systems with discernable stability and performance characteristics [7-9, 109, 116].

4.4 Analytic Model of memory device control

4.4.1 Formal Feedback Control Systems

Feedback control systems adaptively track a changing variable by altering its response based on the error or difference between the set point and a process variable. A common controller used in feedback systems is the Proportional-Integral-Derivative, or PID controller. Historically, PID controllers have been used to control industrial systems [7, 9]. More recently, PID variants have been effectively applied to control TCP traffic flows [79, 126], dynamic voltage and frequency scaling between multiple clock domains within processors [164], and processor thermal dissipation via instruction fetch regulation [143]. Motivated by the performance characteristics and stability properties of PID controllers, we use a PID controller in our control system to manage memory device power state transitions.

In this section we introduce an analytical model of a control system for dynamically scaling online memory capacity to meet memory demand. We describe each of the control system components, derive the system transfer function, and then describe selection of control gains. Readers familiar with control theory are invited to skim the equations in this section and proceed to Section 4.5.

4.4.2 Memory Device Control System Model

Figure 4.3 presents a block diagram of the analytic feedback control system model we developed to dynamically scale online memory while minimizing performance loss.

The input signal to our control system is current memory demand. We model memory demand using a page demand signal generator that samples systemic page demand and outputs the memory demand signal *(r)*. This signal constitutes the set point tracked by the control system. Memory demand is discussed in detail in section 4.4.3.

We model the control system as a single-input, single-output system with feedback. Because the control system includes a feedback loop, the output signal is continually compared with the set point. The difference between the input signal (memory demand) and the feedback signal (online memory capacity) constitutes the error signal, used as input by the controller. When the error is negative, online memory capacity exceeds demand, indicating the system should offline devices. When the error is positive, demand exceeds online memory capacity and additional memory should be onlined to avoid paging.

The controller *(C)* adjusts the rate at which the memory device set changes occur in response to the error signal. Because we sample memory demand in our system, we use a discrete form of the Proportional-Integral-Derivative (PID) controller. PID controllers include several key parameters, known as control gains which provide design flexibility, but must be carefully selected through stability analysis. The controller derivation and parameters used in this study are discussed in section 3.4.

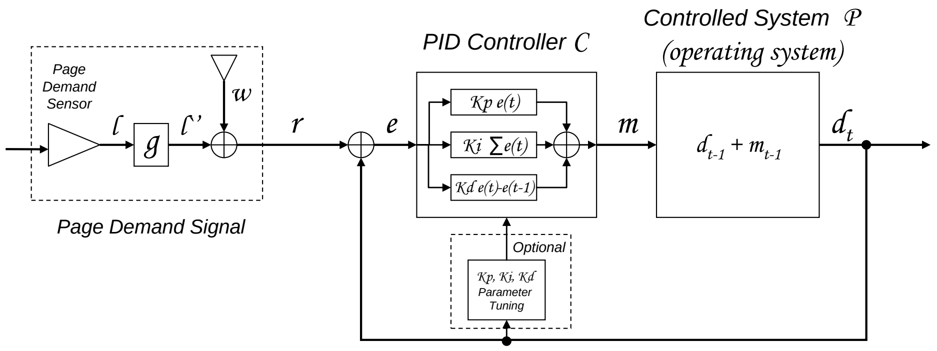


Figure 4.3. Control system model for managing memory power states.

Within the feedback control system, the plant function ($\mathcal{P}$) models the controlled system; in our case the controlled system is the OS memory manager. The plant is a single-input, single output function that uses the controller output signal to manage and modify the state of all system memory devices. The output signal of the plant function is online memory capacity, which also serves as the feedback signal in the control loop. As previously mentioned, the difference between the feedback signal and set point forms the error signal used in the next iteration of the control system.

4.4.3 Modeling Page Demand

We use a simple signal generator structure to model memory demand as shown in Figure 4.3. At the core of the generator is the page demand sensor, which discretely samples page demand at a configurable frequency. The page demand sensor determines the number of allocated pages within the system; these include resident pages (when demand is less than the capacity of online devices) or a combination of in-core pages and out-of-core pages (when demand exceeds the capacity of online devices). We track memory demand at page granularity even though most memory devices are typically larger than a single frame. To account for the disparity between coarse, device-level control and high resolution tracking we transform the page demand signal from the high-resolution signal *l* to device-granular signal *l'* via a simple filter (block *g* in Figure 4.3), where *l' = ceiling(l/# pages per DIMM)*.

4.4.4 Controller Model and Derivation

Formal control theory provides a mathematical foundation for designing robust and stable feedback systems with discernable performance properties. One common controller used in control-theoretic, feedback systems is the Proportional Integral Derivative, or PID controller. A PID controller is generically described by the following equation in the continuous time domain:

$$m(t) = K_P e(t) + K_I \int e(t)dt + K_D \frac{de(t)}{dt} \tag{4.1}$$

m(t) is the output of the controller at time *t* based on the measured error within the system. For our system, this amounts to the number of memory devices that should be onlined or offlined to satisfy memory demand in the next interval. As shown in equation 4.1, the output of the PID controller is the algebraic superposition of three terms:

- $K_P\ e(t)$: The first term is proportional to the error. This causes the system to respond to the error value and direction. Since our system controls memory capacity at device granularity, we define error in terms of number of devices.
- $K_I \int e(t)dt$: The second term is proportional to the integral of the error. This aggregates error over an interval to eliminate the steady-state error.
- $K_D\ e(t)d/dt$: The third term is proportional to the rate of change in the error, reducing overshoot by directionally damping the response.

The terms K_P, K_I, and K_D are control gains that are analytically determined through stability analysis. Poor selection of these parameters can cause system instability manifested as output signal oscillation – which in our case would cause main memory device thrashing and catastrophic performance losses. Careful selection of control gains results in a controller with desirable convergent, performance properties. The first step in determining these parameters is to identify the controller transfer function. Since we sample system memory demand at a configurable frequency, our system is inherently discrete, thus we use the discrete form of equation 1. The z-transform of the discretized PID controller transfer function is:

$$M(z) = \frac{(K_p + K_i + K_d)z^2 - (K_p + 2K_d)z + K_d}{z(z-1)} \tag{4.2}$$

Although the controller transfer function is a constituent of the system transfer function, the complete control system transfer function is necessary to analyze the stability of the control gains. In other words, the controller parameters are determined by considering the interaction of the controller function and the plant function. Having determined the controller transfer function, the next step is to analytically model the controlled system. Relating the transfer functions of the plant model and the controller, we can select and analyze control gains for stability.

4.4.5 Controlled System

In our system, the OS memory manager controls the power state of all memory devices in the system, so we model the memory manager as the plant function. We assume each memory device is capable of transitioning into at least two power states, which we generically classify as either online or offline. These constitute the boundary power states for a given technology. Pragmatically, devices in an offline state may be electrically offline (consuming zero power) or electrically idle (consuming minimal power from a standby well). We characterize online devices as those in a high-power, low-latency state that may be activated quickly to satisfy memory accesses. We avoid modeling restrictions that preclude including additional power states as we plan to explore optimizations to exploit intermediate states in future work.

Within our control system, the output signal of the PID controller constitutes the input signal to the plant. The plant input signal specifies desired changes in online memory capacity at device granularity. Based on the input signal, the plant transitions memory device states (online or offline), resulting in an updated online device set. The size of the new online device set forms the output signal of the plant. We model this change using the following first-order difference equation in the time domain:

$$P(t) = d_t = d_{t-1} + m_{t-1} \tag{4.3}$$

In this equation, d_t is the size of the online device set at time *t* and is calculated based on the size of the online device set in the previous interval $d_{t\text{-}1}$ plus the change specified by the controller. Realistically, d_t has an upper bound, based on the number of memory devices in the system. For this study we assume at least one device must remain online at any time, further bounding d_t such that, $1 \leq d_t \leq d_{max}$, where d_{max} is the maximum number of devices in the system. Applying the z-transform to equation 3 results in the following transfer function for the plant function in the z-domain:

$$P(z) = \frac{z}{z-1} \tag{4.4}$$

Using equation 4.4 for the plant transfer function, we proceed with identification of the control system transfer function by relating the plant transfer function with the controller transfer function.

4.4.6 Control System Transfer Function

A feedback control system transfer function can be expressed generically by the following equation:

$$D(z) = \frac{M(z)P(z)}{1 + M(z)P(z)} \tag{4.5}$$

The numerator reflects the feed-forward transformation of the input signal through the controller and plant transfer functions to the output signal, while the denominator relates the effect of the feedback loop to the output signal. Substituting equations 2 and 5, for M(z) (the full PID controller) and P(z) (the plant) respectively, results in the following transfer function for our feedback control system after algebraic simplification:

$$D(z) = \frac{(K_p + K_i + K_d)z^2 - (K_p + 2K_d)z + K_d}{(K_p + K_i + K_d + 1)z^2 - (K_p + 2K_d + 2)z + K_d + 1} \tag{4.6}$$

Given the transfer function for the full control system, we evaluate control gains using stability analysis. Solving the polynomial in the numerator for zero identifies the zeroes of the transfer function; these characterize the relative excitation of each term [8, 9, 116]. The denominator of a transfer function, also known as the characteristic equation, determines the stability of the control system. From our transfer function, (equation 4.6) we can analyze the stability characteristics by solving the characteristic equation for zero, specifically:

$$(K_p + K_i + K_d + 1)z^2 - (K_p + 2K_d + 2)z + K_d + 1 = 0 \tag{4.7}$$

The roots of the characteristic equation constitute the poles of the system. To be stable, each of the poles must reside within the unit circle, or region of convergence. Parameters that result in pole placement outside the region of convergence lead to system instability. In this case, changes in the set point or disturbance signal will cause signal oscillations or non-convergent response behavior. In contrast, careful selection of control gains results in poles that reside within the region of convergence, so the system will eventually converge despite set point changes or variations in the disturbance signal.

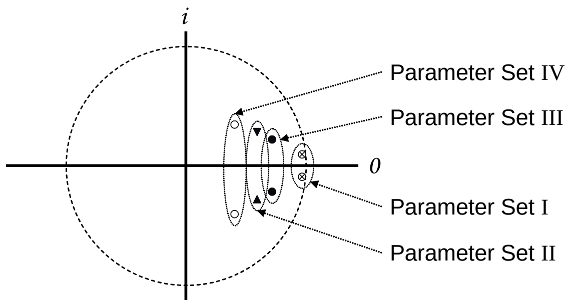


Figure 4.4. Pole placement of parameter sets in table 1 plotted in the complex domain. As shown, the poles using the four parameter sets reside within the unit circle, ensuring control system stability.

For our study, we experimented with numerous parameter sets, four of which are outlined in Table 4.1. Substituting these parameters into the characteristic polynomial (equation 4.7) and applying the quadratic formula to find the roots, we find the poles are clearly within the unit circle. These are also plotted in the complex domain in Figure 4.4.

Although the parameter values in Table 4.1 result in system stability, each parameter set affects performance and transient response differently. As shown in equation 4.1, each term *(K_P, K_I, and K_D)* weights the respective term which collectively determines controller output. For the experimental results reported in this paper, we used the values in parameter set I in Table 4.1 (K_P=K_I=K_D=0.015625) for the control gains.

Table 4.1. PID Controller parameter sets

Set	Control Gains	Poles	Zeroes	Stable?
I	K_P=0.015625 K_I=0.015625 K_P=0.015625	*0.97+0.12i* *0.97-0.12i*	*0.58,* *0.084*	*yes*
II	K_P=0.9 K_I=0.8 K_P=0.4	*0.59+0.30i* *0.59-0.30i*	*0.4+0.16i,* *0.4-0.16i*	*yes*
III	K_P=1 K_I=0.5 K_D=0.8	*0.69+0.23i* *0.69-0.23i*	*0.56+0.16i,* *0.56-0.16i*	*yes*
IV	K_P=0.8 K_I=1.2 K_D=0.4	*0.52+0.36i* *0.52-0.36i*	*0.33+0.23i,* *0.33-0.23i*	*yes*

4.5 Memory Miser Implementation

Memory MISER consists of two primary components: a user-level daemon implementation of a PID controller for controlling memory power state transitions and a modified Linux kernel. The implementation and changes to the Linux kernel are real and portable across system architectures. Memory devices capable of transitioning to a fully offline state at DIMM granularity were not available at the time of this work. So we ran our modified kernel and controller implementation on a real system with standard DDR2 SDRAM. We then modified our kernel to emulate devices of various sizes to evaluate our kernel and controller as well as to study the effects of varying the device sizes that can be offlined. Though we have other results, in this paper we report emulated results for 512Mbyte and 128Mbyte DIMMs on systems with between 6Gbytes (cluster nodes) and 8Gbytes (stand-alone system) capacity.

4.5.1 Controller Implementation

For the controller, we initially implemented a history-based heuristic-driven controller in a root-privileged, application-level daemon as described in chapter 3 [151]. Although we designed the heuristic to avoid adversely impacting performance, it missed many opportunities to turn off memory devices. Even after expanding on the heuristic to consider additional information, it still missed opportunities. This led us to investigate other controllers including the PID. Consequently, we modified the daemon to use a PID controller. This reduced the overhead of the control daemon and simplified the implementation, reducing the primary control function code size by nearly 80%.

We experimented with various sets of PID controller parameters, including the four described in Table 4.1. For the experiments reported in the next section, we used parameter set I from Table 4.1; that is, $K_P=K_I=K_D=0.015625$ or $(1/D_{MAX})$, where D_{MAX} is the maximum number of memory devices available in the system. The control system was stable under all workloads and exhibited reasonable transient response using these parameters.

As previously discussed, we use a discrete PID controller in our daemon implementation as systemic memory demand is sampled at a configurable frequency. To minimize overhead, our initial implementation sampled demand every 5 seconds; however, when demand changed rapidly (such as the case with *pulsed* demand signatures), this limited the effectiveness of the controller and resulted in paging. To more closely track demand we changed the sampling frequency to 0.5 seconds.

To further minimize the possibility that demand spikes caused paging, we added a filter (*w* parameter) to the page demand signal generator. This filter dynamically adjusts the set point signal and serves as our primary mechanism for weighting performance or energy efficiency. In an ideal system, the weighting parameter is set to zero ($w=0$) which balances performance with energy efficiency. In this case, the input signal to the control system is exactly memory demand and only the minimal set of devices necessary to satisfy demand will be retained online. For example, if measured memory demand requires 4 devices ($l'=4$), w is set to 0, and 6 out of a total of 8 devices are currently online but free, the resultant error signal sent to the controller would be -2 (e=6-8). This indicates two additional devices beyond the optimal are online. Increasing the *w* parameter weights performance over energy efficiency. The system still tracks demand, but retains additional devices online, relative to the set point. Assuming the same example scenario, where memory demand requires 4 devices and 6 out of 8 devices are online, if we set *w=2*, the input signal to the controller would be 0 *(e=6-8+2=0)* indicating equilibrium from the controller's perspective. Similarly, setting *w* to less than zero weights energy efficiency over performance. In this case, the control system continues to track demand, but retains proportionally fewer devices online than necessary to satisfy demand.

After shortening the sampling interval to 0.5 seconds and increasing *w* to 4 (the maximum number of devices the system is capable of allocating during 0.5 second sampling interval), we were able to eliminate paging for in-core workloads, even with *pulsed* demand signatures.

4.5.2 Operating System Implementation

As described in Chapter 3, we modified a 2.6 x86-64 version of the Linux kernel with the capability to transparently online and offline physical memory devices while preserving application integrity. This is an extension of earlier work [150] where we modified the Linux kernel to support hot-pluggable memory devices. In that study we focused on preserving system uptime by dynamically changing page to frame mappings in response to physically changing memory capacity.

Building on the same online and offline capabilities, we implemented and evaluated several page allocation *shaping* policies to reduce memory energy consumption. We first investigated *reactive* page shaping, which used migration to periodically consolidated pages from frames scattered through the physical address space to a minimal device set. To reduce migration overhead we developed *proactive* page shaping to always allocate frames from a minimal device set. Proactive shaping also reduced the complications associated with DMA-intensive workloads, since we can direct DMA transactions to memory regions backed by devices that we always keep online. However, workloads that rapidly allocate and free memory resulted in fragmentation across devices, wasting energy. In the MISER implementation we incorporated a combination of both approaches, which we call *hybrid* page shaping. Hybrid page shaping enables power-state driven device selection at allocation time with periodic migration to minimize fragmentation across devices. We found hybrid shaping yielded significant energy improvements [151, 152].

Modifying an OS memory manager can impact runtime performance. We measured page faults and execution time running SPEC benchmarks, LMBench, and the NAS Parallel Benchmarks using our MISER-enabled kernel as well as using an unmodified kernel. For consistency we disabled the controller daemon so memory remained online during the comparison. We observed less than 1% performance variation between kernel implementations over a number of runs for all codes.

4.6 Experimental Results

4.6.1 Serial Results

For our single-server evaluation, we used a dual-processor Intel® system with 8 gigabytes of DDR2-400 memory. We emulate a dense memory topology by logically partitioning physical memory into 64 128MB logical DIMMs. Although most server systems use higher capacity DIMMs, we chose 128MB to evaluate the scalability and controllability of our implementation despite the limited capacity of our test system. Further, this degree of controllability reflects the number of devices (albeit at lower density) used in real systems with large-scale memory topologies [71].

Onlining DIMMs is not instantaneous. Through instrumentation we found that BIOS code for initializing four 4Gbyte DIMMs took approximately 600-700ms. To conservatively account for hardware delay in onlining DIMMs, we used a fixed delay of 500ms for each online operation. Because off lining does not require re-initialization, only the dropping of delivered power and signals to a slot, we added a fixed delay of 50 ms for each offline operation. These delays are included in all of our evaluation results.

We selected benchmark applications from well-known suites, including SPEC CPU2000, NAS, BioBench, and lmbench as well as several synthetic codes, to evaluate Memory-MISER on a single system. We ran all of these benchmark applications individually, iteratively, and concurrently to further characterize systemic memory demand in terms of the pulsed, stepped, and random signatures described in section 4.1. Though we have additional results, in this paper we present benchmarks that exhibited each of the signatures to compare the performance and energy impact of the three control policies. Since most machines retain all memory devices online at runtime, we use the default policy discussed in section 4.1 as the baseline for comparing performance and energy results. Consequently, all energy improvements presented in this section are normalized to the default control policy.

4.6.1.1 LMBENCH

The lmbench suite includes benchmarks for evaluating various aspects of system performance. However, many of the codes test specific processor and network latencies and are not significantly impacted by memory capacity. We ran lmbench using all three control policies and found the only significant differences were in the memory bandwidth benchmarks. We subsequently focused our lmbench evaluation to use the memory bandwidth codes including: a)

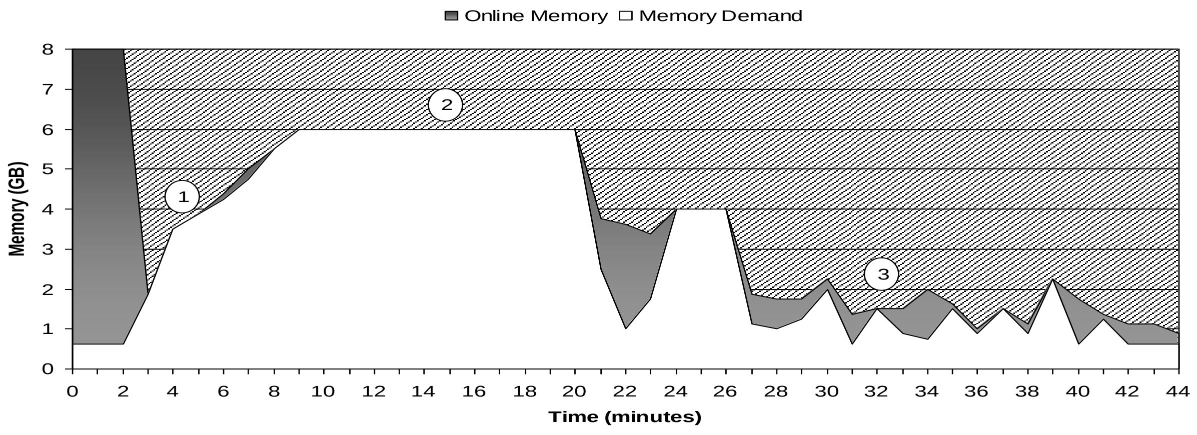


Figure 4.5. Memory MISER dynamically scales online memory to meet demand while running lmbench. The white area shows memory demand, the gray area shows online memory as directed by the PID controller, and the hatched area shows offline devices constituting energy savings.

libc version of bcopy b) hand-tuned version of bcopy c) memory read and d) memory write benchmarks. These benchmarks may be configured to maximally use a significant percentage of online memory capacity; however, each allocates and accesses memory differently. For each run, we used the standard lmbench start-up scripts, which configures and runs selected benchmarks after detecting available memory. For all presented results, lmbench used 5638Mbytes of the total 8Gbyte capacity. We found the collective memory demand of these codes exhibited all three demand signatures when the benchmarks were run serially. To maximize the memory used by lmbench, we started the controller daemon after memory was detected by the start-up scripts. For the tuned static control policy, we retained sufficient memory to satisfy demand. For the untuned static control policy, we offlined half of the memory devices after lmbench started to show the effect of an unexpected change in demand.

Figure 4.5 shows memory demand of the lmbench memory bandwidth codes run serially and how memory capacity was scaled by Memory-MISER. Initially all memory (8Gbytes) is online and memory demand is low (less than 1Gbyte). Once the controller is started, devices are

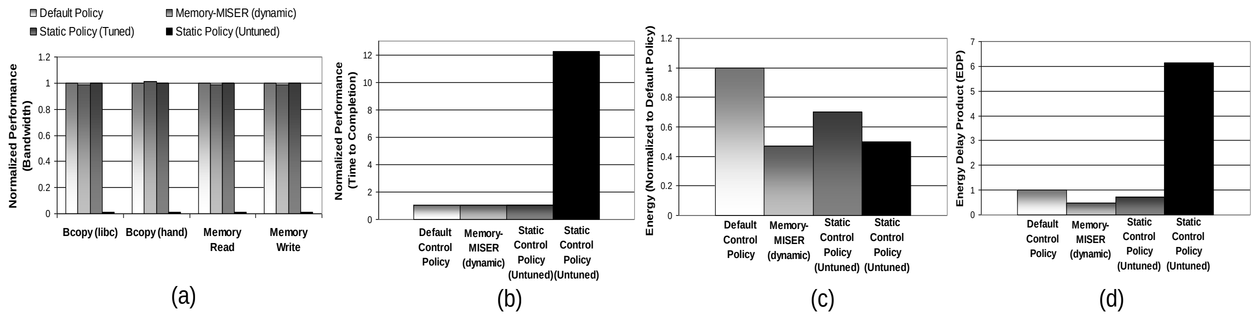


Figure 4.6. (a) Memory bandwidth of default, dynamic (Memory-MISER), and tuned as well as untuned static control policies normalized to default control policy. For all memory-intensive lmbench codes, there is less than 1% difference while using Memory-MISER. Using a static control policy tailored for the workload does not impact bandwidth but severely degrades achievable bandwidth if workload requirements change. (b) Time-to-solution performance of default, static, and dynamic (Memory-MISER) control policies normalized to default control policy under lmbench. Memory-MISER performance is within 0.8% of the default policy. (c) Memory energy consumption of default, dynamic (Memory-MISER), and tuned as well as untuned static control policies normalized to the default control policy. Ideal energy consumption necessary to meet memory demand is 38% of total. Memory-MISER is within 10% of ideal energy consumption with less than 1% performance loss (d) Energy-Delay Product (EDP) results for default, dynamic (Memory-MISER), tuned and untuned static control policies normalized to the default control policy. Relating energy savings with performance loss shows Memory-MISER all of the policies

incrementally off-lined due to the low memory demand. As the controller offlines devices, demand rapidly increases as the benchmarks start executing. We identify the initial demand pulse in Figure 4.5 as phase 1. Memory-MISER detects the change in demand signal, which propagates to the PID controller as a change in set point. The change in the set point manifests as error relative to the feedback signal (online memory capacity in the last interval) and additional devices are quickly onlined. Because we use a full PID controller, sufficient memory is brought online to meet the increased demand without paging.

In the second phase of Figure 4.5, memory demand plateaus and remains near-constant while the memory bandwidth tests run. Since demand is static, only a minimal number of devices are kept online to precisely satisfy demand. Once the bandwidth tests complete, memory is freed bas shown by the precipitous decline in demand.

In the third phase, memory demand exhibits the random signature. There are significant spikes in demand, often followed by short steady states as shown by the second plateau. Because demand is volatile during this phase the set point changes frequently, yet the system remains stable and converges towards the updated set points. This is because controller parameter selection was driven by stability analysis.

Performance Results and Analysis: The default control policy ensures optimal performance by retaining all devices online in a high-power state. Since our lmbench workload remains in-core when all devices are online, the default control policy results in optimal performance.

Lmbench reports achieved memory bandwidth upon completion. Figure 4.6(a) compares the memory bandwidth achieved for specific codes using the four control policies. The bandwidth figures are normalized to the default control policy. Using Memory-MISER and the program normalized, or tuned static control policy memory bandwidth remained within 1% of the default control policy. Expectedly, bandwidth was severely reduced using the untuned static control policy since demand was higher than the online capacity and the system was forced to page. Executing lmbench out-of-core, memory bandwidth was reduced to less than 5% of the default, untuned static or Memory-MISER policies due to paging overhead.

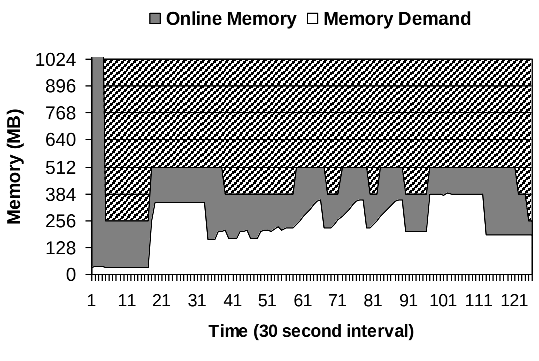
We also measured total lmbench execution time, or time-to-solution (TTS), for the three control policies. Figure 4.6(b) compares the average TTS over 10 runs for each of the control policies normalized to the default control policy. The average TTS for the default control policy was 40.58 minutes, the highest performance among the three control policies. The average time-to-solution for Memory-MISER was 40.9 minutes, an increase of 0.8% relative to the default control policy. The averge TTS for the program normalized static control policy was 40.68 minutes. In contrast, the average TTS for the untuned static control policy was 499.76 minutes -

over an order of magnitude longer than both the default control policy and Memory-MISER. Again, this was due to the extensive paging resultant to executing lmbench out-of-core. The page demand traces recorded during all of the runs revealed that paging was avoided using the default, tuned static, and Memory-MISER control policies.

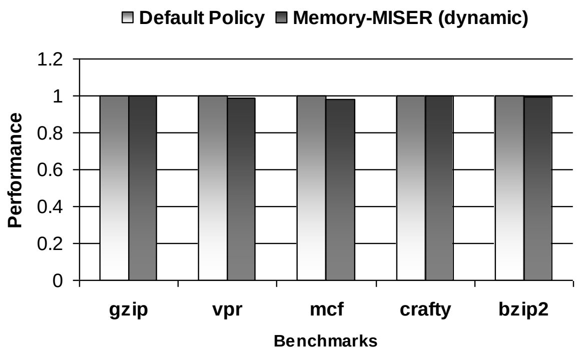
Energy Results and Analysis: Figure 4.6(c) compares the energy consumption of the three control policies, normalized to the default control policy. By tracking memory demand and offlining unneeded devices, Memory-MISER only consumed 47.12% of the energy consumed by the default control policy, realizing 52.88% energy savings. These savings were achieved while preserving performance – recall memory bandwidth and TTS were within 1% of the default control policy.

Because the tuned static control policy minimized the number of online devices based on highest-workload demand, 30% energy savings were achieved over the default control policy. Similarly, the untuned static control policy reduces power by naïvely halving online capacity based on the requirements of a different application set. As a result, bandwidth and time-to-solution performance were severely degraded because memory demand exceeded online capacity, causing the workload to execute out-of-core.

To relate achieved energy savings to performance, Figure 4.6(d) presents the energy-delay product (EDP) for the four policies normalized to the default control policy. The normalized EDP of the untuned static control policy is over 6 times worse (6.155) than the default control policy. This is because the performance degradations outweigh the energy saved. In contrast, because Memory-MISER minimized performance loss (0.8%) and reduced energy consumption by 52.88%, the normalized EDP of the dynamic policy is less than half of the default control policy (0.474).



(a)



(b)

Figure 4.7. (a) Online memory scaling using Memory-MISER while running SPEC CPU2000 benchmarks. The lower line shows memory demand. The upper line constitutes the online memory as controlled by memory-MISER. Note the system has a total capacity of 8Gbytes, but we have omitted the initial phase in this graph during which devices were offlined by the controller to show demand and device scaling in greater detail. (b) Comparison of performance results for SPEC CPU2000 benchmarks using Memory-MISER and an unmodified base kernel. For all codes, there is less than 1% difference in performance.

4.6.1.2 SPEC CPU2000

We also ran the SPEC CPU2000 benchmarks individually and serially to evaluate Memory-MISER. Figure 4.7(a) shows how memory devices are dynamically scaled to meet the memory demand of the SPEC benchmarks when run serially. Initially, all devices are online, but after starting the controller, devices are incrementally offlined based on low demand. Memory demand rises when the SPEC benchmarks start executing and two additional devices are onlined to meet the increased demand. Although demand fluctuates, the variations are sufficiently small that Memory-MISER simply retains 512Mbytes online until demand drops dramatically. Because we retained 8 devices (1Gbyte) to account for pinned pages and Memory-MISER retained 4 devices (512Mbytes) for easy-to-move pages [151], only 18.75% of system memory was kept online. This resulted in 81.25% power savings over the time period SPEC was running.

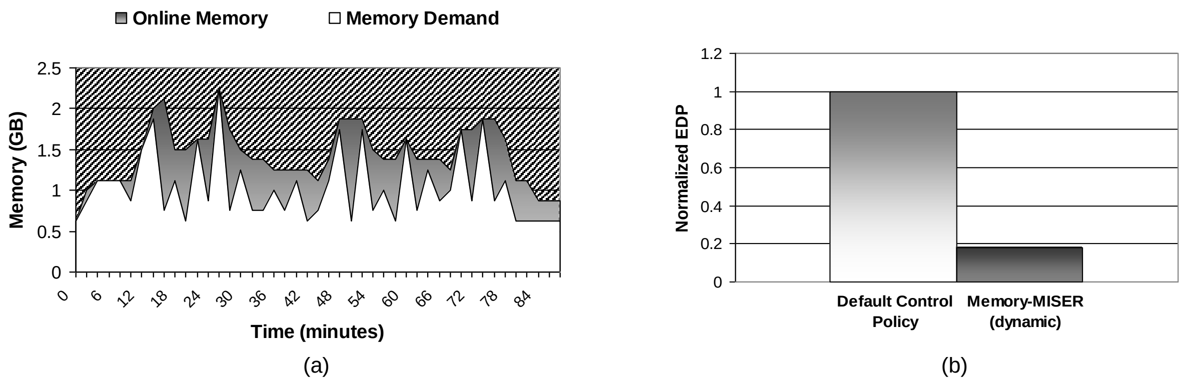


Figure 4.8. (a) Online memory scaling while concurrently running SPEC CPU2000 benchmarks and synthetic, memory-intensive applications. The lower line is memory demand. The upper line constitutes the online memory device set. Memory demand has a random signature due to timing of application memory allocations. (b) Energy Delay Product for Memory-MISER normalized to the default control policy while running SPEC CPU2000 benchmarks and synthetic, memory-intensive application. Ideally, the energy consumption of memory demand is 13.02%. Compared to the default control policy, memory energy consumption is minimized to 17.88% by Memory-MISER.

The energy savings achieved running these benchmarks are significant because the page demand of the SPEC CPU2000 benchmarks is low.

Unlike lmbench, the SPEC benchmarks only require limited memory, so there is not a significant variation in memory demand relative to the capacity of our test system. To induce higher-variance in demand, we ran the SPEC mcf benchmark and two instances of a synthetic, memory-intensive application concurrently. Each synthetic application executes a loop that allocates a configurable amount of memory (in this case 1Gbyte), writes into the newly allocated memory (to ensure frame allocation within the OS), and then frees the memory.

Figure 4.8(a) shows the memory demand of mcf and the synthetic application (lower curve) and the number of online devices as controlled by Memory-MISER. The controller precisely meets demand early in the run and onlines additional devices as demand increases. Throughout the trace, the controller continually minimizes the online device set. Because demand is highly volatile, Memory-MISER fails to optimally meet demand by missing opportunities (shown as the gray areas) to offline devices when demand quickly decreases. However, relative to system memory capacity, we achieve significant energy savings. Further, performance is preserved for all applications. As a result, Figure 4.8(b) shows the energy-delay product for the application set (mcf and synthetic application) was 0.1788 normalized to the default control policy.

4.6.1.3 NAS Serial Benchmarks

We also ran several experiments using the serial versions of the NAS benchmarks. The NAS benchmarks are used to approximate the characteristics of scientific applications. We recorded the memory demand of all NAS serial codes and found they all exhibited stepped memory demand signatures, but similar to the SPEC benchmarks, they did not allocate significant memory relative to our test system.

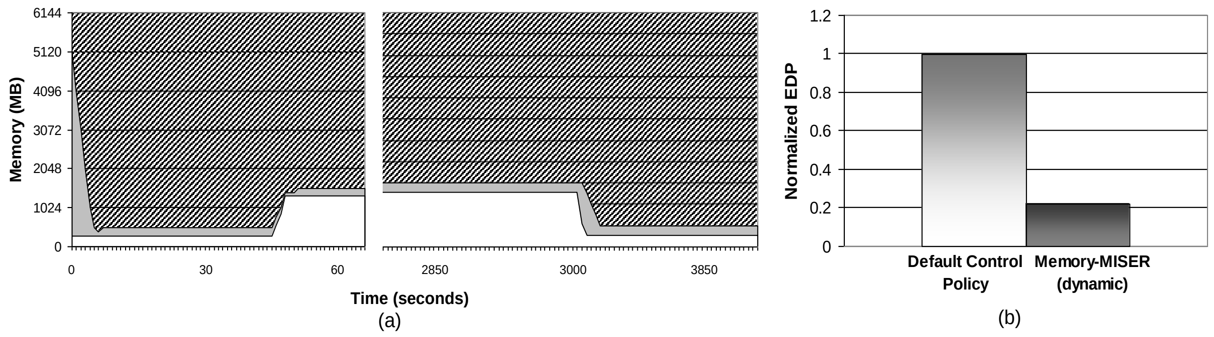


Figure 4.9. (a) Online memory scaling while running the serial version of cg.C NAS benchmark. The lower line shows memory demand. The upper line shows the online memory as controlled by Memory-MISER. The system has a total capacity of 8Gbytes, but we have omitted the initial phase in this graph during which devices were offlined by the controller. We have also omitted the middle of the trace to show how the memory was scaled when demand changed in greater detail. (b) Energy Delay Product for Memory-MISER normalized to the default control policy while running the serial version of the NAS cg.C benchmark. We show the results for cg.C since it s memory demand is characteristic of the serial NAS benchmarks in our experiments. Due to the limited memory demand, we achieved normalized EDP of 0.2193.

Figure 4.9(a)e shows memory demand (lower line) and online device scaling (upper line) as controlled by Memory-MISER while running the cg.C benchmark. Memory demand increases rapidly upon invocation and remains steady for the duration of execution. Once the benchmark completes, demand drops as allocated memory is freed. Despite the rapid increase and decrease in demand, Memory-MISER precisely scaled the online device set. As a result, the performance difference between the default control policy and Memory-MISER was less than 0.2% over 10 runs. Since memory demand was low, we achieved energy savings of 78.07%. Because the performance impact was neglible and online capacity was minimized, Figure 4.9(b) shows Memory-MISER achieved EDP of 0.2193 normalized to the default control policy.

4.6.2 Parallel Results

For our clustered system experiments, we used an 8 node cluster of dual AMD Opteron servers. Each node was populated with 6Gbytes of PC-3200 registered ECC RAM. For parity between results obtained using the cluster and single server, we used the same memory topology emulation scheme by logically partitioning physical memory into 128Mbyte logical DIMMs. We also used the same fixed delays for online (500ms) and offline (50ms) operations. The only difference between the emulated topologies was the memory capacity of the systems.

Servers used for scientific computational analysis often operate on large in-core data sets, requiring a static memory device set over a long time period. On a shared system with a batch scheduler, the memory usage of each of two nodes may be correlated for jobs in which both nodes are involved, and completely unrelated for intervals during which each node executes a different job.

Accordingly, we examine workloads from both categories. First, we examine the performance of Memory MISER under jobs that use the entire system, with the individual nodes allocating memory at the beginning of execution and freeing it upon termination. Specifically, we examine the energy savings during a run of the Parallel Ocean Program (POP). Second, we examine the performance of the system under workloads that also utilize the entire cluster, but allocate data according to a runtime pattern. Specifically, we examine the performance during a run of the FLASH AMR hydrodynamics code. Third, we examine the performance of the system while running independent jobs that utilize nodes in a way similar to a binary tree with synchronizations between scheduling. Specifically, we run random (yet repeatable) instances of the NAS parallel benchmarks, with a single program using 8 nodes, then 2 programs using 4 nodes each, followed by 4 programs using 2 nodes each, and finally 8 single-node instances, with a synchronization point in between each run. This creates results that can be easily correlated between nodes while still providing interesting and asymmetric results between nodes. Fourth, we examine the performance of the system using randomly scheduled workloads with random processor assignments. Specifically, we proceed as in the third workload except we eliminate the synchronization points and schedule the nodes as they become available. This emulates the workload of a batch-scheduled shared cluster; correlating data between nodes is correspondingly difficult. The processor configurations and memory footprints of each of the workloads is given in Table 4.2. These constitute the pool of codes from which the schedulers draw. We chose these codes and combinations for their similarity to workloads observed on the System X supercomputer at Virginia Tech (a cluster of servers) and to exhibit a wide range of memory demand to test the scalability, performance, and robustness of our techniques.

4.6.2.1 Parallel Result Evaluation

Similar to our single-server evaluation, we compare our dynamic control policy to the default control policy. We compute two metrics based on the default control policy. First, we consider

Table 4.2. Workload Configurations

Name		Processor configurations	Memory footprint in MB (max, per processor)
Static workloads	POP	8	1052
	FLASH	8	2906
Scheduled workloads	bt.C	1	1569
		4	429
	cg.C	1	1113
		2	567
		4	308
	ep.C	1	11
		2	11
		4	11
		8	11
	is.C	2	204
		4	114
		8	65
	lu.C	1	712
		2	106
		4	195
		8	40
	mg.C	2	1742
		4	891
	sp.C	1	1338
		4	114

program normalized (PN) energy savings. For the PN version of the default policy, we assume the maximum amount of memory used at any point in the program was powered on during code execution. We then normalize the energy used by our power-aware memory approach to the PN computed value. Basically, this compares our results to the energy that would be used if available physical memory matched maximum memory demand.

Second, we consider system normalized (SN) energy savings. For the SN version of the naïve case, we assume the maximum amount of available memory (3Gbytes per processor) is powered on during code execution. We then normalize the energy used by our power-aware memory approach to the SN computed value. Despite the fact that the additional physical memory seems "free," which would allow it to be put in a lower power state, in reality systems will allocate across all of the available devices. This lowers the probability of these devices being able to change state.

To convert these memory-centric results to system-wide energy savings, see Figure 4.1 to obtain the memory budget percentage for various memory and processor configurations and

Table 4.3. Cluster Energy Savings Summary

	% of mem power		% total system power (32GB/proc, 8 proc)	
Workload	**PN**	**SN**	**PN**	**SN**
Cluster-wide Static (POP)	0.10%	55.97%	0.04%	24.63%
Cluster-wide Dynamic (Flash)	9.33%	56.08%	4.11%	24.68%
Synch Scheduled	34.49%	67.54%	15.18%	29.72%
Unsynch Scheduled	32.29%	67.94%	14.21%	29.89%

multiply by the PN or SN savings. Table 4.3 provides a summary of PN, SN, and total systems energy savings for a system with 8 processors and 32Gbytes SDRAM per processor or a 44% memory power budget.

4.6.2.2 Cluster-wide Static Workloads

We first examine the performance of Memory MISER in cluster-wide workloads with static memory demand. Many scientific codes fall into this category. Specifically, we examine the performance of our system during a run of the Parallel Ocean Program (POP). This code computes ocean circulation according to 3-dimesional hydrodynamics on a sphere. It maps the surface of the earth to an orthogonal grid and parallelizes the grid across processors. Since the dimensions of the grid are known at runtime and the dimensions do not change over the course of the run, the primary memory footprint of the program is allocated at the beginning of the run and freed at the end of the run. Maximum memory allocations being equal, static allocation offers the least potential for energy savings compared to the other types of codes studied. Savings can still be achieved, however, if the memory demand of the code is less than the memory capacity of the system.

Figure 4.10 presents the results on a single processor from a run of POP on 8 processors. The other processors have identical profiles. If we normalize to the maximum memory used in

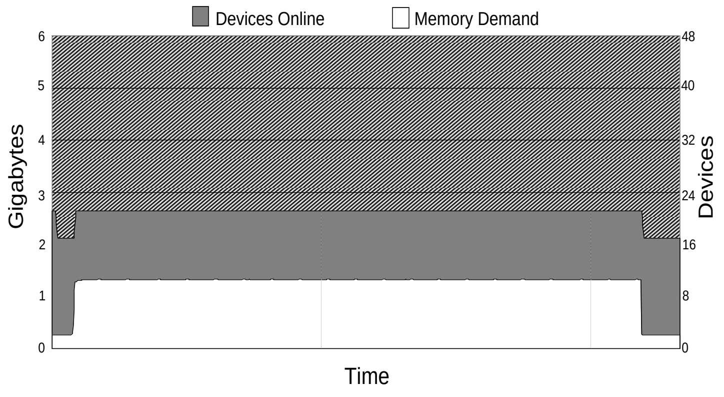


Figure 4.10. Results of Memory MISER for the POP code.

the run, roughly 1Gbyte, achievable savings are realized before the initial allocation and after the final deallocation, accounting for only 0.1% PN energy savings. However, if we examine the SN energy savings, we see that the system saves 55.97% energy. Many static workloads do not use the entire memory capacity of the system; therefore, even for static workloads Memory MISER can achieve considerable savings.

4.6.2.3 Cluster-wide Dynamic Workloads

While many scientific codes use a static allocation policy, others allocate data in a tiered or cyclic pattern during execution. These codes offer further potential for energy savings by minimizing online capacity during periods of low demand. We examine the performance of Memory MISER in the context of one such code, FLASH, a hydrodynamics code that utilizes adaptive mesh refinement (AMR). FLASH allocates data in a tiered pattern, increasing the memory footprint when necessary. Additionally, within each tier it dynamically allocates and frees memory cyclically with a high frequency.

Figure 4.11 presents the results from a run of FLASH for a single processor. Again, the other processors have identical profiles. The tiers and cycles mentioned above are highlighted,

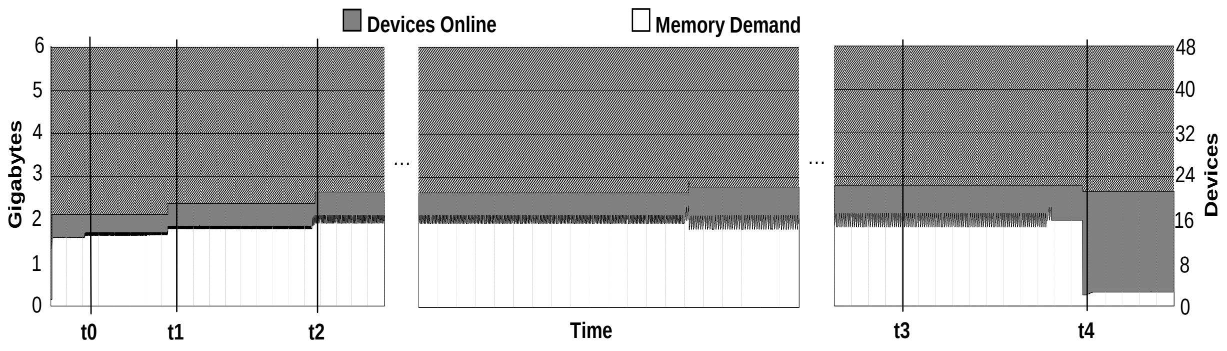


Figure 4.11. Results of Memory MISER during a run of the FLASH code.

with online memory climbing up to a maximum of around 2.9Gbytes from an initial set point of 2.1Gbytes. t0, t1, and t2 are the locations of the tiers. This difference over the course of the execution yields a PN energy savings of 9.33%, a significant improvement over the static case. It also gives a SN energy savings of 56.08%. This constitutes an improvement over the static case, even though maximum demand is higher. The slow offlining at point t4 is due to the long running time of the code; pages have gotten pinned, an artifact of OS page usage that makes migration difficult. Another point of interest is the lack of response to the high-frequency memory demand fluctuations between t2 and t4, even though the fluctuations are large enough at t3 to cross device boundaries. This is a feature of the conservative offlining techniques mentioned in Section 4.4.

4.6.2.4 Synchronized Scheduled Workloads

While single codes that utilize an entire cluster are interesting, even greater energy savings can be achieved in a shared system, where the workloads of individual nodes can be independent. First we consider the case where the system is synchronized between scheduled jobs. This is similar to a pipelined, coupled model, where the next code in the pipeline is dependent on the

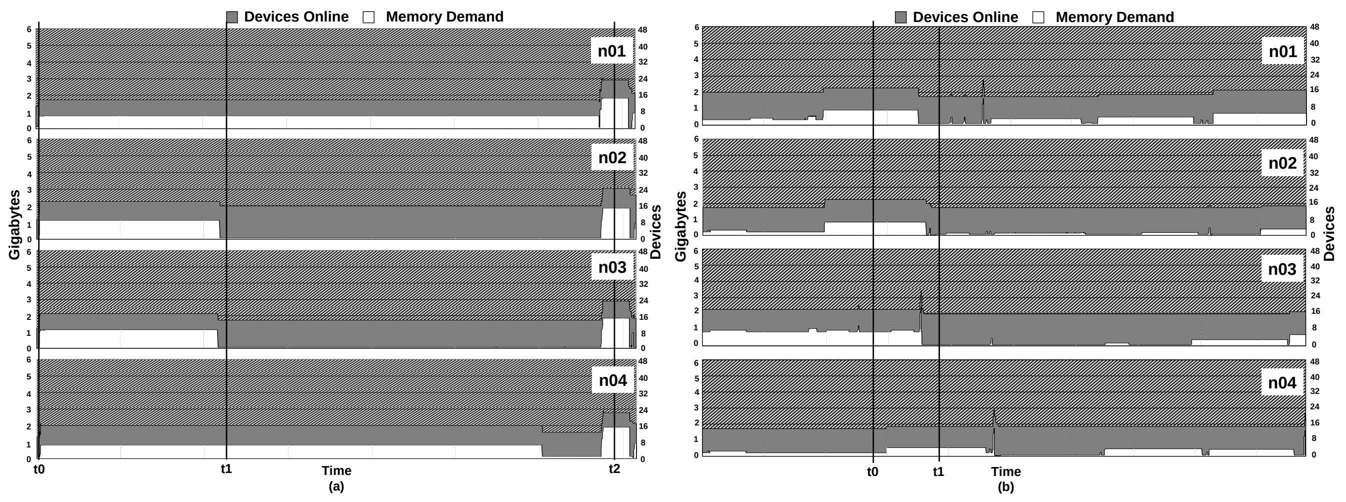


Figure 4.12. Results of Memory MISER during a run of the (a) synchronized scheduler and (b) unsynchronized scheduler.

results from the previous code. It is also a single cluster-wide job, using the NAS Parallel benchmarks, as given by Table 4.2. Initially after scheduling a level of the tree, all processors are busy; since the codes are selected randomly, though, the processors may finish at different times. The system waits until the entire processor pool is free before scheduling the next set of jobs. similar to the case where a system-wide job has to be scheduled while other jobs are running on smaller processor configurations. We created a tree-like synthetic scheduler that cyclically schedules jobs from 8 single-node jobs down to a single cluster-wide job, using the NAS benchmarks, as given by Table 4.2. Initially, after scheduling a level of the tree, all of the processors are busy; since the codes are selected randomly, though, the processors may finish at different times. The system waits until the entire processor pool is free before scheduling the next set of jobs.

Figure 4.12(a) presents the results from a run of the synchronized scheduler for 4 processors with important times highlighted. At time t0, the first set of jobs gets scheduled and demand increases accordingly. At time t1, nodes n02 and n03 have finished, and online memory capacity is reduced as they wait to synchronize.

At time t2, synchronization has completed and the next set of jobs has been scheduled. The synchronization offers potential for normalized energy savings; the run depicted in Figure 4.12(a) achieved PN energy savings of 34.49% cluster-wide, with per-processor PN energy savings of up to 70.23%. Accordingly, the SN energy savings are larger, 67.54% cluster-wide.

4.6.2.5 Unsynchronized Scheduled Workloads

By relaxing the condition that the nodes must be synchronized between different code distribution and execution, we arrive at a system very similar to a multi-user scheduled cluster. This reflects the distribution of jobs on many large-scale systems. The system is partitioned into several smaller logical clusters; users submit jobs to an appropriate queue based on processor configuration and runtime. These jobs are then scheduled on the appropriate logical cluster by a scheduling system like PBS. The random nature and imbalance of the jobs on this type of system offers a high probability for slack in memory demand. Memory MISER converts this slack into energy savings. We use a synthetic scheduler similar to the one described previously, except we remove the synchronization requirements and do not constrain the number of processors for a given job. Jobs are chosen from the pool of codes presented in Table 4.2 and scheduled randomly.

Figure 4.12(b) presents the results from a portion of a run of this random scheduler for 4 processors. The memory demand for each processor is highly featured; these fluctuations in demand allow devices to be offlined. As above, interesting time points have been highlighted. At time t0, nodes n01 and n02 are engaged in a 2-processor memory intensive application, node n03 is engaged in a long-term, memory demand-intensive application, but node n04 is engaged in a computation-bound low memory demand application. It can therefore reduce memory capacity to save energy. At time t1, the situation is the opposite: nodes n01, n02, n03 have completed their memory-intensive applications and started low-memory demand codes, while n04 has begun a new memory demand-intensive application. The online capacities follow this trend perfectly; these are the types of imbalances in scheduled workloads that Memory MISER exploits for energy savings.

The cluster-wide PN energy savings for this application are large, 32.29%, with per-node PN energy savings as low as 18.81% and as high as 39.02%. The cluster-wide SN energy savings are also large, 67.94%. Other points of interest are the sharp spikes near the middle of the execution for 3 of the processors. These rapid jumps in memory demand are the reason for the conservative use of the w parameter; without this, the system would likely have to page to disk, resulting in severe performance loss. By increasing w, no performance loss is incurred.

4.7 Chapter Summary

We have quantified the performance and energy for serial and parallel workloads on several machine configurations and shown that there are significant opportunities to minimize energy consumption on large systems due to variable slack in memory demand. By offlining memory devices using an adaptive PID controller at runtime, we achieved as high as 70% energy savings.

Despite these savings, this chapter has evaluated our control system on sequential memory systems. Modern server systems typically use advanced architectural techniques to exploit inherent parallelism within high-capacity memory systems for performance. Based on this evaluation, it's unclear whether our control system can reduce power and energy on machines with dense, interleaved memory topologies. More generally, it is unclear how interleaving affects memory power and heat production. Given the complexities of interleaving, it is unclear whether our approaches can be scaled to interleaved memory systems. .

While the control-theoretic runtime system proposed in this chapter improves the energy efficiency of sequential memory systems, to be generally useful for high-performance server memory architectures our techniques must consider complex, highly-interleaved memory architectures.

Chapter 5

Evaluating the Power, Performance, and Thermal Efficiency of Interleaved Memory Systems

Memory interleaving, the distribution of data across multiple DRAM chips, is widely accepted as a technique that improves the performance of systems and applications by increasing the bandwidth between processors and main memory. Multi-core architectures demand greater memory bandwidth since the number of data transfers to and from main memory can increase dramatically with an increase in cores per die. For these reasons, traditional memory designs assume a performance-driven, “more is better” approach to memory interleaving design which results in memory architectures with many levels of interleaving (i.e. high dimensionality) designed primarily to increase peak bandwidth.

In this chapter, we demonstrate why the performance-driven, “more is better” interleaved memory design assumption is problematic and should be revisited. Our results indicate that for bandwidth-sensitive benchmarks such as STREAM memory interleaving in a single dimension yields up to 35% average bandwidth improvement and reduces energy consumption by 13%. However, this improved bandwidth results in higher memory device access frequency which results in a 25% increase in memory temperature. For the same benchmarks, further increases in interleaving dimensionality result in little to no performance or energy efficiency gains but still increase temperature nearly 25%. For other benchmarks less sensitive to memory bandwidth, we found interleaving dimensionality often does not significantly improve bandwidth or energy efficiency, but still increase nearly 25%. The additional heat resulting from higher operating temperatures must be exhausted from the system chassis. This increases the need for additional cooling, elevates the cost of the system and, in the case of powered cooling, can increase chassis energy consumption. We conclude that the impact of interleaving on energy and heat production must be considered in future memory designs.

5.1 Introduction

To meet the growing throughput demands of Chip Multiprocessor (CMP) systems, highly interleaved memory topologies have been designed to increase the parallelism within the memory system. These memory systems interleave cache lines across DRAM banks, but also across sets of DRAM banks (e.g. ranks), DIMMs, channels, and memory controllers. Such interleaving is used to reduce access conflicts such as row buffer conflicts within DRAMs, minimize the impact of rank turn-around time penalties, enable concurrent device accesses, and improve memory bus utilization [95, 100, 137, 142, 170]. Reducing memory access latencies translates into lower processor stall latencies, thereby improving application performance.

The traditional approach of simply using an interleaving configuration designed for a theoretical maximum throughput may not be power or thermally efficient for all workloads. For example, previous work to improve processor power-efficiency has shown that for many workloads, using clock frequencies lower than the maximum often does not significantly impact performance, but can save significant energy. In contrast, the power, performance, and thermal impact of different types of interleaving in current server memory systems are not well understood. Consequently, it is difficult to reason about the pros and cons of different interleaving dimensions and schemes in future memory systems.

There has been significant previous work to reduce memory power consumption [42-45, 48, 57-59, 68, 87-90, 100, 113, 117, 119], which has been shown to yield significant energy savings. However, most of this work has focused on exploiting intermediate dynamic random access memory (DRAM) power states and have largely ignored the effects of interleaving. Interleaving at DRAM bank granularity has been well studied [100, 142, 170] and incorporated in numerous commercial memory controllers [95, 137]. However, prior work has not addressed interleaving at other granularities despite the widespread use of these techniques in modern server systems. As far as we are aware, there has not been significant previous work to contrast the power or thermal efficiency with the bandwidth advantages of memory interleaving; nor has there been work to identify the impact of varying the dimensionality of interleaving.

In this chapter, we revisit memory interleaving to determine the energy and thermal efficiency of the traditional, performance-driven practice of maximizing interleaving for increased peak theoretical bandwidth within the memory system. We review the underlying mechanics of complex interleaving schemes, identify and isolate the effects of the different interleaving dimensions on application performance, energy, and heat, and compare the results to determine which yield the best efficiency. To sample a wide-spectrum of application types, we analyze the power, performance, and thermal impact of different interleaving levels using the STREAM, SPEC CPU2006, and the NAS benchmarks. Based our evaluation, this chapter offers the following contributions:

- We show that for memory bandwidth sensitive applications, interleaving yields up to 35% improvement in bandwidth and a 13% reduction in memory energy, but increases mean memory system temperature by 25%.

- Increasing available bandwidth for bandwidth-insensitive applications does not automatically improve performance. We found that although the peak achievable bandwidth was up to 35% higher, application bandwidth for several SPEC CPU2006 benchmarks remained within 3% - 4% of sequential memory systems.
- Interleaving yields up to 56.5% *EDP* improvement and up to 30% higher memory system temperatures for the NAS benchmark suite. The increased bandwidth decreases per-application memory energy costs by reducing execution time, however, the resultant heat reduces reliability.
- Increasing the dimensional complexity of interleaving improved bandwidth or energy consumption less than 2%, while consistently increasing temperature by 25% in our measurements. For the NAS benchmarks, the most complex multi-dimensional interleaving scheme yielded up to 33% EDP improvement normalized compared to 56.5% EDP improvement for an interleaved configuration of lower dimensionality.
- The efficiency of interleaving is highly dependent on application-specific memory access characteristics. For the SPEC CPU2006 codes, interleaving improved EDP by up to 45.7% for some codes, while decreasing EDP up to 60% for others.

5.2 Evaluation Methodology

Evaluating the effects of varying interleaving dimensionality on application power, performance, and heat dissipation is challenging. First, it is often difficult to discern how cache lines are interleaved within the memory system. Even though the configuration registers are often available and accessible in memory mapped I/O space, memory controllers are often not well documented. This makes simulations difficult to implement and validate. Second, using real systems alleviates some of the simulation challenges but does not permit the specification or tuning of interleaving parameters. Once memory devices are detected, firmware initializes memory controllers using a predetermined interleave configuration. So to evaluate different mapping configurations on a real system, the memory controller initialization sequence must be modified for different mapping schemes.

We used Intel and AMD dual-processor (DP) servers that include controllers with interleaving capabilities designed to maximize bandwidth for multi-core processors to evaluate the power, performance, and thermal efficiency of different memory interleaving configurations. We elected to use real systems instead of simulation to gain empirical power, thermal, and performance insight into current, state-of-the-art memory interleaving schemes. Rather than extrapolating dynamic power estimates from DRAM datasheets, tracking and correlating DRAM accesses during application execution, and having to validate our aggregate estimates are within expected real estimates, using real systems enabled us to collect actual memory power rail measurements. Additionally, our temperature measurements are not based on resistance-capacitance temperature models, but actual thermal readings collected via thermocouple probes as well as diodes embedded in silicon components. While simulation would have enabled us to exhaustively explore the efficiency of myriad alternative interleaving schemes on unorthodox topologies, we wanted our baseline efficiency work to be rooted in actual system measurements. Still, we plan further investigation of unorthodox and emergent memory systems through simulation in future work.

We chose DP servers instead of larger systems with higher-density memory systems for several reasons. First, a 2007 survey funded by AMD revealed the power consumption of volume and mid-range servers increased 16% and 51% respectively from 2000 to 2005 [110]. Since these types of servers account for 99% of the more than 1.5 million servers sold in the

United States annually, the energy efficiency of this class of systems is critical to operational costs. Second, although the capacity of DP servers is generally limited compared to larger multiprocessor systems, the memory systems in DP servers often incorporate multiple interleaving options.

5.2.1 Interleaving Configurations

On many recent processors and chipsets, DRAM bank-interleaving is supported by default and may not be disabled. The AMD Opteron 265 processor, which includes an integrated memory controller, supports bank and node interleaving. We found that channel interleaving is supported in recently released Intel i7 processors, which also include an integrated, on-die memory controller; however, these were not commercially available at the time of our evaluation. The Intel 5000 chipset includes several degrees of rank interleaving as well as branch interleaving

To determine the impact of interleaving power and performance, we iterated through the available interleaving schemes in each system in isolation. On the AMD system, we evaluated bank interleaving. On the Intel system, 3 rank interleave configurations and 2 branch interleave configurations were supported for a total of 6 configurations. However, since we had limited access to dual-rank DIMMs, we evaluated all codes on 4 of the 6 available combinations using single-rank 512MB DIMMs on the Intel system. Our evaluations of the 2 remaining interleave configurations that included 4:1 rank interleaving was limited to STREAM and select SPEC benchmarks.

5.2.2 Power Measurements

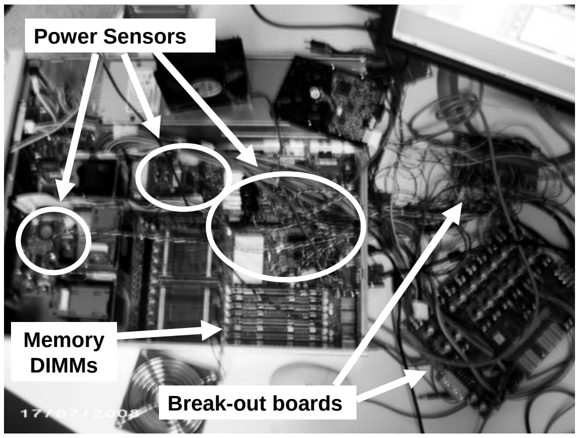


Figure 5.1. Aerial view of evaluation system showing leads and breakout boards for monitoring component power consumption. The logic analyzer (not pictured) is connected to the breakout board and the host system.

To determine the energy impact of different interleaving schemes we measured memory system power for each benchmark. We instrumented the voltage regulators on the system board with power taps as shown in Figure 5.1. We hooked up the power leads to a break out board, which we then connected to an Agilent Data Acquisition Meter and Logic Analyzer. We used National Instruments LabView software to collect power measurements for each of the power rails, including the memory controller and DIMMs.. This enabled us to compare the power impact of the different interleaving dimensions in isolation. Thus, our energy measurements are not affected by other devices such as CPUs, I/O devices, etc.

5.2.3 Thermal Measurements

To measure the thermal impact of interleaving, we attached thermocouple probes to individual DIMMs. On our AMD system we connected thermocouple probes to Berkeley mote-based wireless environmental sensors, which we then polled periodically. Since two of our evaluation systems used fully-buffered DIMMs, we also developed a custom OS device driver to read the embedded thermal diodes within the Advanced Memory Buffer (AMB) on each of the FBDIMMs. This enabled us to monitor the temperature of all DIMMs within the system in real-time while running workloads. We verified the temperatures reported by the thermal diodes were consistent with temperatures we collected using thermocouple probes with a Fluke NETDAQ analyzer. Our driver exported the temperatures of all DIMMs via the Linux sysfs interface. This enabled our test harness to simply poll the temperature of each DIMM at 1 second granularity while each benchmark executed.

5.2.4 Workloads

To account for varied memory utilization loads, we evaluated several benchmarks across the interleaving schemes supported by each of our test systems. We used codes from the SPEC CPU2006 suite as well as STREAM. We also used the class C versions of codes from the NAS 3.0 suite. We built the SPEC CPU2006 and NAS codes using gcc version 4.1.2 using –O3 optimizations. For the base operating system, we used the 2.6.25 kernel rebuilt with support for hardware performance counters.

5.2.5 Experiments

For each of the supported memory configurations, we ran all of the benchmark codes in isolation 5 times. For each code, we recorded memory power and thermal measurements. During all experiments, the ambient air temperature of the laboratory was near constant and the air flow across the DIMMs was held constant to ensure temperature fluctuations were only related to the interleaving configuration. After each run, the performance results for each code were archived along with the power and temperature measurements. This enabled us to analyze the effects of each interleaving dimension and configuration for each code in isolation. All of the results reported in this paper are normalized to the sequential configuration.

5.3 Experimental Results

5.3.1 AMD Server Results

5.3.1.1 System Details

In our initial evaluation, we used a dual-processor, AMD Opteron system with 8Gbytes of single-rank PC-3200 registered ECC DDR DRAM. Figure 5.2 shows the layout of the memory system for this platform. Since each Opteron processor has an integrated memory controller, the system has two dual-channel DDR memory controllers each connected to 4 DIMMs. The system supported two interleaving options configurable through BIOS options: bank interleaving and node interleaving. We used a NUMA-aware 2.6.25 Linux kernel, and enabled the node interleaving option within the BIOS to maximize performance on each node. As previously discussed, we did not evaluate nodal interleaving, limiting our evaluation on this system to bank interleaving. When identical DIMMs are used in adjacent slots, the two 64-bit channels can be logically combined to form a single 128-bit channel. Thus the four DIMMs in our system

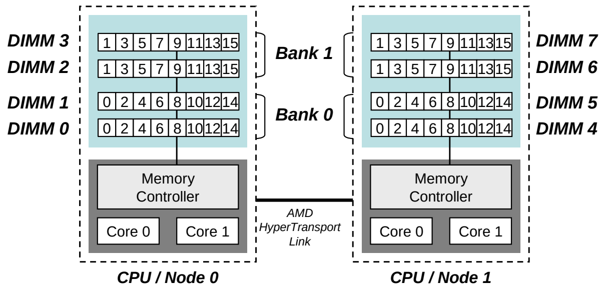


Figure 5.2. Block diagram illustrating bank-level interleaving on the AMD system.

constitute two banks. In Figure 5.2, DIMM 0 and DIMM 1 are paired to form a single bank and DIMM 2 and DIMM3 are paired to form a second bank. When bank interleaving is enabled, the controller interleaves accesses across these two logical banks. Our experimental results evaluate the performance impact of enabling and disabling this form of bank interleaving across benchmarks.

5.3.1.2 Results

We ran the STREAM, SPEC CPU2006, and NAS codes on the AMD machine while measuring memory power and temperature. We used the STREAM benchmark to measure bandwidth configured STREAM to use 2Gbytes of memory to minimize cache effects. We ran all codes on the sequential and bank-interleaved configurations 5 times and calculated the mean bandwidth and performance. For the bandwidth and performance results presented here, the coefficient of variance was less than 0.0612. Given the physical distance between the memory controllers and DIMM sets, our temperature results only compare the DIMM temperatures for the memory directly attached to CPU socket 0. Although we measured the temperature of the memory directly attached to CPU socket 1, we found there was only minimal variance.

On the AMD system, bank interleaving yielded a 7.9% increase in bandwidth, an 8.3% decrease in energy consumption, and less than 1% difference in temperature using STREAM. The improvement was due to the increased channel width. Using the SPEC CPU2006 codes we observed performance improvements for several codes, including lbm (10.4%), GemsFDTD (9.6%), and milc (5.7%). The performance of other codes such as bzip2, gcc, and libquantum improved less than 5%, while other bandwidth-insensitive codes exhibited less than 2% variance. However, we observed that the increased channel width resulted in fewer L2 cache misses reducing memory energy. Temperature differentials caused by using bank interleaving were minimal. For most codes, mean memory system temperature was within 5%. The only form of interleaving available in this system did not have a significant impact on application performance, memory energy or heat.

5.3.2 Intel Server Results

We also used a recent Intel Xeon DP server in our evaluation. This system contained two Dual-core Xeon processors coupled with the Intel 5000 chipset [137] via multiple dedicated front-side bus connections as shown in Figure 5.3. This system supports up to eight fully-buffered memory DIMMs across two memory branches within the memory controller hub; each branch has two channels for a total of four channels, although the channels of each branch operate in lockstep. DRAM bank interleaving was not a configurable option within the BIOS for this memory controller; rather it is always enabled. Finally, the memory controller uses a closed-page mode DRAM page policy.

5.3.2.1 System Details

Branch Interleaving: As discussed in Chapter 2, branch interleaving distributes the physical address space across both memory controller branches. In an interleaved configuration even cache lines are mapped to one branch and odd cache lines to the second branch as illustrated in Figure 5.3. In a sequential configuration, cache lines are sequentially mapped to branch 0 up to the capacity of memory residing behind branch 0; the cache lines that constitute the remaining physical address space are mapped to branch 1.

Rank Interleaving: Within each branch the two channels operate in lockstep such that cache lines are mapped to two physical DIMM ranks on separate channels, which collectively form a

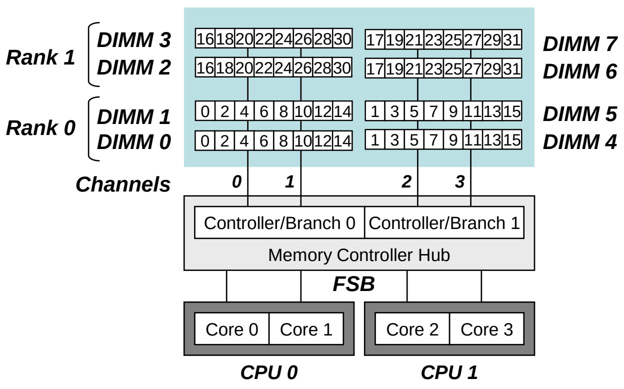


Figure 5.3. Block diagram illustrating branch interleaving with rank sequential (rank 1:1) mapping on Intel Xeon platform.

logical rank. In Figure 5.3, DIMM 0 resides on channel 0 and DIMM 1 resides on channel 1. These two single-rank DIMMs constitute a logical rank from the perspective of the memory controller.

Rank interleaving distributes cache lines within the address space of each branch across logical ranks. Three rank-level mapping schemes are supported by the memory controller: 1:1, 2:1, and 4:1. Figure 5.3 illustrates the 1:1 mapping with single rank DIMMs. Since 1:1 is rank sequential, cache lines 0 to 14 (which are contiguous within the branch address space) are mapped to logical rank 0. Subsequent cache lines are mapped to the next logical rank, rank 1, composed of physical DIMMs 2 and 3. Given there are two logical ranks in Figure 5.3, the memory controller could also be set up to use a 2:1 rank interleave scheme. In this case, a 2:1 rank interleaving scheme would map branch-contiguous cache lines (0, 2, 4, etc.) across the two logical ranks such that cache line 0 would reside on rank 0, cache line 2 would reside on rank 1, cache line 4 would map to rank 1, and so on.

Using DIMMs with multiple physical ranks can change the rank interleaving mapping. For example, using single-rank DIMMs as in Figure 5.3, the maximum rank interleave scheme supported is 2:1. A 4:1 rank interleave scheme can not be used since there are only two logical ranks. However, replacing the single-rank DIMMs with dual-rank yields four logical ranks. Figure 5.3 illustrate how DIMM 0 rank 0 would be paired with DIMM 1 rank 0, DIMM 0 rank 1 would be paired with DIMM 1 rank 1, DIMM 2 rank 0 would be paired with DIMM3 rank 0, and DIMM 2 rank 1 would be paired with DIMM 3 rank 1, resulting in 4 logical ranks. A 4:1 rank interleave scheme maps four contiguous cache lines within the branch address space across these 4 logical ranks. However, directly mapping cache line 0 to rank 0, cache line 1 to rank 1, and so on is suboptimal since this would incur an additional rank-to-rank turnaround time penalty. To mitigate this latency, the 4:1 rank interleave algorithm maps branch-contiguous cache lines amongst the logical ranks that are not physically located on ranks of the same physical DIMMs. For example, rather than mapping contiguous cache lines 0, 1, 2, 3, to logical ranks 0, 1, 2, 3, the 4:1 interleave would be achieved by mapping contiguous cache lines 0, 1, 2, 3 to logical ranks 0, 2, 1, 3.

System Configuration: We found the production BIOS automatically configured the memory system to always use the maximum rank interleaving possible based on detected ranks and to always interleave across the two branches. We modified the BIOS to allow user specification of the interleaving configurations used by the memory controller. Additionally, the system board was instrumented with voltage regulator power taps through extensive rework. Although the number of DIMM slots in this system was limited to two per memory channel, we were able to collect limited results using the 4:1 rank interleave scheme by interleaving across the ranks of dual-rank DIMMs. However, the majority of the results presented here were collected using 8

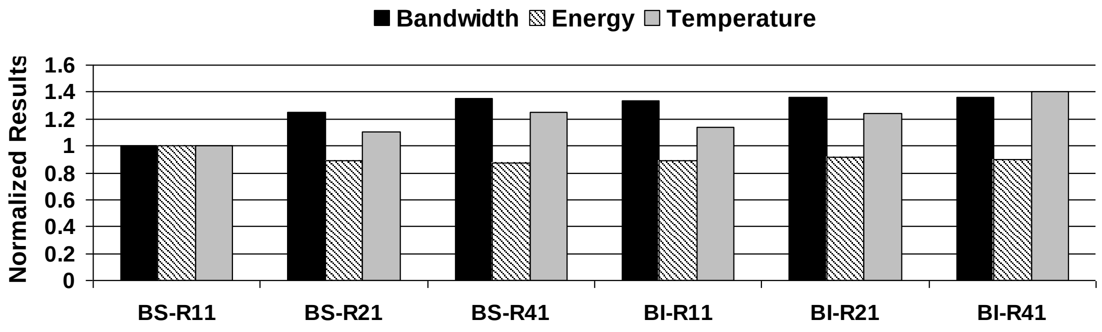


Figure 5.4. STREAM memory bandwidth, energy, and mean temperature comparison of interleaved memory configurations normalized to BS-R11.

single-rank 512 MB DIMMs, where the maximum supported rank interleave was 2:1. The results presented in this section compare power, performance, and temperature for the four configurations using the following shorthand: Branch-Sequential with 1:1 Rank (BS-R11), Branch Sequential with 2:1 Rank (BS-R21), Branch-Sequential with 4:1 Rank (BS-R41), Branch-Interleave with 1:1 Rank (BI-R11), Branch-Interleave with 2:1 Rank Interleave (BI-R21), Branch-Interleave with 4:1 Rank Interleave (BI-R41).

5.3.2.2 Memory Bandwidth

We used STREAM to measure memory system bandwidth for all of the memory configurations. Using performance monitors, we observed L2 cache hits were limited to 9% during each STREAM run. We ran STREAM on each memory configuration 10 times and calculated the mean bandwidth across the four tests for each run. For all STREAM results, the coefficient of variance was less than 0.002.

Figure 5.4 compares mean memory bandwidth as well as energy and mean temperature for all six memory configurations normalized to Branch-Sequential with 1:1 Rank Interleave (BS-R11). BS-R11 maps the lower and upper halves of the address space to branches 0 and 1 respectively. Until memory is allocated by the operating system that spans the physical address space, accesses are serialized to a single branch. Within each branch, contiguous cache lines are mapped to the same rank up to capacity boundaries. This increases DRAM bank row-buffer conflicts within each rank, incurring activation penalties on successive accesses, although precharge penalties are amortized due to using closed-page mode. Collectively, these delays increase access latency and degrade bandwidth. BS-R21 increases the degree of rank interleaving from 1:1 to 2:1 yielding an average of 24.8% higher memory bandwidth, 11.1% reduction in energy, and 10.3% higher mean temperature than BS-R11. Similar to BS-R11, the address space is still divided across the two branches, and STREAM accesses are primarily mapped to a single branch. However, branch-contiguous cache lines are mapped across ranks within each branch, which may be accessed in parallel. BS-R41 exploits the two physical ranks on each DIMM yielding a 25.15% increase in bandwidth, 13% reduction in energy, at a cost of 25% higher memory system temperature. BI-R11 shows that interleaving across branches and not interleaving at rank granularity yields an average of 33.8% higher bandwidth and 10.7% reduction in energy, which increases mean temperature by 13.7% relative to BS-R11. Notably, using Scale on BI-R11 yielded a 45.12% bandwidth improvement over BS-R11 and 20.6% improvement over BS-R21. BI-R21 combines branch interleaving with 2:1 rank interleaving yielding an average 35.85% increase in bandwidth and an 8.8% decrease in energy consumption, but increased mean temperature by 24.13% over BI-R11. Interleaving across both branches and interleaving across ranks all four ranks led to a 35.7% bandwidth improvement and 10.3% reduction in energy, but increased mean memory system temperature by 40% relative to BS-R11. Figure 5.5 shows the STREAM results isolated by effect. Figure 5.5(a) shows the bandwidth, energy, and temperature impact of varying rank interleaving from 1:1 to 2:1 to 4:1 in Branch

Sequential configurations (BS-R11, BS-R21, BS-R41). This shows that increasing rank interleaving in memory systems with a single controller (without coarse-grained controller-granularity interleaving) can provide a significant bandwidth improvement for bandwidth-sensitive codes. However, for the dual-branch controller in our test system, this resulted in limited utilization of the second branch until external fragmentation occurred within the OS memory allocator. In contrast, Figure 5.5(b) shows the bandwidth, energy, and temperature impact of varying rank interleaving in Branch Interleave configurations (BI-R11, BI-R21, BI-R41). In contrast to Figure 5.5(b), increasing rank-interleaving in branch-interleaved configurations only provides up to 1.5% improvement in bandwidth while mean temperature increased by nearly 25% compared to BI-R11. This shows that systems using coarse-granularity interleaving do not necessarily benefit from lower-level interleaving. Unlike Figures 5.5(a) and 5.5(b), where branch interleaving was held constant while varying rank interleaving, figure 5.5(c) shows the effect of varying branch interleaving for each degree of rank-interleaving. In this graph, the data points show bandwidth, energy, and temperature for the three branch-interleaved configurations (BI-R11, BI-R21, and BI-R41) normalized to branch-sequential configurations (BS-R11, BS-R21, BS-R41). Increasing both branch and rank interleaving in concert reduces memory bandwidth and increases energy, while reducing mean memory temperature by 2.1% (albeit 11.6% higher than the Branch-Sequential configuration). On closer inspection, we found the reduction in temperature was caused by the spatial distribution of memory accesses across the memory system resultant to the BI-R41 scheme.

5.3.2.3 SPEC CPU2006

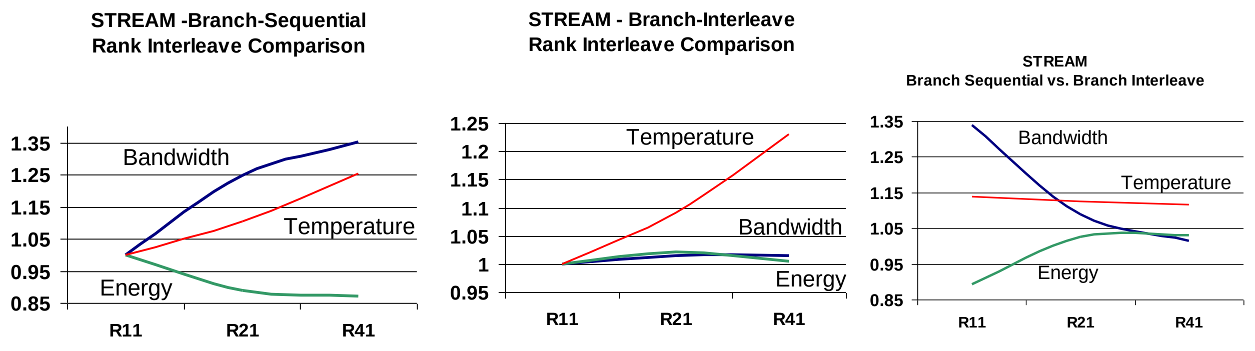
We also ran SPEC CPU2006 integer and floating point benchmarks to further characterize the performance, energy, and thermal impact of the four memory configurations. We did not run all of the SPEC codes on BS-R41 and BI-R21 due to limited memory device availability. The highest coefficient of variance for performance and energy across all of the codes was 0.1087 and 0.1521 respectively.

Performance Sensitivity: As shown in Figure 5.6(a), the effects of the interleaving schemes on benchmark performance varied across the different SPEC codes. Several benchmarks realized improvements in time-to-completion due to interleaving relative to BS-R11, including milc (9.1%), mcf (9.7%), bwaves (10.6%), libquantum (21.1%), gobmk (24.6%), hmmer (33.4%). There were also differences between the different levels of interleaving. For example, the time-to-completion of the gene sequence database search benchmark hmmer was reduced 33.4% using BI-R21 relative to BS-R11, but was also reduced by 32.8% using BS-R22. Similarly, using BI-R21 yielded a 23.9% reduction in time-to-completion relative to BS-R11 using the gobmk benchmark, while BI-R11 improved execution time by 24.6%. Thus, increasing interleaving dimensionality (in this case BI-R21) did not always provide the best performance improvement. Rather, similar to STREAM, interleaving at one level (branch) while using a sequential mapping at another level (rank) improved performance.

Several benchmarks were adversely impacted by interleaving. For example, the computational fluid dynamics benchmark lbm took 10.4% longer to complete for BS-R21, 6.05% longer for BI-R11, and 9.46% longer for BI-R21 relative to BS-R11. We also observed degradations for omnetpp (-3.7%), astar (-6.03%), xalancbmk (-2.8%), zeusmp (-3.83%), cactusADM (-3.47%), and Gems-FDTD (-2.8%).

Many of the computation-bound SPEC codes have a limited working set size. Larger on-chip caches reduce L2 misses which reduces application performance sensitivity to interleaving. Consequently, we found that games (0.7%), gromacs (0.006%), namd (0.005%), calculix (0.0001%), and tonto (0.005%) exhibited less than 1% variance across the four interleaving schemes. Given the bandwidth improvements observed using STREAM, we expected the performance of the SPEC codes to improve as the level of interleaving was increased within the memory system. While this was demonstrated by several codes, most did not substantially benefit from the increased memory bandwidth in isolation.

Energy: Figure 5.6(b) shows memory system EDP for the SPEC benchmarks. The EDP results largely align with the performance results. For example, the performance improvements using gobmk, hmmer, sjeng, and libquantum led to significant EDP improvements. However, the EDP of these codes was also affected by the interleaving configuration. For example, BI-R11 yielded a 33% performance improvement for the hmmer benchmark, which contributed to a 45.7% EDP improvement relative to BS-R11. Increasing the degree of interleaving led to higher EDP (e.g. worse power-performance efficiency) for many of the computation-bound codes. Such codes did not benefit from the increased available bandwidth made available by interleaving. So the additional power required to activate more DIMMs more frequently was wasted, leading to higher per-code energy costs. Unless this higher energy cost is offset by reduced execution time, we observe that interleaving can decrease the energy efficiency of compute-intensive applications within the memory system as measured by EDP. Figure 5.6(b) shows that sequential memory systems can be up to 20% more efficient than highly interleaved systems for applications that do not exploit memory bandwidth; and in some cases, such as tonto, up to 60% more efficient.

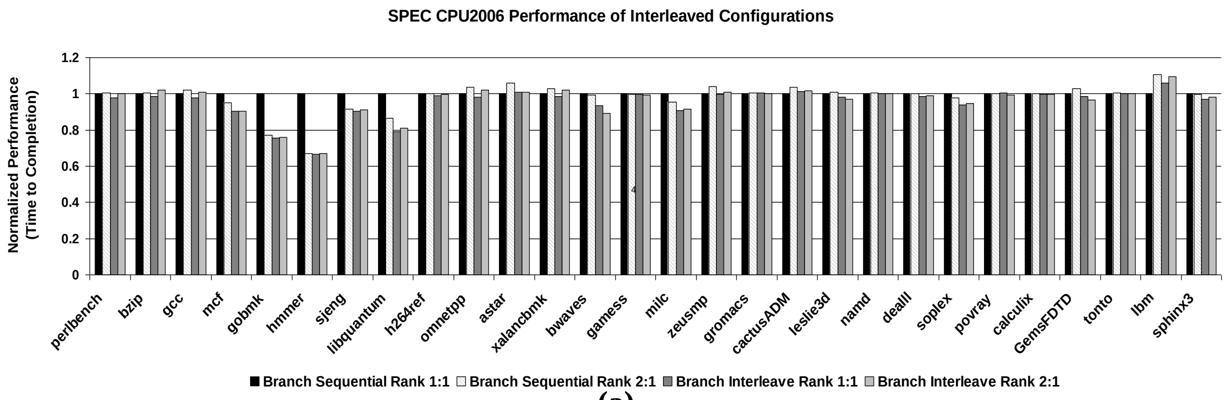


(a) (b) (c)

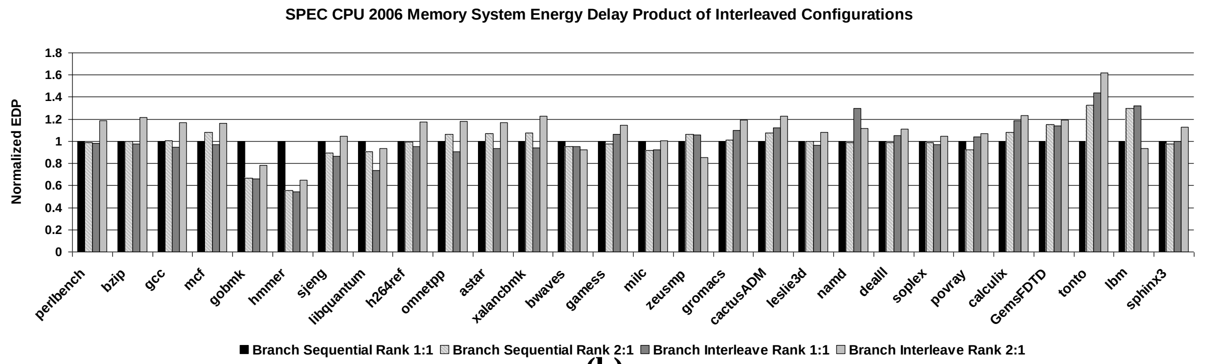
Figure 5.5. Memory bandwidth, energy, and mean temperature of STREAM showing (a) rank interleaving normalized to rank 1:1 under Branch Sequential, (b) rank interleaving normalized to rank 1:1 under Branch Interleave, and (c) Branch-Interleave normalized to Branch-Sequential for each of the rank interleave configurations.

Thermal Impact: Figure 5.6(c) shows the mean DIMM temperatures running the SPEC benchmarks on all four memory system configurations. The temperatures of the BI-R11 and BI-R21 configurations were 17% to 31% higher than the Branch Sequential configurations, while the mean temperature using BS-R21 was 2% to 15% lower. We attribute the higher mean temperatures of BI-R11 and BI-R21 to two factors. First, the distribution of transactions across the memory system increased the activity load, and hence thermal load, on DIMMs behind both branches in the system. Second, elevated temperatures across all DIMMs increased the overall thermal capacitance of the memory system.

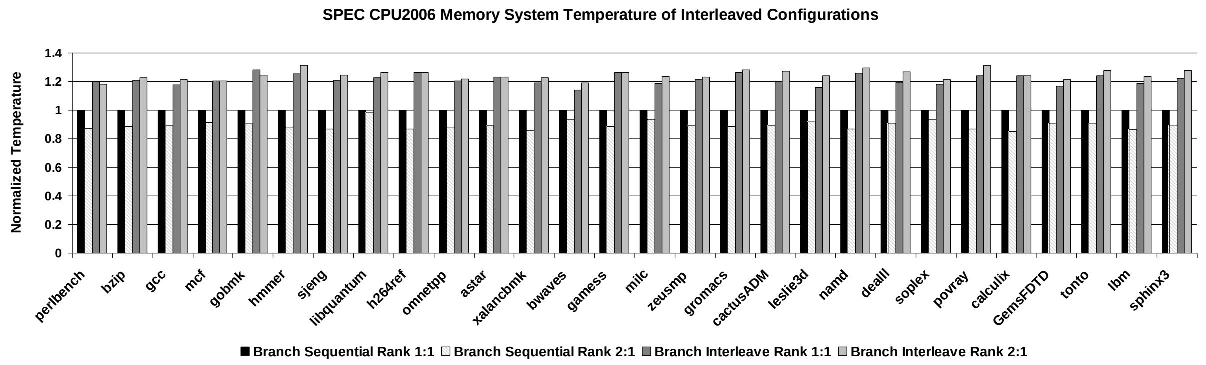
In contrast, the Branch Sequential configurations resulted in lower mean system temperatures, but created hot spots within the memory system. The maximum observed temperature difference between DIMM pairs operating in lockstep was 24°C using BS-R11. Although the temperature difference using BS-R21 was lower since transactions were spread across ranks within each branch, we still observed up to 12°C difference between DIMMs of each branch. In contrast, we found that the temperature difference between DIMMs for BI-R11



(a)



(b)



(c)

Figure 5.6. (a) Time-to-completion, (b) EDP, and (c) memory system temperature results of interleaved memory configurations normalized to the sequential configuration for the SPEC CPU2006 codes.

was within 6°C and 4°C for BI-R21. Consequently, interleaving distributed the thermal load was spread more evenly across the memory system.

5.3.2.4 NAS Serial Benchmarks

We ran each NAS code on each memory configuration 10 times. The highest coefficient of variance for performance and energy across all of the codes was 0.1201 and 0.1508 respectively.

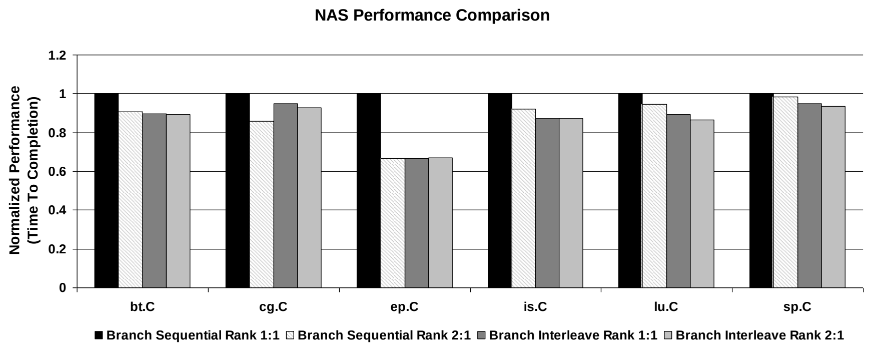
Performance Sensitivity: Interleaving consistently improved the performance of the NAS benchmarks. Figure 5.7(a) shows the NAS code performance of the interleaved memory configurations normalized to BS-R11. Relative to the sequential configuration, time-to-completion for `bt` was reduced by 9.45% using BS-R21 while BI-R11 and BI-R21 yielded 10.6% and 10.7% improvements respectively. Increasing rank interleaving within the sequentially mapped branches improved the performance of the `cg` code by 14.2%, while branch interleaving yields up to a 7.3% improvement relative to BS-R11. The `ep` code realized the highest performance improvement (33%) of the NAS codes we tested. The time-to-completion of the other NAS codes, `is`, `lu`, and `sp`, decreased as the degree of interleaving was increased. The increased performance realized across all of the NAS benchmarks demonstrates the benefits of increased parallelism within the memory system for highly parallelized applications.

Energy: Figure 5.7(b) compares memory system EDP for the interleaved memory configurations normalized to BS-R11 while running the NAS benchmarks. With the exception of `cg` and `sp` codes running on BI-R21 where EDP was within 2.17%, the reduction in time-to-completion coupled with interleaved memory configurations improved memory system EDP by up to 60%. Relative to BI-R11, increasing rank interleaving yielded between 6.5% to 56.5% reduction in EDP. Increasing branch-interleaving instead of rank interleaving (BI-R11 vs. BS-R21) reduced EDP for `is` (24.6% vs. 17.9%), `lu` (19.6% vs. 10.7%) and `sp` (9.9% vs. 6.5%), although the EDP reduction was lower for `cg` (8.6% vs. 17.7%). The use of rank or branch interleaving led to EDP reductions that were within 1% for `ep` and `bt`, for BI-R11 and BS-R21 respectively.

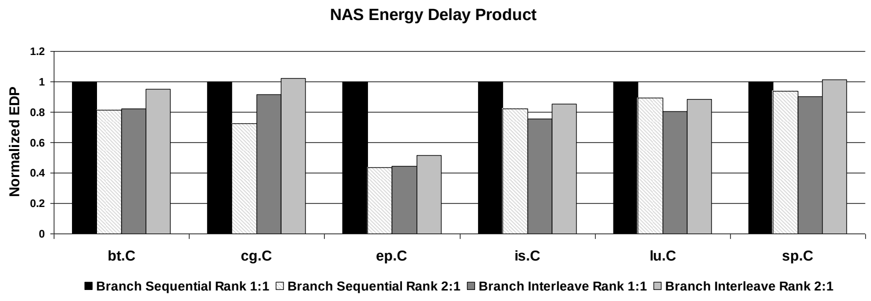
Combining branch and rank interleaving reduced EDP relative to the sequential configuration for 4 of the 6 benchmarks (`bt`, `ep`, is, and `lu`) and was within 2.17% for cg and sp. However, in all cases, the EDP reduction using BI-R21 was lower than using rank (BS-R21) or branch (BI-R11) interleaving in isolation. This was due to the higher power costs of interleaving across all devices in the system. Similar to several of the SPEC codes, the NAS benchmarks did not take advantage of the higher available bandwidth to offset the cost of interleaving.

Thermal Impact: Figure 5.7(c) shows the mean DIMM temperatures running the NAS

(a)



(b)



(c)

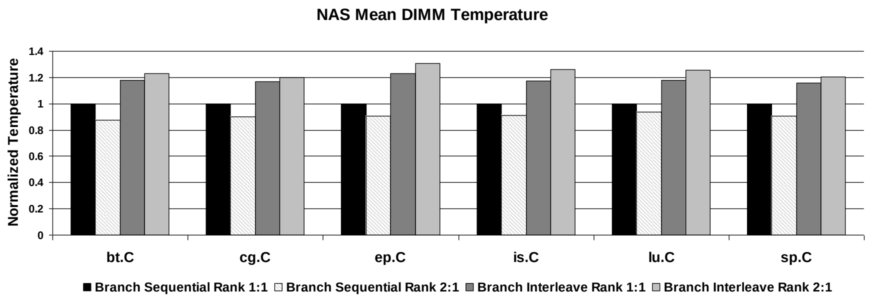


Figure 5.7. (a) Time-to-completion, (b) EDP, and (c) memory system temperature results of interleaved memory configurations normalized to the sequential configuration for the NAS benchmarks.

benchmarks on all four memory system configurations normalized to BS-R11. For all of the NAS codes, BS-R21 reduced the mean memory system temperature by 6.5% to 12.5% relative to BS-R11. Although this configuration did not interleave across branches, the 2:1 rank interleaving distributed accesses across the DIMMs of each branch. Similar to the SPEC results, the mean temperature was lower due to the traffic imbalance between the two branches. The traffic pattern translated directly into a temperature disparity between the hot DIMMs and the cooler DIMMs that led to the lower overall mean temperature. The temperatures of the branch interleaved configurations, BI-R11 and BI-R21, were 17.8% to 30.7% higher than the Branch Sequential configurations. As expected, the higher temperatures were a direct result of increased parallelism within the memory system.

5.4 Chapter Summary

Interleaving exploits inherent parallelism within the memory system to increase bandwidth. Our STREAM results showed that interleaving delivers up to 35% more bandwidth than sequential schemes using the same hardware configuration. However, we've also shown that many applications, such as the SPEC CPU2006 codes, are bandwidth-insensitive and do not significantly benefit from the additional memory bandwidth. Across the spectrum of benchmarks we used, memory energy consumption increased as interleaving dimensionality was scaled; although for a number of applications, our EDP results showed the additional energy costs were offset by performance improvements. Unlike energy, temperature consistently scaled as interleaving dimensionality was increased. The elevated temperatures are cause for concern given that higher temperatures reduce reliability over time.

Given the continuing use of DRAM-based memory systems, multi-core CPUs will continue to drive the need for low-latency high memory bandwidth. However, as we've shown, simply maximizing the dimensionality of interleaving does not automatically result in scalable bandwidth and can reduce reliability due to high operating temperatures.

One approach might include partitioning the traditional monolithic memory system and using single dimensional interleaving schemes within memory regions might also improve memory power, performance, and thermal efficiency. As flash technologies yield devices with lower access latencies, hybrid memory systems that combine new nonvolatile memory with traditional DRAM-based memory may reduce the need for interleaving as well. Further evaluation of such architectures constitutes an area of our future work.

Chapter 6

Improving the Energy Efficiency of Interleaved Memory Systems

In this chapter, we propose combining asymmetrically interleaved memory configuration with an OS-based control system for reducing the energy consumption of complex, highly-interleaved memory systems. We introduce new management and bandwidth-sensitive algorithms that extend our earlier control-theoretic runtime system designed for sequential memory systems to interleaved systems. We show these interleaving-aware techniques reduce memory energy consumption while preserving performance and bandwidth on traditional symmetrically interleaved memory topologies. We then propose using our system on asymmetrically interleaved memory system, in which devices within the topology are interleaved at multiple dimensions. We show that combining our power-aware management techniques on unorthodox memory configurations further improves energy efficiency while maintaining application performance and system bandwidth. These results demonstrate that adept software management can exploit non-traditionally interleaved memory systems to improve memory energy efficiency. Moreover, our results suggest future memory architectures should reevaluate traditional, performance-motivated approaches for power and energy efficiency.

6.1 Introduction

The power consumption of server-class systems has increased substantially in recent years. According to an EPA report to Congress, servers and data centers consumed an estimated 61 billion kilowatt-hours in 2006 at a cost of more than $4.5 billion [55]. Over 68% of the energy consumed was attributed to volume servers, which doubled between 2000 and 2006. This class of servers account for 99% of the more than 1.5 million servers sold in the United States annually. Further, the heat produced requires elaborate, costly cooling technologies internally and across the data center. In the long term, elevated average internal and ambient temperatures may reduce the mean-time-between-failure of these systems. Left unchecked, the power costs of operating servers is expected to double again within the next five years.

Numerous power-aware techniques have been proposed to adaptively schedule component power mode transitions to provide performance (and power) on demand within desired energy constraints. These techniques have been used to reduce the power consumption of processors [17, 25, 26, 35, 67, 107, 164], disks [17, 118, 172], high-speed interconnects [123], and memory [42-45, 48, 58, 59, 68, 87-90, 92, 117, 131, 132, 153, 171]. Recent increases in memory density and capacity have led to the emergence of memory as a significant consumer of system power [114]. This has driven the development of numerous power-aware techniques to reduce memory system power consumption. However, most of the techniques proposed to date have been designed for sequential memory systems, where device power state transitions are unencumbered by the complexities of interleaved address mapping schemes. This is problematic for servers since many chip-multiprocessor servers incorporate multi-dimensional interleaving to maximize memory bandwidth. Given interleaving is a widely accepted technique to improve the performance of systems and applications, previously proposed power-aware memory management techniques may not be practical for servers.

Reducing the energy consumption of highly-interleaved memory systems is challenging. Server memory architectures use complex, multi-dimensional interleaved designs that distribute the cache lines of a contiguous physical address space across multiple memory devices to increase peak theoretical bandwidth. This distribution causes dependencies between devices that limit the effectiveness of memory power-management techniques. In the worst case, these dependencies may prevent devices from being transitioned into lower power states without significantly degrading performance even though only a fraction of memory is being used. Second, reducing interleaving dimensionality within the memory system can improve the power-manageability of the memory system, but can degrade application performance by reducing bandwidth. To avoid performance degradations for bandwidth-sensitive applications, power-aware techniques must balance device bandwidth characteristics with systemic bandwidth utilization during power-state transitions. Third, most systems are configured to use symmetric interleaving schemes in which the level of interleaving throughout the memory system is fixed. The performance and energy of asymmetrically interleaved memory systems has been largely unexplored. Finally, although many operating systems are designed to accommodate regions with non-uniform memory latency characteristics, few are designed to account for regions with varying bandwidth properties.

In this chapter, we address the problem of reducing power and energy in highly interleaved memory systems while preserving application performance and bandwidth. We have extended the control-theoretic, power-aware runtime system we developed for sequential memory systems to proportionally scale memory capacity with application demand to conserve energy within interleaved memory systems [153]. Since interleaving dimensionality changes peak bandwidth, naively transitioning device power states can degrade system and application performance. To avoid these penalties, we developed and incorporated new power-aware algorithms in Memory-MISER to account for the bandwidth implications of transitioning interleaved devices between multiple power states. More specifically, this chapter offers the following contributions:

- We extend our earlier power-aware runtime system to support the detection, onlining, and offlining of memory devices interleaved at multiple granularities.
- We propose, implement, and evaluate new device power-state transition algorithms to reduce memory power within interleaved memory systems by tracking system-wide memory demand and bandwidth utilization.
- We propose and evaluate our interleaving-aware control system on traditional, symmetrically interleaved memory systems. Our results show that we reduced memory energy consumption by up to 50 percent.
- We find that combining our memory power-management techniques with asymmetrically interleaved memory systems improved EDP by up to 58 percent.

This chapter is organized as follows. Section 2 discusses prior work to reduce memory power consumption. Section 6.2 describes the OS and control system changes we made to Memory MISER to reduce the power of interleaved memory systems. Section 6.3 discusses our evaluation methodology. In section 6.4 we discuss energy and performance results on symmetric and asymmetric interleaved memory systems using several control policies. We conclude with a discussion of the implications of our observations in this work on future memory system designs as well as future work.

6.2 Interleaving-Aware Memory Miser Implementation

Interleaved memory systems map contiguous cache lines to multiple devices, breaking the one to one relationship between devices and physical address space used in our earlier implementation. Consider the earlier example memory system composed of 8 devices each with 1Gbyte capacity under interleaving. Even though the physical devices have the same capacity, the mapping of devices into the physical address space is different. Whereas a 1Gbyte device is mapped to a specific 1Gbyte region within the physical address space in a sequentially mapped system, two or more different memory devices may be mapped to the same 1Gbyte region in an interleaved system. Since independently transitioning a device involved in an interleaved range would have dire performance and system stability implications, our previous sequential-oriented approach was impractical for interleaved memory systems. We subsequently extended the operating system and control system within Memory MISER to overcome sequential-oriented design limitations to reduce the power and energy of interleaved memory systems. This section highlights the changes we made to the firmware, OS, and control system to reduce the energy consumption of interleaved memory systems.

6.2.1 System Firmware

System firmware is typically used to initialize the mapping of memory devices to the physical address space within the memory controller after detecting the number and capacity of memory devices populated in a system. On production-quality systems, this firmware maximizes interleaving dimensionality across populated memory systems to improve bandwidth and performance. However, since we wanted to understand the power and performance implications of reducing the level of interleaving within the memory system, we needed to be able to change how memory was interleaved. This led us to modify the memory controller initialization sequence that defines how devices map into the physical address space.

We changed system firmware to allow each interleaving dimension to be varied independently within a topology. Since our evaluation system supported two configurable interleaving dimensions, there were a total of six distinct configuration possibilities. Thus our firmware was designed to configure the memory system to interleave across devices within the same topology at multiple granularities. For example, in our eight device topology we could configure several devices to use a branch-sequential, 4:1 rank interleaving scheme, while mapping the remaining devices sequentially. Alternatively, we could configure two devices to use branch-interleaving, but not interleaved at rank granularity, while interleaving the remaining six devices using a 2:1 rank-interleaving scheme. As a result, we were able to fully control interleaving for each device in the system simply by updating several parameters that we maintained in non-volatile memory.

6.2.2 Operating System Implementation

A key component of Memory MISER is a 2.6 x86-64 version of the Linux kernel modified with the capability to transparently online and offline physical memory devices while preserving application integrity. The earlier Memory MISER implementation was designed for sequential memory systems, where each device could be onlined or offlined independently. However, contiguous cache lines may be mapped across multiple devices for large regions of the physical address space within interleaved memory systems, creating dependencies between devices. Although we were able to leverage the power-state transition interfaces and page allocation shaping techniques proposed in previous work, we had to dynamically repartition the power-state transition logic, allocation mapping and tracking mechanisms, and add bandwidth monitoring logic within the operating system specifically for interleaved memory systems. Moreover, we had to further modify each of these areas to dynamically accommodate a range of asymmetrically interleaved memory topologies and differentiate between devices based on bandwidth characteristics. This section introduces the key new OS changes we developed to efficiently power-manage interleaved memory systems.

First, we updated the initialization sequence to parse the interleave configuration registers within the memory controller to detect the underlying device mapping scheme used for the physical address space. The mapping scheme details how ranks, channels, and controllers are aggregated to form interleaved regions. We also extracted device sizes as well as where interleaved regions mapped into the physical address space. This enabled our system to determine how interleaved devices mapped to the physical address space as well as the interleaving level for each region.

Second, we created logical memory devices to manage the power state transitions of the underlying physical devices involved in an interleaved region of the physical address space. Memory systems may use multiple levels of interleaving which creates dependencies between devices. The dependencies are a function of the level of dimensionality applied across a set of devices. Figure 6.1 depicts an eight device memory system that uses three levels of interleaving. The lower half of the address space (region I) is mapped to devices A, B, C, and D, which is interleaved using a 4:1 rank scheme. Since this scheme distributes contiguous cache lines for the address range 0 to x-1 across all four devices as shown in Figure 6.1(c), devices A, B, C, and D must be transitioned between power states in aggregate. Region II is mapped to devices E and F, which is configured using 2:1 rank interleaving. Similarly, because cache lines for region II are distributed across both devices, these two devices must be transitioned between power states together as well. Region III is

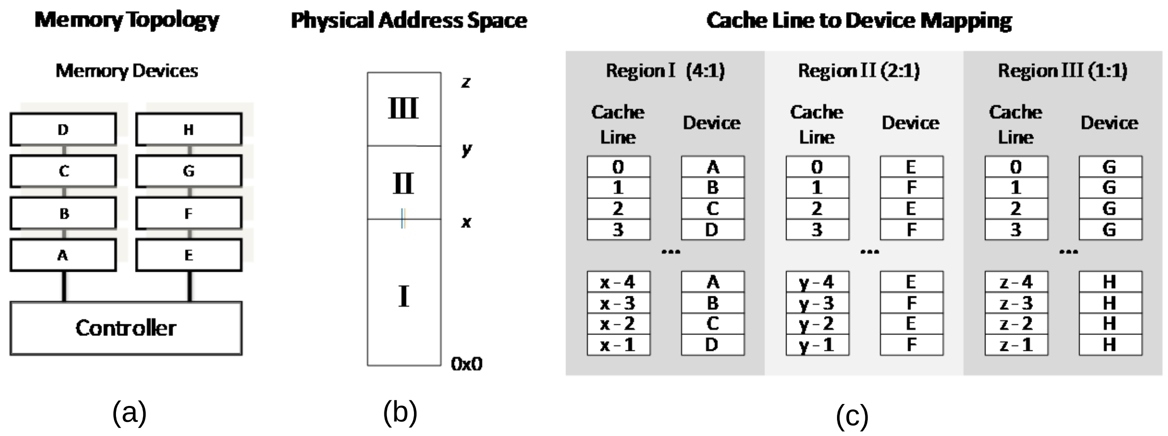


Figure 6.1. (a) Example two-branch, eight device memory system that supports two configurable levels of interleaving: branch and rank. (b) Physical address space showing three memory regions, I, II, and II. (c) Cache line to device mapping for the three memory regions. Region I interleaves cache lines across the ranks of devices A, B, C, and D. Region II interleaves cache lines across devices E and F, and Region III sequentially maps cache lines to devices G and H.

mapped to devices G and H; however, since they are sequentially mapped, there are no dependencies between the two devices and each may be transitioned between power states independently. In contrast, due to interleaving, offlining a single physical device within region I or II would cause cache line accesses to be aborted, resulting in severe performance penalties due to stalls or worse, system faults.

To avoid such degradations, we abstracted dependent physical devices into logical superset memory devices that correspond to sets of interleaved physical devices. We then constrain device power-state transitions at the granularity of logical devices. In other words, onlining or offlining the logical device associated with region I would transition all four underlying devices A, B, C, and D in aggregate. This ensures that inadvertent cache line accesses to offline devices involved in an interleaved region do not cause system instability. More pragmatically, the memory controller in our system initializes the device interleaving configuration during the boot-strap sequence and cannot be modified at runtime. If the memory controller supported dynamically changing the interleaving configuration during normal operation, we could simply update the device mapping scheme to reduce interleaving dimensionality for the other devices. However, this would likely introduce additional performance penalties since memory traffic for the affected devices would have to be halted to prevent faults.

In addition to device dependencies, we also modified the kernel to use the interleaving dimensionality to predict the relative bandwidth of logical memory devices. To enable the control system to measure and monitor bandwidth, we integrated kernel support for accessing hardware performance counters into the kernel. We then categorized each logical device by one of three bandwidth characteristics: low, medium, and high. Highly interleaved logical device sets are identified as high bandwidth devices, whereas sequentially mapped devices are identified as low bandwidth devices and those logical devices using intermediate interleaving schemes are labeled as providing medium bandwidth. This simplistic scheme enables the control system to quickly differentiate logical devices by bandwidth characteristics.

6.2.3 Control System Implementation

For controlling the power-state transitions of interleaved memory devices, we used the same control system model as the one previously proposed for sequential memory systems [153]. Our control system model consists of two parts: a controller and the plant function. For the controller, we use a standard PID controller [8, 9] implemented in a root-privileged, application-level daemon. The plant function models the OS-based device online/offline actuation logic described in the previous section. In the sequential control system, the plant function was modeled as a step function, as the plant would either online or offline physical memory devices based on the input received from the PID controller. In this work, we have modified our control system model to track and control logical memory devices. Although the granularity of tracking and control is different, this amounts to an implementation constraint and does not impact the control system model. Thus, we leverage the control system model from the sequential domain for interleaved memory systems.

In earlier work, the control gains for the PID controller were analytically determined through stability analysis after the control system transfer function was derived [153]. In this study, because we leverage same the control system model from the sequential domain for interleaved memory systems, the mathematical formulation of our control system model remains unchanged. Thus, we used the same control gains as in previous work; that is, $K_P=K_I=K_D=0.015625$.

We initially configured our control system to sample memory demand and bandwidth every 500 milliseconds. However, we found that since the smallest granularity of power-managed memory devices was 1Gbyte, we were able to reduce this frequency to as much as 2 seconds. However, for the results presented in this paper, we used a sample frequency of 1 second to minimize control system response during rapid fluctuations in demand and bandwidth. Throughout all experiments, the control system was stable (e.g. convergent) under all workloads and exhibited reasonable transient response using these control gains and a 1 second sampling frequency.

In sequential and symmetrically interleaved memory systems, all logical devices have the same peak theoretical bandwidth. In these memory systems, selecting which devices to online is simply based on memory demand, while offline operations select devices with the fewest allocated frames to minimize migration costs. However, in asymmetrically interleaved configurations, the bandwidth of logical devices may differ. In such systems, offlining a high-bandwidth logical device when system bandwidth utilization is high may degrade application performance, even though page demand can be satisfied with lower-bandwidth devices. Conversely, offlining a high-bandwidth logical device when system bandwidth is low saves more power than offlining a lower bandwidth logical device since higher bandwidth logical devices include more underlying physical devices. Similarly, onlining a low-bandwidth logical device when bandwidth is high might meet increased page demand, but limit achievable bandwidth, unnecessarily penalizing performance.

We subsequently modified the power state transition logic within our control system to evaluate the trade-off between bandwidth and power when transitioning devices between power states. Using hardware performance counters, we monitored system-level memory bandwidth by regularly aggregating bus transactions for all cores and translating from transaction counts to bandwidth. We retained recent bandwidth observations within a history buffer and changed the device power state transition algorithm to predict the bandwidth impact of changing device power states. On each pass, we assigned weights to devices after evaluating the fitness of onlining or offlining logical devices given recent bandwidth utilization. The algorithm then selected devices with the highest weights for the power-state operation. While the control system continued to track memory demand, this alleviated the previously described cases in which device state changes limit bandwidth. These changes ensure that the memory devices with bandwidth characteristics necessary for the workload remain online to satisfy page demand.

6.3 Evaluation Methodology

We used an Intel Xeon dual-core, dual-processor (DP) with an Intel 5000 chipset-based server system in our evaluation. We chose DP servers instead of larger systems with higher-density memory configurations for several reasons. First, a 2007 survey funded by AMD revealed the power consumption of volume and mid-range servers increased 16% and 51% respectively from

2000 to 2005 [110]. Since these types of servers account for 99% of the more than 1.5 million servers sold in the United States annually, the energy efficiency of this class of systems is critical to operational costs. Second, although the capacity of DP servers is generally limited compared to larger multiprocessor systems, the memory systems often use multiple levels of interleaving.

Our test system supported two levels of interleaving that were configurable by firmware - branch and rank. DRAM bank interleaving was also supported, but was always enabled in hardware and thus not configurable. The memory system consisted of four physical memory channels, two per branch; Paired channels operated in lockstep creating one logical channel per memory branch. Since the branch and rank interleaving configuration controls were separate, we were able to vary each variant in isolation. For rank interleaving, the memory controller supported up to a 4:1 interleave; however, we limited our evaluation to maximally using 2:1 rank interleave due to the limited DIMM slots and to experiment with unorthodox, mixed interleaving configurations.

6.3.1 Memory System Emulation

Since platforms with memory devices capable of transitioning to a fully offline state at DIMM granularity were not available at the time of this work, we ran our modified kernel and controller implementation on a system that supported multiple levels of interleaving and emulated power-state transitions. This section discusses the two forms of emulation we used to evaluate our approach.

Power-State Emulation: Transitioning memory devices online is not instantaneous. To ensure our emulation techniques account for the cost of power-state transitions, we instrumented our firmware found the time to initialize four 4Gbyte DIMMs took approximately 600-700ms. To be conservative, in this evaluation we used a fixed delay of 500ms for each device online operation to account for hardware delays. We also added a 50ms delay for each offline operation to account for the cost of dropping power and related signals from affected devices.

Memory Configuration Emulation: Our evaluation system was populated with eight 2Gbyte DIMMs for a total capacity of 16Gbytes. However, even with eight DIMMs, the number of interleave configurations we could evaluate was limited due to layout constraints of the memory topology in our platform. To avoid constraining our experiments and biasing our results due to the limitations of our evaluation system, we also emulated interleaved memory topologies.

We emulated an 8Gbyte capacity memory system consisting of eight 1Gbyte memory devices. For each experiment, we would first configure firmware to physically interleave the eight 2Gbyte DIMMs using a specified experimental configuration. We would then limit the amount of memory the OS could use to 8Gbytes, which we would partition into eight 1Gbyte devices. Based on the experimental configuration used, we would map these eight 1Gbyte devices to regions of the physical address space that used the matching interleaving scheme at the level of physical devices. This ensured that the bandwidth characteristics of interleaved devices managed by our control system were real. The remaining 8Gbytes of physical memory capacity was not used by the OS. This multi-level emulation enabled us to evaluate the real bandwidth and implications of transitioning interleaved memory devices between power states, but provided the flexibility to consider the effects of unorthodox, asymmetrically interleaved memory systems. We ensured performance and bandwidth correctness in our emulated topologies by verifying memory bandwidth and application performance for each of the logical devices was consistent with the emulated level of interleaving using STREAM, several micro-benchmarks, and hardware performance counters.

For the symmetric configuration evaluation we emulated four standard configurations, branch-sequential,1:1 rank-interleave (BS-R11), branch-sequential, 2:1 rank-interleave (BS-R21), branch-interleave, 1:1 rank-interleave (BI-R11), and branch-interleave, 2:1 rank-interleave (BI-R21) , the specifics of which were described in section 2. For the asymmetric evaluation, we emulated six asymmetrically interleaved configurations and compared those to two of the symmetric configurations. Highly interleaved configurations, such as BI-R21, maximize peak theoretical bandwidth but limit the number of devices that can be effectively power-managed. In contrast, sequential memory systems such as BS-R11 increase the number of devices that can be power-managed at the expense of lower peak bandwidth. The six intermediate configurations constitute the spectrum of unorthodox mapping schemes between these two extremes.

6.3.2 Workloads

To account for varied memory utilization loads, we evaluated several benchmarks across the interleaving schemes supported by each of our test systems. We used benchmarks from SPECint (*perlbench, bzip2, gcc, mcf, gobmk, hmmer, sjeng, libquantum, h264ref, omnetpp, astar, and xalancbmk*) and SPECfp (*bwaves, games, milc, zeusmp, gromacs, cactusADM, leslie3d, namd, dealll, soplex, povray, calculix, GemsFDTD, tonto, lmb, and sphinx3*) from the SPEC CPU2006 suite. To measure the impact of interleaving on memory bandwidth, we used the STREAM benchmark built using the GNU gfortran compiler. We also used the class C versions of codes from the NAS 3.0 suite (*bt.C, cg.C, ep.C, is.C, lu.C, and sp.C*) to analyze the power-performance effects of interleaving. We built the SPEC CPU2006 and NAS codes using gcc version 4.1.2 using –O3 optimizations.

6.3.3 Experiments and Metrics

We ran the SPEC and NAS codes on the eight emulated memory configurations using our interleave-aware Memory MISER. For each experiment, we executed the SPEC and NAS codes serially on each memory configuration 5 times with our control system disengaged followed by 5 runs with the control system engaged. We measured and recorded memory demand, online memory capacity as controlled by Memory MISER, as well as memory bandwidth. For the results presented in this chapter, the results compare the average of the 5 runs using our control system.

Throughout our evaluation, we use the following metrics: total memory system energy, memory bandwidth, and the energy-delay product [56, 69]. For each of the configurations, the results reported in this paper are normalized to the same memory configuration without our power management control system.

To ensure our OS memory manager changes did not impact runtime performance, we measured page faults and execution time running SPEC benchmarks, LMBench, and the NAS Parallel Benchmarks using our modified kernel as well as using an unmodified kernel. For consistency we disabled the controller daemon so memory remained online during the comparison. We observed less than 1% performance variation between kernel implementations for all codes across a number of runs.

6.4 Experimental Results

We evaluated our interleave-aware Memory MISER using SPEC and NAS benchmark applications on a total of 10 different interleaved memory configurations, 4 symmetric and 6

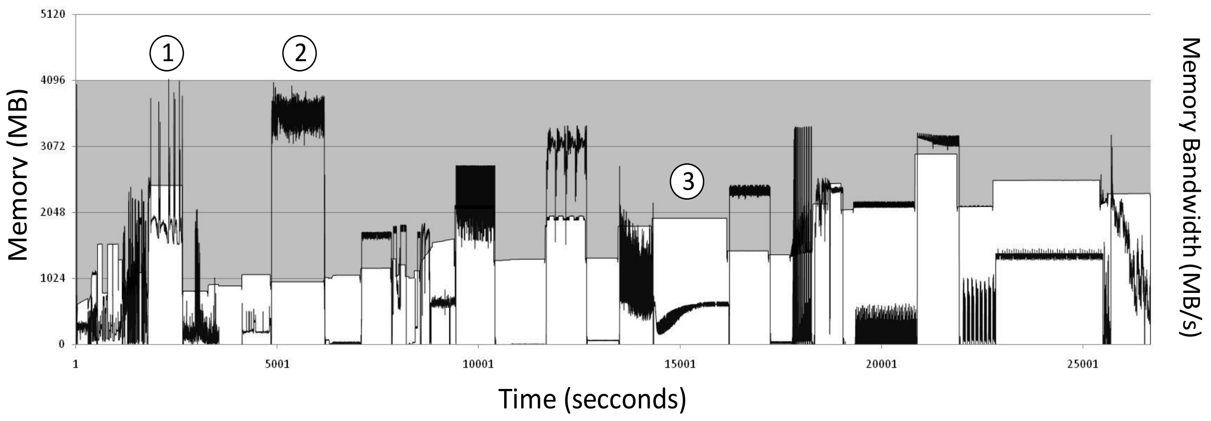


Figure 6.2. Memory MISER scales online memory capacity to meet demand while running SPEC and NAS benchmarks using the highly-interleaved BI-R21 configuration. The white area shows memory demand, the gray area shows online memory, and the black line shows memory bandwidth.

asymmetric configurations. For each configuration, we ran all of the benchmarks serially to characterize memory demand and bandwidth utilization. We then compared the energy delay product (EDP) of our techniques relative to the default control policy and a static control policy for each memory configuration. We also evaluated the performance and energy impact of asymmetric configurations relative to symmetric configurations using our power-aware techniques.

Figure 6.2 shows memory demand and memory bandwidth during a run of all benchmarks on one of the highly-interleaved (BI-R21) configurations. The white area shows memory demand, the gray area shows how Memory MISER scaled online device capacity, and the black line shows memory bandwidth. Running these codes serially resulted in a wide variance in both memory demand and bandwidth utilization. We have identified several points within the trace that exhibit notable memory allocation and utilization patterns. For example, point 1 in Figure 6.2 highlights the execution of the SPEC benchmark mcf. Memory demand increases rapidly when mcf starts, remains near constant for the duration of execution, and then subsides after allocated memory is freed. Memory bandwidth varies significantly, ranging from 1.6Gbytes/second to as high as 4Gbytes/second. Because demand never exceeds 3Gbytes and 4Gbytes is kept online by our control system, the additional memory capacity (1Gbyte) that is online while the benchmark is running is not needed. However, since the minimal logical device granularity for this configuration is 4Gbytes, the excess 1Gbyte is kept online to avoid performance penalties. In a lesser-interleaved configuration, the additional capacity may have been offlined to save additional energy, at the cost of bandwidth. In this case, since bandwidth utilization is high, performance and stability are preserved even though the achievable energy savings is at most 50 percent.

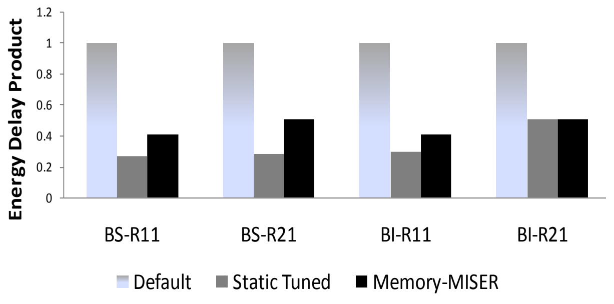
The second point in Figure 6.2 highlights the importance of tracking bandwidth in addition to demand in interleaved configurations. For this benchmark (libquantum), systemic demand is only 1Gbyte, which could be minimally met with a single device in a sequential memory system. However, given the high bandwidth utilization of this application, offlining devices based solely on memory demand would cause only a single device with lower realizable bandwidth to be online. This would restrict available bandwidth and degrade application performance.

In contrast, the third point in Figure 6.2 illustrates an application with moderate memory demand, but low bandwidth utilization. In this case, the bandwidth insensitivity of this application indicates 2 sequentially mapped 1Gbyte devices could be used to meet demand without adversely impacting performance. However, in this case transitioning interleaved devices into low power states and onlining sequentially mapped devices is cost prohibitive. This also constrains realizable energy savings, but ensures application performance is not degraded.

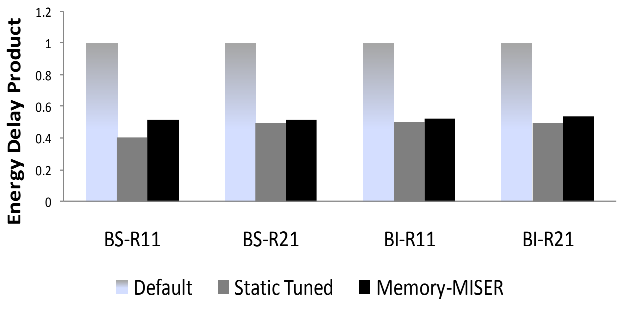
6.4.1 Symmetric Interleaving Configurations

We initially evaluated our interleave-aware Memory MISER on memory configurations that symmetrically interleaved cache lines across the memory system. Since interleaving dimensionality decreases the number of devices available to transition into lower power states, we varied the level of interleaving used across the memory system to measure the efficiency of our power-aware techniques when interleaving dimensionality is relaxed. Using our customized firmware, we tuned NVRAM parameters to evaluate up to six memory interleave configurations, including BS-R11, BS-R21, BS-R41, BI-R11, BI-R21, BI-R41. However, because our system only had 8 DIMMs it was impractical to offline devices at runtime using the fully interleaved configurations. Consequently, we limited our evaluation to configurations with at least two logical devices. The results here used the following configurations: BS-R11, BS-R21, BI-R11, and BI-R21.

Control Policy Analysis: For each of the interleave configurations, we ran all of the SPEC and NAS benchmarks using three different control policies: a default control policy, a statically-tuned, oracle control policy, and Memory MISER. The default control policy keeps all devices online to maximize performance and bandwidth for a given configuration, but wastes energy when demand is less than total capacity. The statically tuned control policy uses a priori knowledge of workload memory demand and bandwidth characteristics to optimize power-state transitions for performance and energy efficiency on each memory configuration. Because the oracle policy is specifically tuned for the workload, it should yield the best EDP results for each of the configurations. Finally, we use our dynamic control system, Memory MISER to dynamically scale memory capacity based on memory demand and bandwidth utilization.



(a)

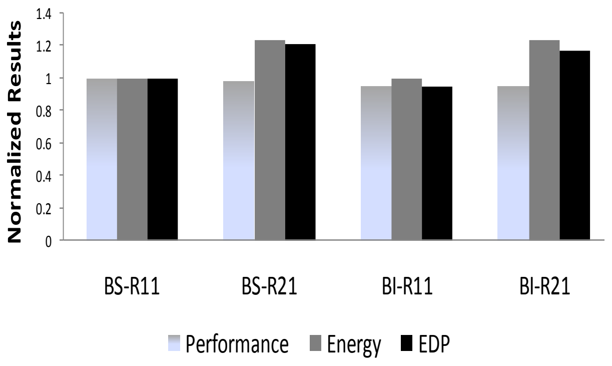


(b)

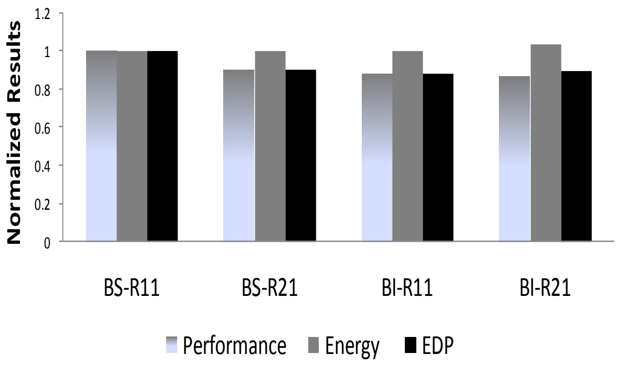
Figure 6.3. EDP results for default, static tuned, and dynamic (Memory MISER) control polices normalized to the default policy for the four symmetric configurations using the (a) SPEC and (b) NAS benchmark suites.

Figure 6.3 compares the Energy-Delay Product of the three control policies normalized to the default control policy for the SPEC and NAS benchmark suites on each of the interleaved memory configurations. The performance of the oracle policy was within 1 percent of the default policy. By tuning for the memory demand and bandwidth characteristics of each workload, the oracle control policy reduced memory energy consumption significantly. Relative to the default control policy, the energy consumption using the oracle control policy on the BS-R11 configuration was reduced by 73.4 percent for SPEC and 48.1 percent for NAS. The energy consumption of the other configurations was also reduced: 71.8 percent for SPEC and 59.1 percent for NAS on the BS-R21 configuration, 71.3 percent for SPEC and 50 percent for NAS on the BI-R21 configuration, and 50 percent for both SPEC and NAS on the BI-R21 configuration. However, these savings constituted the best-case reductions using extensive tuning of device power-state transitions for each application within the respective execution sequences. We found that simply changing the order benchmarks were executed caused significant performance degradations, leading to up to 22 percent worse EDP for SPEC and 11 percent than the default policy.

By dynamically tracking memory demand and bandwidth and offlining unneeded devices Memory MISER reduced EDP of the SPEC and NAS suites on all of the memory configurations by up to 50 percent. For the sequential configuration (BS-R11), our dynamic control system reduced energy consumption for SPEC by 40.1 percent with less than 0.2 percent performance impact relative to the default configuration. Similarly, for NAS, Memory MISER reduced energy consumption by 48.1 percent with only 0.4 percent performance degradation. However, when interleaving dimensionality was increased, the number of logical devices decreased, reducing the effectiveness of our power-aware control system. This was evident using the BS-R21 configuration, in which the level of rank interleaving was increased relative to BS-R11. This reduced the number of logical devices to four, limiting how closely our system could track memory demand. As a result, the energy savings realized using Memory MISER were not as significant as the oracle policy, but still reduced memory energy for SPEC by 50.1 percent relative to the default control policy with less than 0.53 percent impact on performance. Similarly, using the NAS codes, our system reduced memory energy to 48.1 percent of the default control policy with less than 0.35 percent impact on performance. The BI-R11 configuration uses branch interleaving but not rank interleaving. For this configuration, Memory MISER reduced memory energy for SPEC to 40.48 percent and NAS to 47.6 percent relative to the default control policy; the performance impact was limited to 0.89 percent and 0.69 percent for SPEC and NAS suites respectively. Because the BS-R21 configuration uses branch and rank interleaving only two logical memory devices were available for power-management. So for our test system, any workload with sustained memory demand greater than 4Gbytes, or half of total capacity, would not likely benefit from using our power-aware techniques. Because the memory demand of the SPEC and NAS benchmarks did not exceed the granularity of the high-capacity



(a)



(b)

Figure 6.4. Performance, Energy, and EDP results for the four symmetric configurations using Memory MISER for the (a) SPEC and (b) NAS benchmark suites normalized to the sequential configuration.

logical devices in the BS-R21 configuration, Memory MISER reduced energy consumption to 50.2 percent of the default configuration for SPEC and 51.2 percent for NAS, with only a 0.53 percent and 0.29 percent impact on performance respectively.

Symmetric Configuration Analysis: Increasing interleaving dimensionality improves peak theoretical bandwidth, and hence application performance at the cost of higher memory energy consumption. Figure 6.4(a) compares the performance, energy consumption, and EDP of SPEC for the four memory configurations normalized to the sequential configuration, BS-R11. Due to the overall bandwidth insensitivity of the SPEC benchmarks in aggregate, increasing interleaving dimensionality did not significantly improve performance. For example, the mean performance of all runs on BS-R21 was within 2 percent of the sequential BS-R11 configuration, while using both BI-R11 and BI-R21 led to performance improvements of less than 5 percent. Additionally, energy consumption was up to 23.5 percent higher for the interleaved configurations than the sequential configuration for the SPEC benchmarks. This was because of the additional devices that were kept online to avoid degradations or faults due to interleaving. Combining the minimal performance improvements of SPEC with the higher memory energy led to higher EDP for interleaved configurations when normalized to the sequential configuration.

Figure 6.4(b) compares the performance, energy consumption, and EDP of the NAS benchmarks for the four memory configurations normalized to BS-R11. The additional bandwidth realized by increasing interleaving dimensionality improved the performance of the NAS codes. Using the BS-R21 configuration reduced NAS execution time by 9.7 percent by increasing rank interleaving. Similarly, using branch interleaving reduced execution time by 11.8 percent compared to the sequential configuration, and by 13.3 percent when combined with rank interleaving. The energy savings achieved using the interleaved configurations were within 5 percent of the sequential configuration. Memory energy for BS-R21 and BI-R11 using Memory MISER was within 1 percent of the sequential configuration, while the BI-R21 configuration consumed 3.76 percent more energy. In this case, the performance improvements of interleaving outweighed the minimized energy costs realized by Memory MISER as evidenced by the normalized EDP results. The EDP of BS-R21 was reduced by 9.7 percent relative to the sequential configuration, while the EDP of the two branch-interleaved configurations was reduced by 11.8 percent and 10 percent respectively. These improvements were a direct result of improvements in application performance due to increased memory bandwidth.

Based on these initial experiments, we made two observations. First, the device dependencies caused by high dimensionality limit the effectiveness of our power-management techniques. For example, since our system only had 8 DIMMs, a fully-interleaved configuration made it impractical to offline any devices at runtime. Further, even though demand was well below the capacity of the highly interleaved devices, energy was still wasted since dependent devices could not be offlined. Second, we found that increasing interleaving dimensionality does not automatically improve application performance. Based on our SPEC evaluation, we found that performance for bandwidth-insensitive applications varied less than five percent between interleaved and sequential configurations. Even using sophisticated power-aware techniques, highly-interleaved configurations can reduce the energy efficiency of the memory system. These two observations led us to explore using our power-aware techniques on asymmetrically interleaved memory configurations.

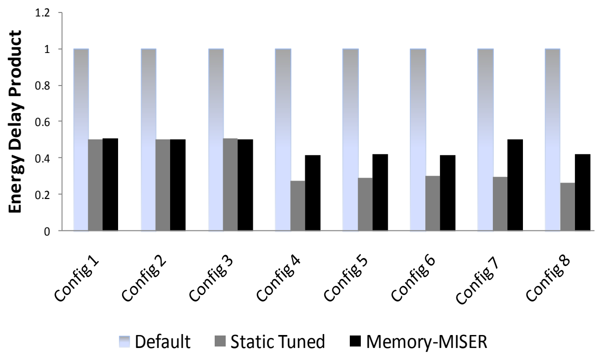
6.4.2 Asymmetric Interleaving Configurations

Instead of simply using a single symmetric interleaving scheme to uniformly distribute cache lines across the memory system we configured the memory controller to interleave devices within the memory system at multiple granularities. This improved the power-manageability of memory devices in the system by increasing the number of logical devices. However, it also created memory regions with different bandwidth characteristics. In our experiments with symmetrically interleaved configurations, the control system tracked page demand, but did not have to consider the bandwidth implications of device power state transitions. This section discusses the impact of using our power-aware techniques on eight interleaved memory configurations listed in Table 6.1, two symmetric and six asymmetric configurations. Config-1 provides the highest memory bandwidth since it is interleaved across branch and rank dimensions, whereas Config-8 is sequentially mapped with the lowest bandwidth characteristics.

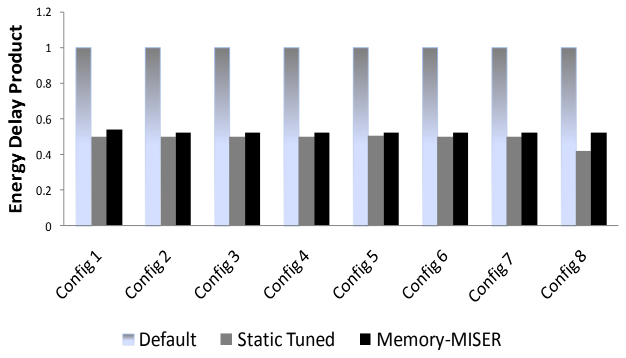
Control Policy Analysis: We ran the SPEC and NAS benchmarks on all eight interleave configurations using the same three control policies we used in the symmetric configuration experiments: a default control policy, a statically-tuned, oracle control policy, and Memory

MISER. For the oracle control policy, we tuned the power-state transitions for the SPEC and NAS code execution sequence on each of the configurations. For Memory MISER we used the same control gains within our control system to dynamically scale memory capacity based on demand. However, since the realizable bandwidth of the logical devices varied, we engaged the capability to evaluate the bandwidth impact of device power state transitions.

Figure 6.5 compares the EDP of using the three control policies normalized to the default policy while running the SPEC and NAS suites. For Config-1 and Config-8 we expected and observed similar results as outlined in the previous section since the configurations were symmetrically interleaved. We compared the asymmetric results against these two configurations since they constituted the boundary cases for interleaved configurations – Config-1 uses the highest degree of interleaving while Config-8 is a sequential configuration. For asymmetric configurations Config-2 and Config-3, we found the EDP of the oracle and our dynamic control policy was near 50 percent of the default policy. This was because we constrained the two control policies to retain at least one device online at all times. Further, to maximize bandwidth for OS use, we ensured that we always kept the logical device with the highest level of interleaving online at all times. For Config-1, Config-2, and Config-3, this meant that the logical device using the BI-R21 scheme was kept online, limiting achievable energy savings to 50 percent. For the other configurations, logical devices used either branch or rank interleaving schemes (but not both) which increased realizable energy savings. For SPEC, the EDP for the oracle policy was only 26.5 percent to 30.1 percent of the default control policy for Config-4, Config-5, Config-6, Config-7, and Config-8. In contrast, the EDP realized using the oracle control policy for the NAS codes ranged from 42.3 percent to 51.2 percent normalized to default control policy. EDP using the NAS benchmarks on the asymmetric configurations was lower



(a)



(b)

Figure 6.5. EDP results for default, static tuned, and dynamic (Memory MISER) control polices normalized to the default policy for the 2 symmetric configurations (Config-1 and Config-8) and the six asymmetric configurations (Config-2 through Config-7) using the (a) SPEC and (b) NAS benchmark suites.

than the symmetric configuration. This was because when we tuned the oracle policy, we ensured logical devices with the necessary bandwidth were used during power-state transitions. The EDP improvements using the oracle control policy was a consequence of three things: the increased number of logical devices that could be transitioned into lower power states, the performance insensitivity of the SPEC benchmarks to memory bandwidth, and for the NAS codes our predictions of which devices to use for applications that require high memory bandwidth.

Figure 6.5 also shows that Memory MISER scaled memory capacity to reduce memory energy consumption by 42 percent to 54 percent relative to the default policy for the SPEC and the NAS benchmarks. While the EDP improvements were significant, EDP did not improve as much as the oracle control policy for several of the asymmetric configurations. We found this gap in efficiency was caused by our bandwidth tracking algorithm. Comparing the traces of two runs for the oracle policy and Memory MISER, we discovered several cases in which Memory MISER selected logical devices with higher interleaving dimensionality to online than the oracle policy. Similarly, there were several points where our system offlined sequentially mapped devices instead of interleaved logical devices, whereas the oracle policy offlined interleaved devices. After reviewing the bandwidth traces before these transitions we found that our system had predicted that memory bandwidth would remain steady or increase. This prediction led to our conservative, performance-centric policy to select lower-bandwidth sequentially mapped devices to be offlined instead of interleaved devices. Similarly, when transitioning devices into the online state, Memory MISER selected devices with higher levels of interleaving, sacrificing energy for bandwidth to avoid performance penalties. In contrast, because the oracle policy had a priori knowledge of the memory demand and bandwidth characteristics for the workload, more aggressive power-down transitions were used. However, this required extensive tuning, whereas Memory MISER did not require any tuning and still improved EDP within 1 percent to 12 percent of the best case oracle policy.

Asymmetric Configuration Analysis: We also wanted to understand the efficiency of alternative asymmetric configurations using our interleaving-aware Memory MISER system. Figure 6.6 compares the performance, energy consumption, and EDP of the SPEC and NAS suites for the eight memory configurations described in Table 6.1 normalized to Config-1, the configuration with the highest level of interleaving. For the SPEC benchmarks, we found that using BI-R21 (which used 4 physical devices) and then varying interleaving dimensionality for the remaining four physical devices in the topology did not yield significant improvements since only half of the memory capacity was needed. As a result, the four devices with varied dimensionality were kept offline most of the time, yielding similar energy characteristics. However, the results for Config-4, Config-5, and Config-6 illustrate the energy benefits of reducing interleaving dimensionality for applications with modest bandwidth requirements, such

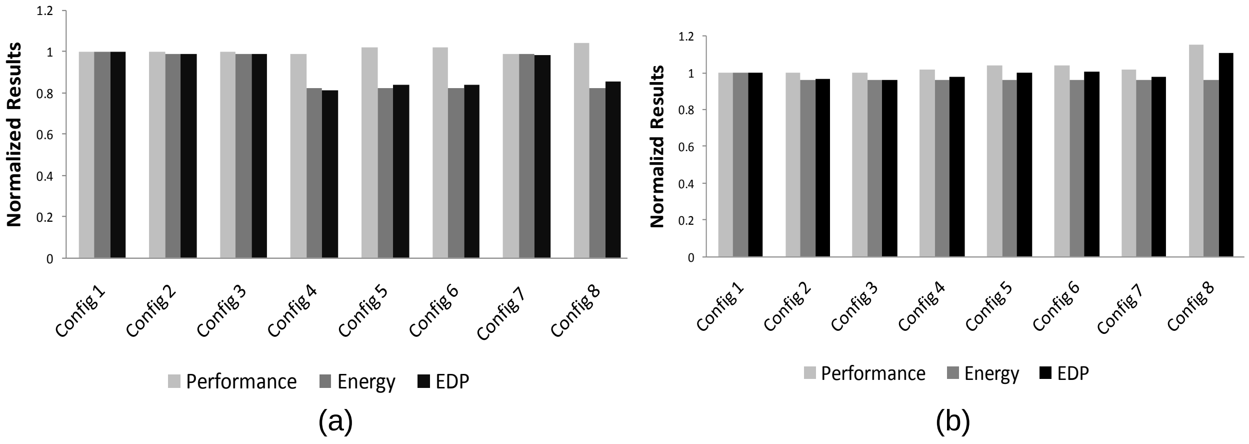


Figure 6.6. Performance, Energy, and EDP results for the 2 symmetric configurations (Config-1 and Config-8) and the six asymmetric configurations (Config-2 through Config-7) using Memory MISER while running the (a) SPEC and (b) NAS benchmark suites normalized to the symmetrically interleaved Config-1.

as the SPEC benchmarks. Energy consumption was reduced by 17.8 percent using Config-4, 17.6 percent Config-5, and 17.5 percent for Config-6. These energy reductions coupled with limited performance degradations improved EDP for these asymmetrically interleaved configurations relative to Config-1. Since the differences in performance were minimal, these improvements were a consequence of increasing the number of logical devices that could be transitioned into lower power states by our control system. After combining the performance impact with energy savings, we found that using Memory MISER on asymmetric configurations realized up to 19.51 percent lower EDP than the just using a standard highly-interleaved symmetric configuration.

Given the insensitivity of the SPEC suite to memory bandwidth, we also ran the serial NAS benchmarks suite on the eight configurations described in Table 6.1. Figure 6.6(b) compares the performance, energy, and EDP of the eight configurations normalized to the highly interleaved configuration, Config-1. We found that performance was degraded less than 0.12

percent by the variations introduced in Config-2 and Config-3. Reducing rank interleaving for the devices mapped to the lower half of the address space, but still interleaving across branches (from BI-R21 to BI-R11) reduced overall NAS code performance by 1.6 percent for Config-4 and 1.7 percent for Config-7. Increasing rank interleaving and alleviating branch interleaving for devices mapped to the lower half of the address space in Config-5 and Config-6 degraded performance by 4.1 percent and 4.6 percent respectively. Since Config-8 is sequentially mapped and does use interleaving, the performance of the NAS benchmarks was collectively reduced by 15.26 percent.

Using Memory MISER, the total memory energy consumption for the eight memory configurations was with 4 percent normalized to Config-1. This was because the memory demand of the NAS benchmarks did not fluctuate significantly during execution. Consequently, even though there were more devices available to transition into lower power states, the low variance in demand over time limited the opportunities to save additional energy. Because of this, the energy consumption benefit of using asymmetric memory configurations was limited.
Figure 6.6 also shows the Energy Delay Product of the asymmetric configurations relative to Config-1. Combining the limited energy savings with performance, the EDP results illustrate the benefits of using our dynamic control system on asymmetrically interleaved memory configurations. For example, the EDP realized for the six asymmetric configurations was decreased by up to 5 percent for the NAS codes over a standard symmetric configuration. Although this improvement (5 percent) is not as significant as the improvements observed running SPEC (up to 19.51 percent), using Memory MISER on asymmetric configurations further improved EDP over standard, symmetric configurations despite variations in workload memory demand and bandwidth utilization.

Table 6.1. **Memory configurations emulated for asymmetric evaluation**

Configuration	Type	Physical Address Regions	Interleave Type	Contiguous Cache Line Mapping	Logical Devices
Config-1	Symmetric	0 – 4GB 4– 8GB	BI-R21 BI-R21	A, E, B, F C, G, D, H	2
Config-2	Asymmetric	0 – 4GB 4 – 6GB 6 – 8GB	BI-R21 BI-R11 BI-R11	A, E, B, F C, G D, H	3
Config-3	Asymmetric	0 – 4GB 4 – 6GB 6 – 7GB 7 – 8GG	BI-R21 BI-R11 BS-R11 BS-R11	A, E, B, F C, G D H	4
Config-4	Asymmetric	0 – 2GB 2 – 4GB 4 – 6GB 6 – 8GB	BI-R11 BI-R11 BS-R21 BS-R21	A, E B, F C, D G, H	4
Config-5	Asymmetric	0 – 2GB 2 – 4GB 4 – 6GB 6 – 8GB	BS-R21 BS-R21 BS-R21 BS-R21	A, B C, D E, F G, H	4
Config-6	Asymmetric	0 – 2GB 2 – 3GB 3 – 4 GB 4– 6GB 6 – 7GB 7 – 8GB	BS-R21 BS-R11 BS-R11 BS-R21 BS-R11 BS-R11	A, B C D E, F G H	6
Config-7	Asymmetric	0 – 2GB 2 – 4GB 4 – 5GB 5 – 6GB 6 – 7GB 7 – 8GB	BI-R21 BI-R21 BS-R11 BS-R11 BS-R11 BS-R11	A, E B, F C D G H	6
Config-8	Symmetric	0 – 1 GB 1 – 2GB 2 – 3GB 3 – 4GB 4 – 5GB 5 – 6GB 6 – 7GB 7 – 8 GB	BS-R11 BS-R11 BS-R11 BS-R11 BS-R11 BS-R11 BS-R11 BS-R11	A B C D E F G H	8

6.5 Chapter Summary

We have quantified the performance and energy for several workloads on four symmetric and six asymmetric, interleaved memory configurations and shown that there are significant opportunities to minimize energy consumption on server systems due to variable slack in memory demand. By transitioning asymmetrically interleaved memory devices into low power states using an adaptive PID controller, we achieved as high as 58 percent energy savings. We found that using power-aware techniques on asymmetrically interleaved memory systems increases the energy efficiency of the memory system across a wide spectrum of application types.

Chapter 7

Conclusions and Future Work

7.1 Conclusions

In this thesis, we have presented new memory management algorithms and techniques, advanced power-state control systems, and proposed memory system architectural adaptations that improve the energy efficiency of high-capacity memory systems commonly used in high-performance computing systems. The goal of our work has been to dynamically scale memory capacity with application demand to reduce operational costs and improve reliability. However, given the complexities of system software and intricacies of high-performance memory architectures, this is a challenging proposition. Our approach has been to weave power-aware techniques into traditional operating system memory management to enable memory devices to transition between multiple power states without impacting application integrity or performance. Building on these mechanisms, we then proposed and incorporated sophisticated control logic from formal control theory to efficiently manage device power-state transitions. After demonstrating the effectiveness of our techniques in sequential memory systems, we then extended our work to highly-complex interleaved memory systems. More specifically, the contributions and findings in this thesis include:

1. **Memory management algorithms for energy efficiency.** To reduce power consumption we extend the operating system to enable memory device power-state transitions without impacting application integrity or performance. We propose

several OS-level page allocation and management *shaping* techniques to proactively and reactively direct allocations to a minimal number of devices. We implement these shaping techniques into a prototype Linux-based operating system and evaluate them on *real* systems. Using a simple heuristic to direct memory device power-state transitions we observe that our techniques yield up to 60% memory energy savings with less than 1% performance loss.

2. **Formal control systems for improving memory energy efficiency** . Efficiently scaling online memory capacity requires a responsive, yet stable control system capable of quickly adapting to rapid changes in memory demand. Our early experiments used inefficient heuristics that required constant retuning. This led us to develop a provably stable, analytic model of a feedback control system from formal control theory. We formally discuss the details of our control-theoretic system as well as an implementation of our control system model. We combine our control system with our prototype operating system into a complete runtime system and compare the energy and performance of a spectrum of workloads using our control system relative to several alternative policies. We found that using our dynamic control system on an 8-node cluster of servers reduced memory energy up to 56.8% with no performance degradation for scientific codes. We achieved memory energy savings of up to 67.94% with no performance degradation for multi-user workloads. Normalizing to total system energy consumption, our power-aware memory approach reduces energy between 18.81% and 39.02%.
3. **Evaluating the power, performance, and thermal efficiency of interleaved memory systems.** Memory interleaving exploits parallelism in the memory system to

improve memory bandwidth and reduce latency. Since interleaving changes memory access patterns, the power and thermal impact on the memory system was unclear. Through extensive experimentation and analysis, we present why the performance-driven, “more is better” interleaved memory design assumption is problematic and should be revisited in future memory designs. Our results indicate that for bandwidth-sensitive benchmarks such as STREAM, memory interleaving in a single dimension improves average bandwidth by 35% and reduces energy consumption by 13% but increases memory temperature by 25%. We also found that further increases in interleaving dimensionality result in little to no performance or energy efficiency gains but still increase temperature nearly 25% for the same codes. For other bandwidth-insensitive benchmarks, we found increasing interleaving dimensionality often does not significantly improve bandwidth or energy efficiency while temperatures increase nearly 25%. The additional heat requires additional cooling, which elevates the cost of the system and can increase chassis energy consumption. Based on our experiments, the effects of interleaving on energy and thermals must be considered in future memory designs.

4. **Improving the energy efficiency of interleaved memory systems.** To improve the energy efficiency of highly-interleaved memory systems, we developed an interleaving-aware control system to dynamically scale memory capacity based on demand and memory bandwidth. We propose new device power-state transition algorithms to reduce memory power within interleaved memory systems by tracking system-wide memory demand as well as bandwidth utilization. We also evaluate the power and performance impact of several novel asymmetric interleaving schemes that

when exploited by our dynamic control system improves Energy Delay Product by up to 58%.

As the memory capacities of systems continue to scale, the need for energy efficient memory systems will become more pronounced. We have shown that our novel techniques can effectively realize significant energy efficiencies with little or no application performance impact, even in complex memory systems.

7.2 Future Work

Although we have presented complete runtime systems that we have evaluated across simple and complex memory topologies, there is still significant work to be completed. Since our system is allocation-based, there may be allocated pages that may not be used for long periods. We plan to study algorithms for identifying and reclaiming allocated, but infrequently accessed pages to further improve energy efficiency. We also plan to study the use of intermediate power states to reduce the overhead of onlining and offlining memory devices and further increase the energy efficient utilization of memory.

Traditionally, memory systems have been aggregated behind a memory controller connected to one or more processors. However, the integration of memory controllers into the processor package has already transformed traditional SMP systems into NUMA systems. Moreover, the proliferation of multi- and many-core processor designs is likely to exacerbate memory access latencies due to on-die interconnect coherency requirements and congestion. In highly integrated, dense many-core systems, the optimal composition and location of the memory system is unclear. We plan to explore the trade-offs in memory system architectures for such many-core processor designs.

The memory systems used in commodity systems have been based on variants of DRAM for a long time. As the access latencies of non-volatile, flash-based storage technologies decrease, this opens the door for multi-level, hybrid memory architectures. The introduction of such hybrid memory systems could have a dramatic impact on the energy efficiency of memory as well as change the face of memory management algorithms. We plan to explore the architectural trade-offs of such hybrid memory systems as well as how memory management could be evolved to take advantage of low-latency persistent storage working in concert with more traditional memory technologies.

Bibliography

[1] "Rambus Inc. RDRAM. http://www.rambus.com," 1999.

[2] "Advanced Configuration and Power Interface (ACPI) Specification 3.0 http://www.acpi.info," 2005.

[3] T. F. Abdelzaher, J. A. Stankovic, C. Lu, R. Zhang, and Y. Lu, "Feedback Performance Control in Software Services," *IEEE Control Systems Magazine*, vol. 23, pp. 74-90, 2003.

[4] N. AbouGhazaleh, B. Childers, D. Mosse, and R. Melhem, "Near-Memory Caching for Improved Energy Consumption," *IEEE Transactions on Computers*, vol. 56, pp. 1441-1455, 2007.

[5] R. L. Adema and C. S. Ellis, "Memory Allocation Constructs to Complement NUMA Memory Management," presented at 3rd IEEE Symposium on Parallel and Distributed Processing, 1991.

[6] N. Aggarwal, J. F. Cantin, M. H. Lipasti, and J. E. Smith, "Power Efficient DRAM Speculation," presented at IEEE International Conference on High Performance Computer Architecture (HPCA), Salt Lake City, UT, 2008.

[7] K. J. Astrom, *Introduction to Stochastic Control Theory*, vol. 70. New York, NY: Academic Press, 1970.

[8] K. J. Astrom and T. Hagglund, *PID Controllers: Theory, Design, and Tuning*, 2nd ed: Instrument Society of America, 1995.

[9] K. J. Astrom and B. Wittenmark, *Adaptive Control*: Adison-Wesley, 1995.

[10] D. H. Bailey, "Performance of Future High-end Computers," in *DOE Mission Computing Conference*, 2003.

[11] L. A. Barroso, K. Gharachorloo, and E. Bugnion, "Memory System Characterization of Commercial Workloads," presented at 25th Annual International Symposium on Computer Archtitecture (ISCA '98), Barcelona, Spain, 1998.

[12] L. A. Barroso and U. Holzle, "The Case for Energy-Proportional Computing," *IEEE Computer*, vol. 40, pp. 33-37, 2007.

[13] F. Baskett and A. J. Smith, "Interference in Multiprocessor Computer Systems with Interleaved Memory " *Communications of the ACM*, vol. 19, pp. 327-334, 1976.

[14] L. Benini, A. Bogliolo, and G. D. Micheli, "A Survey of Design Techniques for System-Level Dynamic Power Management," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 8, pp. 299-316, 2000.

[15] L. Benini, A. Bogliolo, G. A. Paleologo, and G. D. Micheli, "Policy Optimization for Dynamic Power Management," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 18, pp. 813-833, 1999.

[16] L. Benini and G. De Micheli, "System-level power optimization: techniques and tools," *ACM TODAES*, vol. 5, pp. 115-192, 1999.

[17] R. Bianchini and R. Rajamony, "Power and Energy Management for Server Systems," *IEEE Computer*, vol. 37, pp. 68-74, 2004.

[18] P. Bohrer, E. N. Elnozahy, T. Keller, M. Kister, C. Lefurgy, C. Mcdowell, and R. Rajamony, "The Case For Power Management in Web Servers," in *Power Aware*

Computing, R. Graybill and R. Melhem, Eds. IBM Research, Austin TX 78758, USA.: Klewer Academic, 2002.

[19] W. Bolosky, R. Fitzgerald, and M. Scott, "Simple But Effective Techniques for NUMA Memory Management," presented at SOSP-12, 1989.

[20] S. Y. Borkar, "Performance, Power, and the Platform," in *Technology @ Intel Magazine*, 2005.

[21] D. Brooks and M. Martonosi, "Dynamic Thermal Management for High-Performance Microprocessors," presented at 7th Annual International Symposium on High-Performance Computer Architecture, Monterrey, Mexico, 2001.

[22] D. Brooks and M. Martonosi, "Dynamic Thermal Management for High-Performance Microprocessors," presented at the Seventh International Symposium on High-Performance Computer Architecture, Nuevo Leone, Mexico, 2001.

[23] A. R. Butt, C. Gniady, and Y. C. Hu, "The Performance Impact of Kernel Prefetching on Buffer Cache Replacement Algorithms," presented at *SIGMETRICS'05*, Banff, Alberta Canada, 2005.

[24] M. Calzarossa and G. Serazzi, "Workload Characterization: A Survey," *Proceedings of the IEEE*, vol. 81, pp. 1136-1150, 1993.

[25] K. W. Cameron, X. Feng, and R. Ge, "Performance- and Energy-Conscious Distributed DVS Scheduling for Scientific Applications on Power-aware Clusters," presented at 17th High Performance Computing, Networking and Storage Conference (SC 2005), Seattle, WA, 2005.

[26] K. W. Cameron, R. Ge, and X. Feng, "High-Performance, Power-Aware Distributed Computing for Scientific Applications," *IEEE Computer*, vol. 38, pp. 40-47, 2005.

[27] J. Cantin, M. H. Lipasti, and J. E. Smith, "Improving Multiprocessor Performance with Coarse-Grain Coherence Tracking," presented at 32nd International Symposium on Computer Architecture (ISCA), Madison, WI, 2005.

[28] J. Cantin, A. Moshovos, M. H. Lipasti, J. E. Smith, and B. Falsafi, "Coarse-Grain Coherence Tracking: RegionScout and Region Coherence Arrays," *IEEE Micro*, 2006.

[29] R. W. Carr, "Virtual Memory Management," in *Computation Research Group, Stanford Linear Accelerator Center*, vol. Ph.D.: Stanford, 1981.

[30] R. W. Carr and J. L. Hennessey, "WSCLOCK - A Simple and Effective Algorithm for Virtual Memory Management," presented at *SOSP-08*, 1981.

[31] J. Carter, W. Hsieh, L. Stoller, M. Swanson, L. Zhang, E. Brunvand, A. Davis, C.-C. Kuo, R. Kuramkote, M. Parker, L. Schaelicke, and T. Tateyama, "Impulse: Building a Smarter Memory Controller," presented at 5th International Symposium on High Performance Computer Architecture (HPCA-5), Orlando, FL, 1999.

[32] S. Chandra and A. Vahdat, "Application-specific Network Management for Energy-Aware Streaming of Popular Multimedia Formats," presented at USENIX 2002.

[33] P. Chaparro, J. Gonzalez, and A. Gonzalez, "Thermal Effective Clustered Microarchitectures," presented at First Workshop on Temperature-Aware Computer Systems (TACS-1) Held in Conjunction with ISCA, Munich, Germany, 2004.

[34] J. Chase, D. Anderson, P. Thakur, and A. Vahdat, "Managing Energy and Server Resources in Hosting Centers," presented at 18th Symposium on Operating System Principles *(SOSP'01)*, 2001.

[35] G. Chen, K. Malkowski, M. Kandemir, and P. Raghavan, "Reducing Power with Performance Constraints for Parallel Sparse Applications," presented at 1st Workshop on High-Performance Power-Aware Computing, Denver, CO, 2005.
[36] J. Chen and B. N. Bershad, "The Impact of Operating System Structure on Memory System Performance," presented at 14th ACM Symposium on Operating System Principles, Asheville, NC, 1993.
[37] T. F. Chen and J. L. Baer, "Reducing memory latency via non-blocking and prefetching caches," presented at 5th Architectural Support for Programming Languages and Operating Systems (ASPLOS), Boston, MA, 1992.
[38] C.-F. Chiasserini and R. R. Rao, "Improving Battery Performance by Using Traffic Shaping Techniques," *IEEE Journal on Selected Areas in Communications*, vol. 19, pp. 1385-1394, 2001.
[39] F. J. Corbato, "A Paging Experiment with the Multics System," MIT MAC Report MAC-M-384, Boston May 1968.
[40] V. Cuppu, B. Jacob, B. Davis, and T. Mudge, "A Performance Comparison of Contemporary DRAM Architectures," presented at 26th International Symposium on Computer Architecture *(ISCA'99)*, 1999.
[41] M. Debois, C. Scheurich, and F. Briggs, "Memory Access Buffering in Multiprocessors," presented at 13th International Symposium on Computer Architecture, Tokyo, Japan, 1986.
[42] V. Delaluz, M. Kandemir, and I. Kolcu, "Automatic Data Migration for Reducing Energy Consumption in Multi-Bank Memory Systems," presented at Design Automation Conference, New Orleans, 2002.
[43] V. Delaluz, M. Kandemir, N. Vijaykrishnan, A. Sivasubramaniam, and M. J. Irwin, "DRAM Energy Management Using Sof ware and Hardware Directed Power Mode Control," presented at Seventh International Symposium on High-Performance Computer Architecture (HPCA'01), Nuevo Leone, Mexico, 2001.
[44] V. Delaluz, M. Kandemir, N. Vijaykrishnan, A. Sivasubramaniam, and M. J. Irwin, "Hardware and Software Techniques for Controlling DRAM Power Modes," *IEEE Transactions On Computers*, vol. 50, pp. 1154-1173, 2001.
[45] V. Delaluz, A. Sivasubramaniam, M. Kandemir, N. Vijaykrishnan, and M. J. Irwin, "Scheduler Based DRAM Energy Management," presented at 39th Design Automation Conference *(DAC'02)*, 2002.
[46] P. Denning, "Virtual Memory," *ACM Computing Surveys (CSUR)*, vol. 2, pp. 153-189, 1970.
[47] P. Denning, "Working Sets Today," presented at IEEE Computer Software and Applications Conference, 1978.
[48] B. Diniz, D. Guedes, and R. Bianchini, "Limiting the Power Consumption of Main Memory," presented at International Symposium on Computer Architecture (ISCA), San Diego, CA, 2007.
[49] J. Donald and M. Martonosi, "Leveraging Simultaneous Multithreading for Adaptive Thermal Control," presented at Second Workshop on Temperature-Aware Computer Systems (TACS) in conjunction with ISCA-32, Madison, WI, 2005.
[50] F. Douglis, P. Krishnan, and B. N. Bershad, "Adapative Disk Spin-Down Policies for Mobile Computers," presented at 2nd USENIX Symposium on Mobile and Location-Independent Computing, 1995.

[51] T. Economist, "Computing Going Green," 2007.
[52] C. S. Ellis, "The Case for Higher-Level Power Management," presented at The Seventh Workshop on Hot Topics in Operating Systems, Rio Rico, AZ, 1999.
[53] M. Elnozahy, M. Kistler, and R. Rajamony, "Energy Conservation Policies for Web Servers," presented at 4th USENIX Symposium on Internet Technologies and Systems Seattle, WA, 2003.
[54] A. Elwalid and D. Mitra, "Traffic Shaping at a Network Node: Theory, Optimum Design, Admission Control," presented at INFOCOM '97 Sixteenth Annual Joint Conference of the IEEE Computer and Communications Societies, 1997.
[55] EPA, "Report to Congress on Server and Data Center Energy Efficiency (Public Law 109-431)," E. S. P. U.S. Environmental Protection Agency, Ed., 2007.
[56] S. Eranian, "What can performance counters do for memory subsystem analysis?," presented at 2008 Workshop on Memory Systems Performance and Correctness, held in conjuction with ASPLOS '08, Seattle, WA, 2008.
[57] X. Fan, C. Ellis, and A. R. Lebeck, "Modeling of DRAM Power Control Policies Using Deterministic and Stochastic Petri Nets," presented at Workshop on Power-Aware Computer Systems *(PACS '02)*, 2002.
[58] X. Fan, C. S. Ellis, and A. R. Lebeck, "Memory controller policies for DRAM power management," presented at ISPLED, 2001.
[59] X. Fan, C. S. Ellis, and A. R. Lebeck, "Synergy between power-aware memory systems and processor voltage scaling," presented at Workshop on Power-Aware Computing Systems, San Diego, CA, 2003.
[60] W. Felter, K. Rajamani, T. Keller, and C. Rusu, "A Performance-Conserving Approach for Reducing Peak Power Consumption in Server Systems," presented at ACM International Conference on Supercomputing, Boston, MA, 2005.
[61] M. E. Femal and V. W. Freeh, "Boosting Data Center Performance Through Non-Uniform Power Allocation," presented at Second International Conference on Autonomic Computing (ICAC'05), Seattle, WA, 2005.
[62] W. Feng and C. Hsu, "Green Destiny and Its Evolving Parts," presented at 19th International Supercomputer Conference, Heidelberg, Germany, 2004.
[63] X. Feng, R. Ge, and K. W. Cameron, "Power and Energy of Scientific Applications on Distributed Systems," presented at 19th International Parallel and Distributed Processing Symposium (IPDPS 05), Denver, CO, 2005.
[64] K. Flautner and T. Mudge, "Vertigo: Automatic Performance-Setting for Linux," presented at 5th Symposium on Operating System Design and Implementation (OSDI-'05), 2002.
[65] V. Freeh and D. K. Lowenthal, "Using Multiple Energy Gears in MPI Programs on a Power-Scalable Cluster," presented at 10th AMC Symposium on Principles and Practice of Parallel Programming (PPOPP), Chicago, IL, 2005.
[66] J. Fu and J. Patel, "Data prefetching in multiprocessor vector cache memories," presented at 18th International Symposium on Computer Architecture (ISCA), 1991.
[67] R. Ge, X. Feng, and K. W. Cameron, "Improvement of Power-Performance Efficiency for High-End Computing," presented at 19th IEEE/ACM International Parallel and Distributed Processing Symposium (IPDPS), 2005.

[68] M. Ghosh and H.-H. S. Lee, "Smart Refresh: An Enhanced Memory Controller Design for Reducing Energy in Conventional and 3D Die-Stacked DRAMs," presented at 40th Annual IEEE/ACM International Symposium on Microarchitecture Chicago, IL, 2007.

[69] R. Gonzalez and M. Horowitz, "Energy Dissipation in General Purpose Microprocessors," *IEEE Journal of Solid-State Circuits*, vol. 31, pp. 1277-1284, 1996.

[70] M. Gorman, *Understanding the Linux Virtual Memory Manager*. New Jersey: Prentice Hall, 2004.

[71] G. Grohoski, "Niagra2: A Highly-Threaded Server-on-A-Chip," presented at HotChips 18, 2006.

[72] J. Haas and P. Vogt, "Fully-buffered DIMM Technology Moves Enterprise Platforms to the Next Level," in *Technology@Intel Magazine*, vol. 3, 2005.

[73] S. Hand, "Self-paging in the Nemesis Operating System," presented at OSDI-3, 1999.

[74] T. Heath, A. P. Centeno, P. George, L. Ramos, Y. Jaluria, and R. Bianchini, "Mercury and Freon: Temperature Emulation and Management for Server Systems," presented at Architectural Support for Programming Languages and Operating Systems (ASPLOS), San Jose, CA, 2006.

[75] D. Helmbold, D. Long, and E. Sherrod, "Dynamic Disk Spin-Down Techniques for Mobile Computing," presented at IEEE Conference on Mobile Computing, 1996.

[76] J. L. Henning, "SPEC CPU2006 Memory Footprint," *Computer Architecture News*, vol. 35, pp. 84-89, 2007.

[77] H. Hjalmarsson, M. Gevers, S. Gunnarsson, and O. Lequin, "Iterative Feedback Tuning: Theory and Applications," *IEEE Control Systems Magazine*, vol. 18, pp. 26-41, 1998.

[78] C. V. Hollot, A. Misra, D. Towsley, and W.-B. Gong, "On Designing Improved Controllers for AQM Routers Supporting TCP Flows," presented at IEEE INFOCOM, Anchorage, Alaska, 2001.

[79] C. V. Hollot, V. Misra, D. Towsley, and W.-B. Gong, "A Control-Theoretic Analysis of RED," presented at INFOCOM 2001. Twentieth Annual Joint Conference of the IEEE Computer and Communication Societies, Anchorage, AK, 2001.

[80] J. Hom and U. Kremer, "Energy Management of Virtual Memory on Diskless Devices," presented at Workshop on Compilers and Operating Systems for Low Power *(COLP'01)*, 2001.

[81] T. R. Hotchkiss, N. D. Marschke, and R. M. McClosky, "A New Memory System Design for Commercial and Technical Computing Products," *Hewlett Packard Journal*, 1996.

[82] C.-H. Hsu and U. Kremer, "Compiler-directed dynamic voltage scaling for memory-bound applications," Department of Computer Science Rutgers University, Piscataway August 2002.

[83] C.-H. Hsu and U. Kremer, "The design, implementation, and evaluation of a compiler algorithm for CPU energy reduction," presented at ACM SIGPLAN Conference on Programming Languages, Design, and Implementation (PLDI'03), San Diego, CA, 2003.

[84] W. C. Hsu and J. E. smith, "Performance of Cached DRAM Organizations in Vector Supercomputers," presented at 20th Annual International Symposium on Computer Architecture, 1993.

[85] Z. Hu, S. Kaxiras, and M. Martonosi, "Timekeeping in the Memory System: Predicting and Optimizing Memory Behavior," presented at 29th Annual International Symposium on Computer Architecture (ISCA), Anchorage, Alaska, 2002.

[86] H. Huang, W. Hung, and K. G. Shin, "FS2: Dynamic Data Replication in Free Disk Space for Improving Disk Performance and Energy Consumption," presented at International ACM Symposium on Operating System Principles *(SOSP'05)*, Brighton, UK, 2005.
[87] H. Huang, C. Lefurgy, T. Keller, and K. G. Shin, "Memory Traffic Reshaping for Energy-Efficient Memory," presented at International Symposium on Low Power Electronics and Design *(ISLPED)*, San Diego, CA, 2005.
[88] H. Huang, P. Pillai, and K. Shin, "Design and Implementation of Power-aware Virtual Memory," presented at Usenix 2003 Annual Technical Conference, 2003.
[89] H. Huang, K. G. Shin, C. Lefurgy, and T. Keller, "Improving Energy Efficiency by Making DRAM Less Randomly Accessed," presented at International Symposium on Low Power Electronics and Design, San Diego, CA, 2005.
[90] H. Huang, K. G. Shin, C. Lefurgy, K. Rajamani, T. Keller, E. V. Hensbergen, and F. Rawson, "Cooperative Software-Hardware Power Management for Main Memory," presented at Workshop on Power-Aware Computer Systems, Portland, OR, 2004.
[91] M. Huang, J. Renau, S.-M. Yoo, and J. Torrellas, "A Framework for Dynamic Energy Efficiency and Temperature Management," presented at 33rd annual ACM/IEEE international symposium on Microarchitecture Monterey, CA, 2000.
[92] I. Hur and C. Lin, "A Comprehensive Approach to DRAM Power Management," presented at IEEE International Conference on High Performance Computer Architecture (HPCA), Salt Lake City, UT, 2008.
[93] C.-H. Hwang and A. C.-H. Wu, "A Predictive System Shutdown Method for Energy Saving of Event-Driven Computation," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 5, pp. 226-241, 2000.
[94] Intel, "Intel E8500 Chipset External Memory Bridge (XMB) Datasheet," vol. 2005, 2005.
[95] Intel, "Quad-Core Intel Xeon Processor 5400 Series Datasheet," 2007.
[96] S. Irani, S. Shukla, and R. Gupta, "Competitive Analysis of Dynamic Power Management Strategies for Systems with Multiple Power Saving States," presented at Conference on Design, Automation, and Test in Europe (DATE), 2002.
[97] C. Isci and M. Martonosi, "Runtime Power Monitoring in High-End Processors: Methodology and Empirical Data," presented at 36th International Symposium on Microarchitecture (MICRO-36), San Diego, CA, 2003.
[98] C. Isci, M. Martonosi, and A. Buyuktosunoglu, "Long-term Workload Phases: Duration Predictions and Applications to DVFS," *IEEE Micro*, vol. 25, pp. 39-51, 2005.
[99] B. Jacob and T. Mudge, "Virtual Memory in Contemporary Microprocessors," *IEEE Micro*, vol. 18, pp. 60-75, 1998.
[100] B. Jacob, S. Ng, and D. Wang, *Memory Systems: Cache, DRAM, Disk*: Morgan Kaufmann, 2007.
[101] S. Jiang, F. Chen, and X. Zhang, "CLOCK-Pro: An Effective Improvement of the CLOCK Replacement," presented at USENIX ATC, 2005.
[102] F. Jones, B. Prince, R. Norwood, J. Hartigan, W. C. Vogley, C. A. Hart, and D. Bondurant, "A New Era of Fast Dynamic RAMs," *IEEE Spectrum*, vol. 29, pp. 43-49, 1992.
[103] R. Joseph, D. Brooks, and M. Martonosi, "Control Techniques to Eliminate Voltage Emergencies in High Performance Processors," presented at the 9th International

Symposium on High Performance Computer Architecture (HPCA-9), Anaheim, CA, 2003.
[104] N. Jouppi, "Improving Direct-Mapped Cache Performance by the Addition of a Fully Associative Cache and Prefetch Buffers," presented at 17th International Symposium on Computer Architecture (ISCA), Seattle, WA, 1990.
[105] K. Kant and Y. Won, "Server Capacity Planning for Web Traffic Workload," *IEEE Transactions on Knowledge and Data Engineering*, vol. 11, pp. 731-747, 1999.
[106] L. S. Kaplan, "A Flexible Interleaved Memory Design for Generalized Low Conflict Memory Access," presented at the 6th Distributed Memory Computing Conference, 1991.
[107] N. Kappiah, V. W. Freeh, D. K. Lowenthal, and F. Pan, "Exploiting Slack Time in Power-Aware High-Performance Programs," presented at IEEE/ACM Supercomputing 2005 (SC | 05), Seattle, WA, 2005.
[108] C. Karamanolis, M. Karlsson, and X. Zhu, "Designing Controllable Computer Systems," presented at 10th conference on Hot Topics in Operating systems, Santa Fe, NM, 2005.
[109] D. E. Kirk, *Optimal Control Theory*. Englewood Cliffs, NJ: Prentice Hall, 1970.
[110] J. G. Koomey, "Estimating Total Power Consumption By Servers in the U.S. and the World," *Final Report*, 2007.
[111] C. E. Kozyrakis, S. Perissakis, D. Patterson, T. Anderson, K. Asanovic, N. Cardwell, R. Fromm, J. Golbus, B. Gribstad, K. Keeton, R. Thomas, N. Treuhaft, and K. Yelick, "Scalable Processors in the Billion-Transistor Era: IRAM," *IEEE Computer*, vol. 30, pp. 75-78, 1997.
[112] B. Kristiansson and B. Lennartson, "Robust Tuning of PI and PID Controllers," *IEEE Control Systems Magazine*, pp. 55-69, 2006.
[113] A. R. Lebeck, X. Fan, H. Zeng, and C. Ellis, "Power Aware Page Allocation," presented at ASPLOS-IX, 2002.
[114] C. Lefurgy, K. Rajamani, F. Rawson, W. Felter, M. Kistler, and T. Keller, "Energy Management for Commercial Servers," *IEEE Computer*, vol. 36, pp. 39-48, 2003.
[115] C. Lefurgy, X. Wang, and M. Ware, "Server-level Power Control," presented at Fourth International Conference on Autonomic Computing (ICAC'07), Jacksonville, FL 2007.
[116] W. S. Levine, *Control System Fundamentals*. Boca Raton, FL: CRC Press, 1999.
[117] X. Li, Z. Li, F. Danvid, Y. Zhou, and S. Kumar, "Performance Directed Energy Management for Main Memory and Disks," presented at ASPLOS '04, Boston, 2004.
[118] X. Li, Z. Li, P. Zhou, Y. Zhou, S. Adve, and S. Kumar, "Performance Directed Energy Management for Storage Systems," *IEEE Micro*, vol. 24, pp. 38-49, 2004.
[119] J. Lin, H. Zheng, Z. Zhu, H. David, and Z. Zhang, "Thermal Modeling and Management of DRAM Memory Systems," presented at International Symposium on Computer Architecture (ISCA), San Diego, CA, 2007.
[120] Y.-H. Lu, L. Benini, and G. D. Micheli, "Operating-System Directed Power Reduction," presented at International Symposium on Low Power Electronics and Design (ISPLED), Rapallo, Italy, 2000.
[121] Y.-H. Lu, L. Benini, and G. D. Micheli, "Power-Aware Operating Systems for Interactive Systems," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 10, 2002.

[122] W. L. Lynch, B. K. Bray, and M. J. Flynn, "The Effect of Page Allocation on Caches," presented at 25th Annual International Symposium on Microarchitecture Portland, OR, 1992.

[123] N. Magen, A. Kolodny, U. Weiser, and N. Shamir, "Interconnect-Power Dissipation in a Microprocessor," presented at International Sorkshop on System Level Interconnect Prediction (SLIP '04), Paris, France, 2004.

[124] R. McDougall and J. Mauro, *Solaris Internals*. Menlo Park, CA: Sun Microsystems Press, 2001.

[125] S. A. McKee, A. Aluwihare, B. H. Clark, R. H. Klenke, T. C. Landon, C. W. Oliver, M. H. Salinas, A. E. Szymkowiak, K. L. Wright, W. A. Wulf, and J. H. Aylor, "Design amd Evaluation of Dynamic Access Ordering Hardware," presented at 10th International Conference on Supercomputing, Philadelphia, PA, 1996.

[126] V. Misra, W.-B. Gong, and D. Towsley, "Fluid-Based Analysis of a Network of AQM Routers Supporting TCP Flows with an Application to RED," presented at SIGCOMM, Stockholm, Sweden, 2000.

[127] T. Mudge, "Power: A First Class Design Constraint for Future Architectures," presented at High Performance Computer Conference (HiPC), Boston, MA, 2000.

[128] K. S. Narenda and A. M. Annaswamy, *Stable Adaptive Systems*. Englewood Cliffs, NJ: Prentice-Hall, 1989.

[129] C. Natarajan, B. Christenson, and F. Briggs, "A Study of Performance Impact of Memory Controller Features in Multi-Processor Server Environment," presented at Workshop on Memory Performance Issues, Munich, Germany, 2004.

[130] A. Pajuelo, A. Gonzalez, and M. Valero, "Speculative Execution for Hiding Memory Latency," *ACM SIGARCH Computer Architecture News*, vol. 33, pp. 49-56, 2005.

[131] V. Pandey, W. Jiang, Y. Zhou, and R. Bianchini, "DMA-Aware Memory Energy Management," presented at 12th International Symposium on High-Performance Computer Architecture *(HPCA '06)*, Austin, TX, 2006.

[132] S. Park, W. Jiang, S. Adve, and Y. Zhou, "Managing Energy-Performance Tradeoffs for Multithreaded Applications on Multiprocessor Architectures," presented at International Conference on Measurement and Modeling of Computer Systems, San Diego, CA, 2007.

[133] D. A. Patterson and J. L. Hennessey, *Computer Architecture: A Quantitative Approach*, 2nd ed. San Francisco, CA: Morgan Kaufmann Publishers, 1996.

[134] J. Pisharath, A. Choudhary, and M. Kandemir, "Energy Management Schemes for Memory-Resident Database Systems," presented at ACM International Conference on Information and Knowledge Management, Washington D.C., 2004.

[135] M. D. Powell, M. Gomaa, and N. Vijaykrishnan, "Heat-and-Run: Leveraging SMT and CMP to Manage Power Density Through the Operating System," presented at 11th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), Boston, MA, 2004.

[136] J. Quarterman, A. Silberschatz, and J. Peterson, "4.2BSD and 4.3BSD as Examples of the UNIX System," *Computing Surveys*, vol. 17, pp. 379-418, 1985.

[137] S. Radhakrishnan, S. Chinthamani, and K. Cheng, "The Blackford Northbridge Chipset for the Intel 5000," *IEEE Micro*, vol. 27, pp. 22-32, 2007.

[138] R. Raghavendra, P. Ranganathan, V. Talwar, Z. Wang, and X. Zhu, "No "Power" Struggles: Coordinated Multi-level Power Management for the Data Center," presented at

Architectural Support for Programming Languages and Operating Systems (ASPLOS), Seattle, WA, 2008.

[139] L. Ramos and R. Bianchini, "C-Oracle: Predictive Thermal Management for Data Centers," presented at IEEE International Conference on High Performance Computer Architecture (HPCA), Salt Lake City, UT, 2008.

[140] R. Rashid, A. Tevanian, M. Young, D. Golub, R. Baron, D. Black, W. Bolosky, and J. Chew, "Machine-Independent Virtual Memory Management for Paged Uniprocessor and Multiprocessor Architectures," presented at ASPLOS '87, 1987.

[141] E. Rotenberg and R. K. Venkatesan, "The State of ZettaRAM," presented at 1st IEEE International Conference on Nano-Networks, Lausanne, Switzerland, 2006.

[142] J. Shao and B. T. Davis, "The Bit-Reversal SRAM Address Mapping," presented at Workshop on Software and Compilers for Embedded Systems, Dallas, TX, 2005.

[143] K. Skadron, T. Abdelzaher, and M. R. Stan, "Control-Theoretic Techniques and Thermal-RC Modeling for Accurate and Localized Dynamic Thermal Management," presented at International Symposium on High-Performance Computer Architecture (HPCA), 2002.

[144] K. Skadron, M. R. Stan, W. Huang, and S. Velusamy, "Temperature-Aware Microarchitecture," presented at 30th Annual International Symposium on Computer Architecture (ISCA '03), San Diego, CA, 2003.

[145] A. Smith, "Cache Memories," *ACM Computing Surveys (CSUR)*, vol. 14, pp. 473-530, 1982.

[146] SPEC-Subcommittee, "SPEC CPU2006 Benchmark Descriptions," *ACM SIGARCH Computer Architecture News*, vol. 34, 2006.

[147] L. Spracklen and S. G. Abraham, "Chip Multithreading: Opportunities and Challenges," presented at 11th International Symposium on High-Performance Computer Architecture (HPCA-11), San Francisco, CA, 2005.

[148] M. B. Srivastava, A. P. Chandrakasan, and R. W. Brodersen, "Predictive System Shutdown and Other Architectural Techniques for Energy Efficient Programmable Computation," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 3, pp. 42-55, 1996.

[149] M. A. Suleman, M. K. Qureshi, and Y. N. Patt, "Feedback-Driven Threading: Power-Efficient and High-Performance Execution of Multi-threaded Workloads on CMPs," presented at Architectural Support for Programming Languages and Operating Systems, Seattle, WA, 2008.

[150] M. E. Tolentino, "Flexible Operating System Structure for Dynamic Memory Management," in *Computer Science*, vol. M.S.: University of Washington, 2004, pp. 88.

[151] M. E. Tolentino, J. Turner, and K. W. Cameron, "An Implementation of Page Allocation Shaping for Energy Efficiency," presented at 3rd Workshop on High-Performance, Power-Aware Computing, Long Beach, CA, 2007.

[152] M. E. Tolentino, J. Turner, and K. W. Cameron, "Memory-MISER: A Performance-Constrained Runtime System for Power-Scalable Clusters," presented at ACM International Conference on Computing Frontiers, Ischia, Italy, 2007.

[153] M. E. Tolentino, J. Turner, and K. W. Cameron, "Memory-MISER: Improving Main Memory Energy Efficiency in Servers," *IEEE Transactions on Computers*, vol. 58, 2009.

[154] J. Tse and A. J. Smith, "CPU Cache Prefetching: Timing Evaluation of Hardware Implementations," *IEEE Transactions on Computers*, vol. 47, pp. 509-526, 1998.

[155] A. Vahdat, A. Lebeck, and C. S. Ellis, "Every Joule is Precious: The Case for Revisiting Operating System Design for Energy Efficiency," presented at 9th ACM SIGOPS European Workshop, Kolding, Denmark, 2000.

[156] A. Varma, B. Ganesh, M. Sen, S. R. Choidhury, L. Srinivasan, and B. Jacob, "A Control-Theoretic Approach to Dynamic Voltage Scheduling," presented at International Conference on Compilers, Architectures, and Synthesis for Embedded Systems (CASES), San Jose, CA, 2003.

[157] R. K. Venkatesan, A. S. AL-Zawawi, and E. Rotenberg, "Tapping ZettaRAM for Low-Power Memory Systems," presented at 11th International Symposium on High-Performance Computer Architecture (HPCA-11) San Francisco, CA, 2005.

[158] N. Vijaykrishnan, M. Kandemir, M. J. Irwin, H. S. Kim, and W. Ye, "Energy-Driven Integrated Hardware-Software Optimizations Using SimplePower," presented at 27th Annual International Symposium on Computer Architecture, Vancouver, British Columbia, Canada, 2000.

[159] M. Vilayannur, A. Sivasubramaniam, and M. Kandemir, "Pro-active Page Replacement for Scientific Applications: A Characterization," presented at IEEE International Symposium on Performance Analysis of Systems and Software, Austin, TX, 2005.

[160] E. Waingold, M. Taylor, D. Srikrishna, V. Sarkar, W. Lee, V. Lee, J. Kim, M. Frank, P. Finch, R. Barua, J. Babb, S. Amarasinghe, and A. Agarwal, "Baring it All to Software: Raw Machines," *IEEE Computer*, vol. 30, pp. 86-93, 1997.

[161] C. A. Waldspurger, "Memory Resource Management in VMware ESX Server," presented at 5th Symposium on Operating Systems Design and Implementation (OSDI '02), Boston, MA, 2002.

[162] M. Wang, N. Kandasamy, A. Guez, and M. Kam, "Adaptive Performance Control of Computing Systems via Distributed Cooperative Control: Application to Power Management in Computing Clusters," presented at 3rd International Conference on Autonomic Computing (ICAC '06), Dubling, Ireland, 2006.

[163] M. Wang, N. Kandasamy, A. Guez, and M. Kam, "Distributed Cooperative Control for Adaptive Performance Management," *IEEE Internet Computing*, vol. 11, pp. 31-39, 2007.

[164] Q. Wu, P. Juang, M. Martonosi, and D. W. Clark, "Formal Online Methods for Voltage/Frequency Control in Multiple Clock Domain Multiprocessors," presented at Architectural Support for Programming Languages and Operating Systems (ASPLOS), Boston, MA, 2004.

[165] Q. Wu, P. Juang, M. Martonosi, L.-S. Peh, and D. W. Clark, "Formal Control Techniques for Power-Performance Management," *IEEE Micro*, vol. 25, pp. 52-62, 2005.

[166] W. A. Wulf and S. A. McKee, "Hitting the Memory Wall: Implications of the Obvious," *ACM SIGARCH Computer Architecture News*, vol. 23, pp. 20-24, 1995.

[167] H. Zeng, C. S. Ellis, A. R. Lebeck, and A. Vahdat, "ECOSystem: Managing Energy as a First Class Operating System Resource," presented at Architectural Support for Programming Languages and Operating Systems (ASPLOS X), San Jose, CA, 2002.

[168] H. Zeng, C. S. Ellis, A. R. Lebeck, and A. Vahdat, "Currentcy: A Unifying Abstraction for Expressing Energy Management Policies," presented at Annual Conference on USENIX Annual Technical Conference, San Antonio, TX, 2003.

[169] X. Zhang, A. Dasdan, M. Schulz, R. K. Gupta, and A. A. Chien, "Architectural Adaptation for Application-Specific Locality Optimizations," presented at International Conference on Computer Design (ICCD), Austin, TX, 1997.

[170] Z. Zhang, Z. Zhu, and X. Zhang, "A Permutation-Based Page Interleaving Scheme to Reduce Row-Buffer Conflicts and Exploit Data Locality," presented at 33rd Annual International Symposium on Microarchitecture Monterrey, CA, 2000.

[171] P. Zhou, V. Pandey, J. Sundaresan, A. Raghuraman, Y. Zhou, and S. Kumar, "Dynamic Tracking of Page Miss Ratio Curve for Memory Management," presented at ASPLOS '04, Boston, 2004.

[172] Q. Zhu, Z. Chen, L. Tan, Y. Zhou, K. Keeton, and J. Wilkes, "Hibernator: Helping Disk Arrays Sleep Through the Winter," presented at 20th ACM Symposium on Operating Systems Principles *(SOSP '05)*, Brighton, UK, 2005.

[173] Z. Zhu and Z. Zhang, "A Performance Comparison of DRAM Memory System Optimizations for SMT Processors " presented at 11th International Symposium on High-Performance Computer Architecture (HPCA'05), San Francisco, CA, 2005.

[174] B. Zorn and D. Grunwald, "Evaluating Models of Memory Allocation," *ACM Transactions on Modeling and Computer Simulation*, vol. 4, pp. 107-131, 1994.