DATA PLACEMENT FOR EFFICIENT MAIN MEMORY ACCESS

by

Kshitij Sudan

A dissertation submitted to the faculty of
The University of Utah
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

in

Computer Science

School of Computing

The University of Utah

May 2013

The University of Utah Graduate School

STATEMENT OF DISSERTATION APPROVAL

The dissertation of **Kshitij Sudan**

has been approved by the following supervisory committee members:

Rajeev Balasubramonian	, Chair	**10/05/2012** Date Approved
John B. Carter	, Member	**10/24/2012** Date Approved
Alan L. Davis	, Member	**10/25/2012** Date Approved
Ashutosh Dhodapkar	, Member	Date Approved
Mary W. Hall	, Member	**10/26/2012** Date Approved

and by **Alan L. Davis**, Chair of the Department of **School of Computing**

and by Donna M. White, Interim Dean of The Graduate School.

ABSTRACT

The main memory system is a critical component of modern computer systems. Dynamic Random Access Memory (DRAM) based memory designs dominate the industry due to mature device technology and low cost. These designs, however, face several challenges moving forward. These challenges arise due to legacy DRAM device design choices, advances in Central Processing Unit (CPU) design, and the demand for higher memory throughput and capacity from applications. Due to the cost-sensitive nature of the DRAM industry, changes to the device architecture face significant challenges for adoption. There is thus a need to improve memory system designs, ideally without changing the DRAM device architectures.

This dissertation addresses the challenges faced by DRAM memory systems by leveraging data management. Historically, data management/placement and its interaction with the memory's hardware characteristics have been abstracted away at the system software level. In this dissertation, we describe mechanisms that leverage data placement at the operating system level to improve memory access latency, power/energy efficiency, and capacity. An important advantage of using these schemes is that they require no changes to the DRAM devices and only minor changes to the memory controller hardware. The majority of the changes are limited to the operating system. This thesis also explores data management mechanisms for future memory systems built using new 3D stacked DRAM devices and point-to-point interconnects.

Using the schemes described here, we show improvements in various DRAM metrics. We improve DRAM row-buffer hit rates by co-locating parts of different Operating System (OS) pages in the same row-buffer. This improves performance by 9% and reduces energy consumption by 15%. We also improve page placement to increase opportunities for power-down. This enables a three-fold increase in memory capacity for a given memory power budget. We also show that page placement is an important ingredient in building efficient networks of memories with 3D-stacked memory devices. We report a performance improvement of 49% and an energy reduction of 42% with a design that optimizes page placement and network topology.

To my parents.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

First and foremost, I would like to thank Rajeev for being a great advisor. I have learned a lot from him, not just about computer architecture, but more importantly, about being a good researcher. I always admire his quick-thinking abilities and the ease with which he takes my jargon-heavy text (which also misses all the articles!), and transforms it into an easily readable and accessible text. I would also like to thank my committee members John Carter, Al Davis, Ashutosh Dhodapkar, and Mary Hall. I have had the pleasure of working extensively with all my committee members. John was my mentor when I started graduate school, and then again while I interned at IBM-Austin. I want to thank him for all the opportunities and for helping me understand the "big-picture" view of systems research. Al has given me wonderful feedback on my projects and I've always learned something new in every conversation with him. Ashutosh mentored me when I interned at SeaMicro and Violin Memory. He is primarily the reason I got interested in ways to optimize datacenter platforms, and he taught me a lot about systems engineering. Mary was instrumental in helping me understand the challenges of hardware-software co-optimizations.

I would like to thank Karthick Rajamani for being an awesome mentor while I interned at IBM-Austin. I am also thankful to Wei Huang and Freeman Rawson for their patience and all the help while I was at IBM. I want to thank Sadagopan Srinivasan and Ravi Iyer for being great collaborators. Dave Nellans not only was a lab-rat with me at Utah, but he also mentored me during my internship at Fusion-io. I had a great time working with him and thank him for always listening to me patiently. Saisantosh Balakrishnan was not only a collaborator during my internships at SeaMicro and Violin, but also a good friend. He taught me a great deal about good coffee, and thanks to him, I now run on quality caffeine!

My colleagues at the Utah-Arch group were simply superb, and five years with them went by in a blink. I want to thank Niti, Karthik, Vivek, Manu, Aniruddha, Niladrish, Seth, Manju, Aasheesh, and Ali for all the great discussions in the lab, and their friendship. I owe a big thanks to my friends - Rahul, Abhishek, Guarav, and Shekhar - for patiently listening to my grad school gripes. I also want to thank Chelsea for her great company and great times over the past five years.

Finally, I would like to thank my parents and my brother. I would not have been able to achieve anything without them being by my side all along. Without your inspiration and encouragement, I would not have been the person I am today.

CHAPTER 1

INTRODUCTION

1.1 Emerging Memory System Challenges

Main memory access is a primary bottleneck in modern computing systems. Memory access latency directly impacts system performance and therefore is a critical design parameter. While access latency has been a dominant design parameter for multiple decades, the memory capacity and power/energy consumption have lately emerged as first-order design goals as well.

Memory systems are typically built using the Dynamic Random Access Memory (DRAM) technology. The Joint Electron Devices Engineering Council (JEDEC) [63] standards for a main memory describe a system-level design using commodity DRAM devices. The DRAM devices themselves improve as per Moore's Law for improvements in the manufacturing process. However, system-level design choices limit the ability to reduce memory system latency, increase capacity, and reduce energy consumption at the same rate. The reasons for this smaller rate of improvement are described next.

- **Memory Access Latency** - DRAM access latency depends on the DRAM device architecture, the memory channel frequency, and the memory access pattern. DRAM device access latency has not changed significantly over the past three generations of JEDEC standards because these latencies are limited by delays for global wires that have not reduced. The data rate (and hence the total system bandwidth), however, has increased significantly to feed the increased memory traffic from multicore Central Processing Units (CPUs). Multicore CPUs, however, have significantly changed the memory access pattern. Each core on the CPU socket generates an independent stream of memory traffic, and these independent streams end up being intertwined at the memory controller. This leads to a loss of locality in the aggregate DRAM access pattern, resulting in higher row-buffer conflicts and thus higher access latency. As a consequence, a large number of commercial servers end up using a close-page row-buffer policy due to an insignificant reduction in latency with an open-page policy.

- **Capacity and Bandwidth** - The growth in memory capacity and bandwidth is primarily limited by the power consumption and poor signal integrity due to high channel frequency. The contribution of the memory system to overall system power has increased steadily. It is well known that the memory system alone can consume up to a third of the total system power in modern servers [76]. To increase capacity further would mean increasing the memory power budget, which might not be feasible due to limits on the power delivery and cooling networks for the servers. The memory channel frequency also has a direct impact on capacity. Channel frequency has increased with every generation of JEDEC standards to increase the system bandwidth. This, however, comes at the cost of reduced capacity per channel due to poor signal integrity associated with high-frequency, bus-based channels. At data rates of 400 MHz, a memory channel could support up to four Dual In-Line Memory Modules (DIMMs) per channel; however, at 1066 MHz, only 2 DIMMs per Channel (DPC) can be achieved. This severely limits the capacity of the memory system. Note that with every process technology generation, the DRAM die capacity increases, but with reduced DPC and increasing demand from the applications and more cores per CPU socket, the memory system becomes more capacity and bandwidth constrained.

- **Energy Consumption** - Energy consumed per memory access quantifies the memory access efficiency. Energy is consumed by the DRAM core access, the peripheral circuitry on the DRAM device, and finally the energy needed to drive the channel. With every generation of JEDEC standards, the memory access efficiency has been decreasing. The access granularity (the granularity at which the CPU accesses the memory) has remained constant at 64 bytes. However, with increasing DRAM capacity, the DRAM devices have increased the number of DRAM columns and rows. These columns read large chunks of data in DRAM cells, a very small fraction of which are used while servicing a single memory request. The resulting over-fetch of data increases the energy per request. Typically, for every 64 byte request from the CPU, the DRAM row-buffers fetch up to 8 KB of data. This implies each row-buffer fetch has a utilization of less than 1% for a single memory request with the closed-page row-buffer policy. This over-fetch not only adds to access latency, but wastes energy.

1.2 Data Placement to Improve Memory Accesses

This dissertation looks at how data placement can help improve memory access latency, power/energy consumption, bandwidth, and capacity. Data placement has often been

ignored as a tool that can be used to make memory accesses more efficient. This is perhaps an artifact of the fact that data placement is typically closely tied to, or implemented at, the system software. For example, the Operating System (OS) is responsible for allocating frames in the main memory to pages being read from the disk. This is implemented as part of the virtual memory subsystem of an OS where the priority is to optimize efficient memory capacity utilization. The hardware aspects of the memory access are often abstracted away at this level and as a result, OS designers do not necessarily optimize for efficient hardware access. We next discuss three data placement mechanisms that aim at improving the memory access latency, power/energy, bandwidth, and capacity.

- **Improving Access Latency and Energy** - With growing CPU cores per socket, DRAM-based memory subsystems are starting to show the negative impact of legacy hardware design choices. Current designs were based on assumptions such as relatively stable memory access patterns, i.e., a single core exhibits temporal and spatial locality in its DRAM access pattern. As a result, large DRAM row-buffers along with an open-page policy were suitable for single-core CPUs. This design choice is now emerging as a bottleneck because it takes a considerable amount of energy and time to read these large row-buffers. With multicore CPUs, the open-page policy is not able to exploit locality in the data access pattern of a single thread due to interfering requests from other cores. In some cases, data have been mapped to DRAM such that they are striped at cache line granularity to improve available bandwidth. This mapping leads to poor row-buffer utilization due to conflicting requests. Intelligent data placement helps improve row-buffer reuse, leading to reduced access time and reduction in energy consumption. This is achieved by coalescing frequently used data from the most accessed OS page in the same DRAM row-buffer. This coalescing increases row-buffer hit rates even with increasingly interfering memory request streams due to multicore CPUs.

- **Increasing Capacity in a Fixed Memory Power Budget** - The fraction of total system power that is consumed by the main memory is increasing with every generation because of higher signaling rates, and more capacity per system. However, high-end systems, especially those used for workload consolidation, require even more memory than currently available at nominal power budgets. At the same time, the various low-power DRAM modes that are supported by JEDEC-based DRAM memories are hard to exploit. These power modes are applied at a coarse rank-level granularity. Since there is a latency overhead for transitioning in or out of these

modes, it is difficult to leverage them for significant power savings because of current data placement choices and the resulting access patterns. Frequently accessed data are often striped across multiple ranks, thus making it very hard to leverage deep low-power modes.

- **Data Placement for Future Memory Systems** - With each technology generation, memory channels are being clocked at higher rates to improve aggregate memory system throughput. This trend, however, is not sustainable due to signal integrity issues, and limits on power consumption. To overcome these challenges, serial, point-to-point links are seen as the next step in implementing the CPU-memory interconnect. Serial, point-to-point channels can be used to construct a network of memories on a motherboard. Such a network will offer nonuniform latency and energy when accessing different pages. Smart data placement policies can help improve access latency by bringing frequently accessed data to memory devices that are reachable within a few hops.

1.3 Thesis Statement

The dissertation tests the hypothesis that intelligent data placement policies can improve memory access time and energy; it shows three different ideas and settings where careful data placement improves memory behavior by optimizing row-buffer hit rates, rank idle periods, or network hop counts.

1.4 Thesis Organization

This dissertation is organized into six chapters. Chapter 2 provides background by describing DRAM-based memory systems in detail. Each subsequent chapter describes mechanisms to improve memory access latency, power/energy consumption, and capacity by leveraging data management techniques. Chapter 3 describes techniques to consolidate frequently used data in the same DRAM row-buffers, leading to reduced latency and increased energy efficiency for each access. Chapter 4 describes schemes to consolidate active data in few DRAM ranks, thereby providing an opportunity to increase capacity in a fixed power budget by powering down rarely used ranks. Chapter 5 describes a memory system design that builds a network-of-memories to increase capacity. The access latency in such systems depends on the location of data (whether it is a few hops away on the network, or many hops away). Data placement is leveraged for this design to bring frequently accessed

data closer to the CPU, leading to faster accesses and opportunities to power down distant memory nodes. Chapter 6 finally summarizes the conclusions and describes future work.

CHAPTER 2

DRAM BACKGROUND

In this chapter, we describe the organization and operation of modern DRAM-based main memory systems built following the JEDEC memory standard. We also describe the Input/Output (I/O) link technology used for typical DRAM channel designs.

2.1 DRAM Basics

Modern Double Data Rate (DDR) systems [59] are organized as modules (DIMMs), composed of multiple devices, each of which is an individually packaged integrated circuit. The DIMMs are connected to the memory controller via a data bus and other command and control networks (Figure 2.1). DRAM accesses are initiated by the CPU requesting a cache line worth of data in the event of a last-level cache miss. The request shows up at the memory controller, which converts the request into carefully orchestrated commands for DRAM access. Modern memory controllers are also responsible for scheduling memory requests according to policies designed to reduce access times for a request. Using the physical address of the request, the memory controller typically first selects the channel, and then a rank. Within a rank, DRAM devices work in unison to return as many bits of data as the width of the channel connecting the memory controller and the DIMM.

Each DRAM device is arranged as multiple banks of mats - a grid-like array of cells. Each cell comprises a transistor-capacitor pair to store a bit of data. These cells are logically addressable with a row and column address pair. Accesses within a device begin with first selecting a bank and then a row. This reads an entire row of bits (whose address is specified using the Row Access Strobe (RAS) command on the command bus) to per-bank sense amplifiers and latches that serve as the row-buffer. Then, a Column Access Strobe (CAS) command and the address selects a column from this buffer. The selected bits from each device are then aggregated at the bank level and sent to the memory controller over the data bus. A critical fact to note here is that once a RAS command is issued, the row-buffer holds a large number of bits that have been read from the DRAM cells. For the DDR2 memory

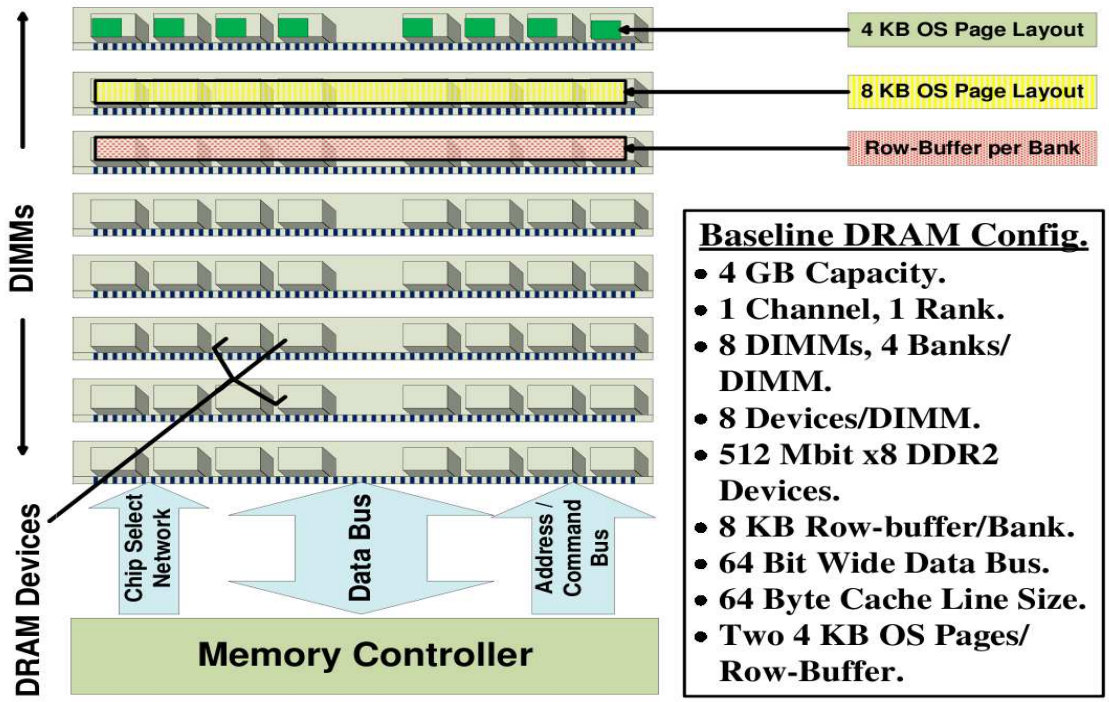


Figure 2.1: Typical DRAM Main Memory Organization and Data Layout.

system that we model for our experiments, the number of bits in a per-bank row-buffer is 64 K bits, which is typical of modern memory configurations. To access a 64 byte cache line, a row-buffer is activated and 64 K bits are read from the DRAM cells. Therefore, less than 1% of data in a bank's row-buffer is actually used to service one access request.

On every access, a RAS activates word-lines in the relevant mats and the contents of the cells in the selected row are driven on the bit-lines to the sense amplifiers where they are latched. The set of latches is the row-buffer. These actions are collectively referred to as the activation of the row-buffer. This read operation is a destructive process and data in the row-buffer must be written back after the access is complete. This write-back can be on the critical path of an access to a new row. The activation of bit-lines across several DRAM devices is the biggest contributor to DRAM power. To reduce the delays and energy involved in row-buffer activation, the memory controller adopts one of the following row-buffer management policies:

- *Open-page policy:* The bit-lines for the row are precharged only after a request to a different row is encountered.
- *Close-page policy:* The bit-lines for the row are precharged immediately after servicing

a cache line request.

The open-row policy is based on an optimistic assumption that some accesses in the near future will be to this open page - this amortizes the mat read energy and latency across multiple accesses. State-of-the-art DRAM address mapping policies [127, 80] try to map data such that there are as few row-buffer conflicts as possible. Page-interleaved schemes map contiguous physical addresses to the same row in the same bank. This allows the open-page policy to leverage spatial and temporal locality of accesses. Cache-line interleaved mapping is used with multichannel DRAM systems with consecutive cache lines mapped to different rows/banks/channels to allow maximum overlap in servicing requests.

On the other hand, the rationale behind close-page policy is driven by the assumption that no subsequent accesses will be to the same row. This is focused towards hiding the latency of bit-line precharge when subsequent requests are to different rows and is best suited for memory access streams that show little locality - like those in systems with high core counts.

2.2 DRAM Memory Channel Designs

A typical memory system is composed of multiple components. The Memory Controllers (MCs), which are implemented on the CPU die, control all memory accesses. MCs are connected via CPU socket pins to the memory bus, which on the other end is connected to the memory modules. The predominant choice for the bus connecting the CPU to the memory modules is a multidrop bus [29]. Although this design is simple and economical, it has many disadvantages. Issues like signal integrity, electrical loading, skew, jitter, etc. limit its usefulness for high-performance, high-capacity memory systems [93, 59].

An alternative to the multidrop bus is the Rambus memory design [31, 98]. The Rambus memory channel is a packet-based channel designed for a high bandwidth interface between the CPU and the DRAM devices. The Rambus designs not only target high bandwidth, they also ensure that high bandwidth is sustainable with a variable number of DRAM devices connected to the same bus. This allows building systems with upgradeable capacity, while achieving high data transfer rates. Rambus' designs, however, involve changes to the DRAM device architecture [59], and are thus limited to niche markets due to higher cost.

In the last decade, the Fully Buffered-Dual In-Line Memory Module (FB-DIMM) [46] architecture was invented to counter the shortcomings of JEDEC memory systems, while being minimally invasive on DRAM device design. The FB-DIMM design features a serial, point-to-point link-based channel. This allows adding FB-DIMM modules on the same

channel in a daisy-chain fashion to increase capacity without loss of signal-integrity. While it was a step in the right direction, several shortcomings of the design prevented it from replacing the JEDEC designs. Most notable were the power consumption and increased access latency for memory requests. Increased latency occurred due to two factors:*(1)* serial/de-serialization delay on the channel, and *(2)* when DIMMs were added to a channel, the memory controller scheduled all requests for worst-case channel latency. This model was adopted to keep the controller complexity low.

The use of a serial channel for the CPU-memory interface is again starting to gain traction, primarily due to its ability to support higher capacity and bandwidth within a limited CPU pin count. Intel's Scalable Memory Interface (SMI) design uses a buffer chip placed on the server motherboard that acts as an interface between the CPU's serial channel and JEDEC-based commodity DIMMs [58]. IBM's Power7 CPU also has a custom CPU-to-memory interface that uses a serial link from the CPU to the buffer chip on the motherboard [121, 104, 20]. These buffer chips provide features like high channel bandwidth, buffering, etc. while allowing the use of commodity DRAM parts [29]. These custom designs echo aspects of FB-DIMM memory, but alleviate some of the FB-DIMM problems by limiting the number of DIMMs connected to a single processor link.

Several systems use serial, point-to-point (P2P) high-speed links with differential signaling to achieve fast data rates [58, 104, 98]. Multiple loads on a bus increase the signal reflections, adding noise on the bus and thus reducing the signal-to-noise ratio. This noise is difficult to eliminate and it ultimately limits the memory capacity per channel because it restricts the number of DIMMs that can be loaded on to a channel. P2P links, however, do not suffer from these issues because there is only one reflection of the signal on the channel, which can be terminated without slowing down the bus relatively easily. The use of differential signaling for higher data rates and low power, however, incurs a pin count penalty. Differential signaling uses two pins per bit, compared to one pin per bit for buses. In spite of this pin count penalty, these P2P links provide a higher pin bandwidth than what DRAM chip pins designed following the Double Data Rate-3 (DDR3) standard are capable of. Each link is kept narrow – this enables more links and more DIMMs at a given processor pin budget. In spite of the narrow link, the transfer time for a cache line is not very high because of the higher signaling rate. The link typically has separate upstream and downstream sublinks, with possibly different widths. Serial links also add additional delay and energy due to the Serialization/De-Serialization required to format large data packets before/after link transmission. Despite these shortcomings, P2P serial links are becoming

the preferred memory channel technology, as evident from state-of-the-art memory system designs [29].

2.3 3D Stacked Memory Devices

A number of 3D stacked memory products have been announced in recent years [95, 112, 42]. Micron's Hybrid Memory Cube (HMC) has attracted much attention and multiple companies are collaborating to develop HMC-related solutions [8]. The first-generation HMC prototype has four DRAM chips stacked upon a logic layer. The DRAM chips are organized into 128 independent banks, offering very high parallelism [61]. The logic layer and its SerDes circuits account for a large fraction of HMC power [102]. These SerDes circuits drive 32-bit links (16 bits for transmit and 16 bits for receive) and each HMC offers four links [95]. A link can connect multiple HMCs to form a network of memories. Although the internal details of the HMC logic layer are not publicly available, it is safe to assume that the logic layer can accommodate the circuits required to implement typical routing functionality for this network. The HMC package integrates an entire DIMM and its buffer chip on to a single package. While each HMC is itself rated at a nontrivial 11 W [102], this approach is more efficient and scalable than the SMI approach because there is no protocol conversion and no DDR3 I/O circuitry within the HMC.

The HMC pipeline is assumed to be as follows. When a request arrives, it must first be De-Serialized. The incoming packet is examined and placed in a DRAM Access Queue if it is intended for this HMC. If the packet must be forwarded, it is buffered and made to go through a regular 3-stage router pipeline [96] that includes routing computation, switch allocation, and switch traversal. The packet must go through Serialization logic before being placed on the output link.

Requests in the DRAM Access Queue are issued as the banks become available, preserving a First-In-First-Out (FIFO) order as far as possible. In other words, we assume that the HMCs do not perform any intelligent scheduling and the processor retains control of mechanisms that enforce scheduling priority among threads. As for server systems with many threads, we assume a close-page policy for the memory banks. Once data are available (for a read), they are placed in a Completed Request Queue and injected into the router in FIFO order.

CHAPTER 3

MICRO-PAGES

This chapter describes data placement schemes that aim at improving both memory access delay and energy consumption. These schemes are targeted at traditional DDR3-based memory systems and leverage the data access pattern at a cache-line granularity to improve DRAM row-buffer utilization.

3.1 Introduction

Destructive interference among independent memory access streams from multicore CPUs, coupled with large over-fetch at the DRAM row-buffers, leads to low row-buffer hit rates and inefficient memory accesses. In this chapter, we describe schemes that operate within the parameters of existing DRAM device architectures and the JEDEC protocols while improving row-buffer hit rates and increasing memory access efficiency. Our approach stems from the observation that accesses to heavily referenced OS pages are clustered around a few cache blocks. This presents an opportunity to co-locate these clusters from different OS pages in a single row-buffer. This leads to a dense packing of heavily referenced blocks in a few DRAM row-buffers. The performance and power improvements due to our schemes come from improved row-buffer utilization that inevitably leads to reduced energy consumption and access latencies for DRAM memory systems.

This chapter describes a scheme to co-locate heavily accessed clusters of cache blocks by controlling the address mapping of OS pages to DRAM devices. Two implementations of this scheme are described here which modify the address mapping by employing software and hardware techniques. The software technique relies on reducing the OS page size, while the hardware method employs a new level of indirection for physical addresses. These schemes can easily be implemented in the OS or the memory controller. Thus, the relative inflexibility of device architectures and signaling standards does not preclude these innovations. Furthermore, the schemes also fit nicely with prior work on memory controller scheduling policies, thus allowing additive improvements.

3.2 Motivation

Traditional uniprocessor systems perform well with an open-page policy since the memory request stream being generated follows temporal and spatial locality. This locality makes subsequent requests more likely to be served by the open-page. However, with multicore systems, the randomness of requests has made open-page policy somewhat ineffective. This results in low row-buffer utilization. This drop in row-buffer utilization (hit-rate) is quantified in Figure 3.1 for a few applications. Results are present for 4 multithreaded applications from the PARSEC [22] suite and one multiprogrammed workload mix of two applications each from SPEC CPU2006 [51] and BioBench [12] suites (*lbm*, *mcf*, and *fasta-dna*, *mummer*, respectively). Multithreaded applications were first run with a single thread on a 1-core system and then with four threads on a 4-core CMP. Each application was run for 2 billion instructions, or the end of parallel-section, whichever occurred first. Other simulation parameters are described in detail in Section 3.4.

It was observed that the average utilization of row-buffers dropped in 4-core CMP systems when compared to a 1-core system. For multithreaded applications, the average row-buffer utilization dropped from nearly 30% in the 1-core setup to 18% for the 4-core setup. For the 4 single-threaded benchmarks in the experiment (*lbm*, *mcf*, *fasta-dna*, and *mummer*), the average utilization was 59% when each benchmark was run in isolation. It dropped to 7% for the multiprogrammed workload mix created from these same applications. This points towards the urgent need to address this problem since application performance is sensitive to DRAM access latency, and DRAM power in modern server systems can account for nearly 30% of total system power [17].

Before describing the data placement mechanisms, the baseline system configuration is described to facilitate description of the mechanisms.

3.2.1 Baseline Memory Organization

For experimentally evaluating the schemes, a 32-bit system with 4 GB of total main memory is modeled. The 4 GB DRAM memory capacity is spread across 8 non-ECC, unbufferred DIMMs, as depicted in Figure 2.1. Micron MT47H64M8 [84] parts are used as the DRAM devices to build this memory system - this is a 512 Mbit, x8 part. Further details about this DRAM device and system set-up are in Section 3.4. Each row-buffer for this setup is 8 KB in size, and since there are 4 banks/device, there are 4 row-buffers per DIMM. The memory system is modeled following the DDR2-800 memory system specifications, with DRAM devices operating at 200 MHz, and a 64-bit wide data bus operating at 400 MHz.

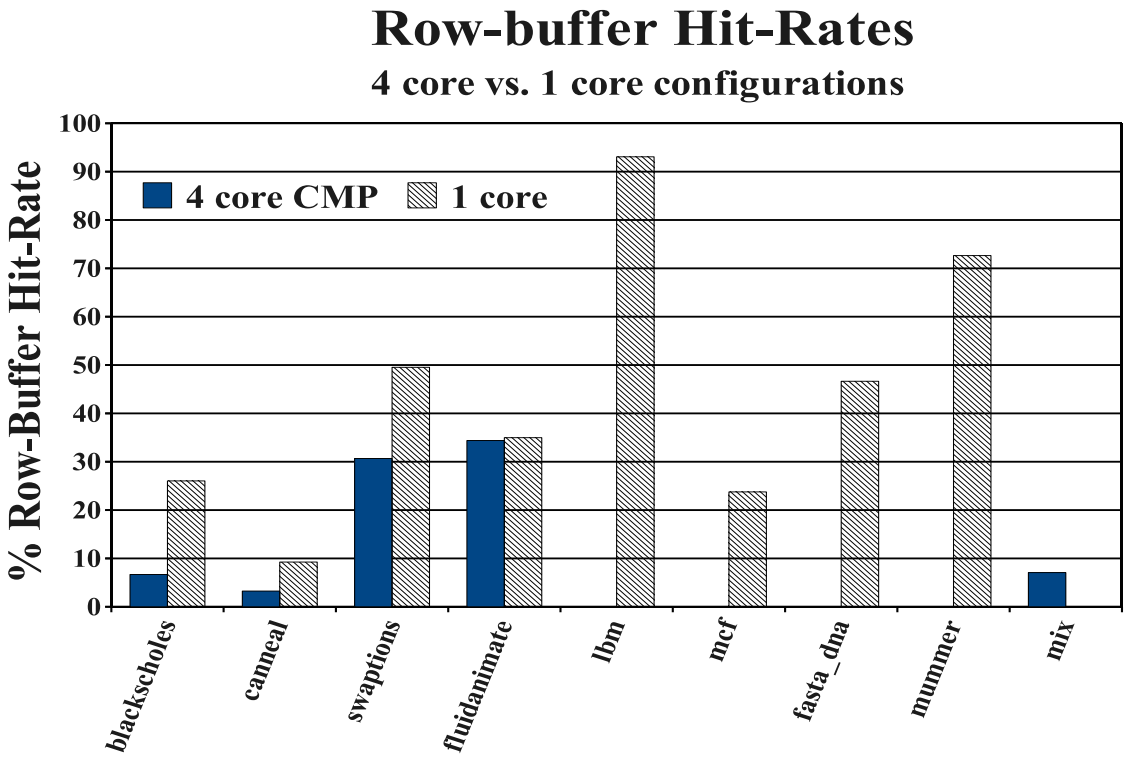


Figure 3.1: Row-buffer Hit-Rates for 1- and 4-Core Configurations

For a 64 byte sized cache line, it takes 8 clock edges to transfer a cache line from the DIMMs to the memory controller.

3.2.2 Baseline DRAM Addressing

Using the typical OS page size of 4 KB, the baseline page-interleaved data layout is as shown in Figure 3.2. Data from a page is spread across memory such that they reside in the same DIMM, same bank, and the same row. This mapping is similar to that adopted by Intel 845G Memory Controller Hub for this configuration [59, 118, 5], and affords simplicity in explaining our proposed schemes. We now describe how the bits of a physical address are interpreted by the DRAM system to map data across the storage cells. For a 32-bit physical address (Figure 3.2), the low order 3 bits are used as the byte address, bits 3 through 12 provide the column address, bits 13 and 14 denote the bank, bits 15 through 28 provide the row I.D, and the most significant 3 bits indicate the DIMM I.D. An OS page therefore spans across all the devices on a DIMM, and along the same row and bank in all the devices. For a 4 KB page size, a row-buffer in a DIMM holds two entire OS pages, and each device on the DIMM holds 512 bytes of that page's data.

3.2.3 Motivational Results

An interesting observation for DRAM memory accesses at OS page granularity is that for most applications, accesses in a given time interval are clustered around few contiguous cache-line sized blocks in the most referenced OS pages. Figures 3.3 and 3.4 show this pattern for two SPEC CPU2006 applications. The X-axis shows the sixty-four 64 byte cache blocks in a 4 KB OS page, and the Z-axis shows the percent of total accesses to each block within that page. The Y-axis plots the most frequently accessed pages in sorted order.

The data presented here include pages that account for approximately 25% of total DRAM requests served for a 2 billion instruction period. These experiments were run with 32 KB, 2-way split L1 I and D-cache, and 128 KB, 8-way L2 cache for a single-core setup.

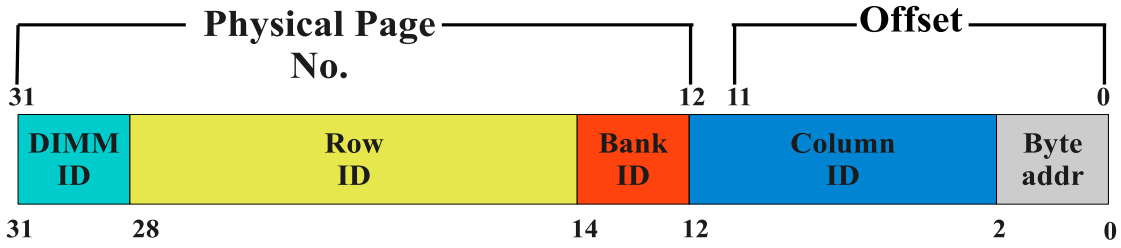


Figure 3.2: Baseline DRAM Address Mapping

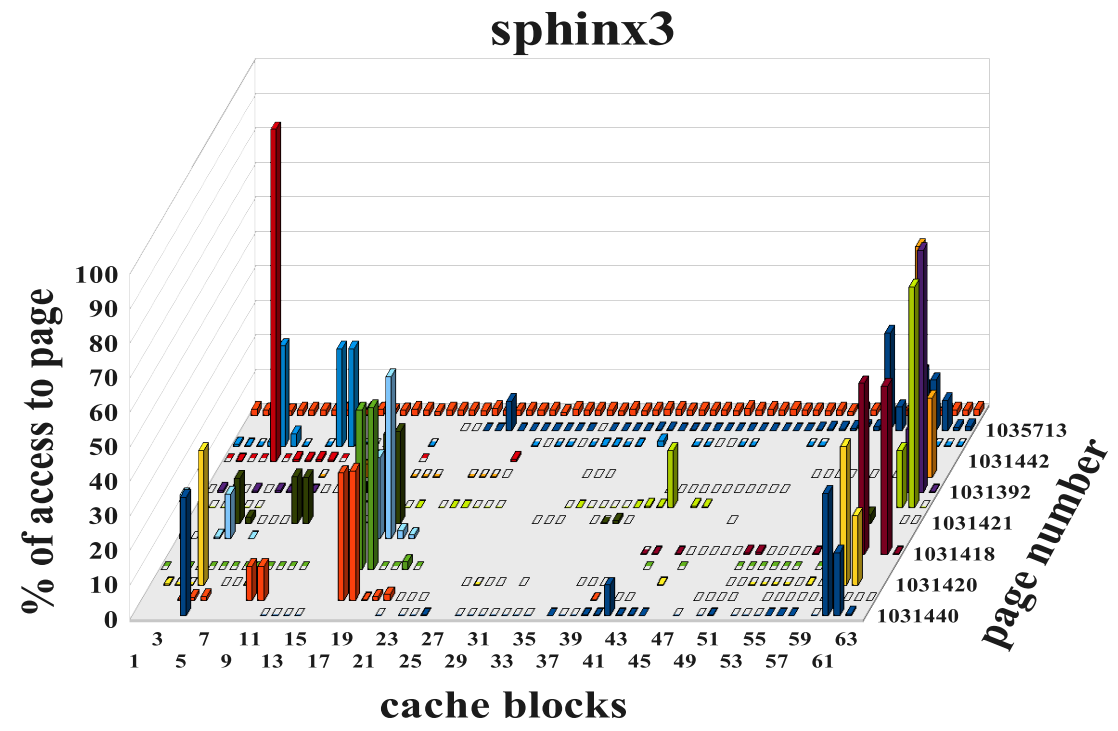


Figure 3.3: sphinx3

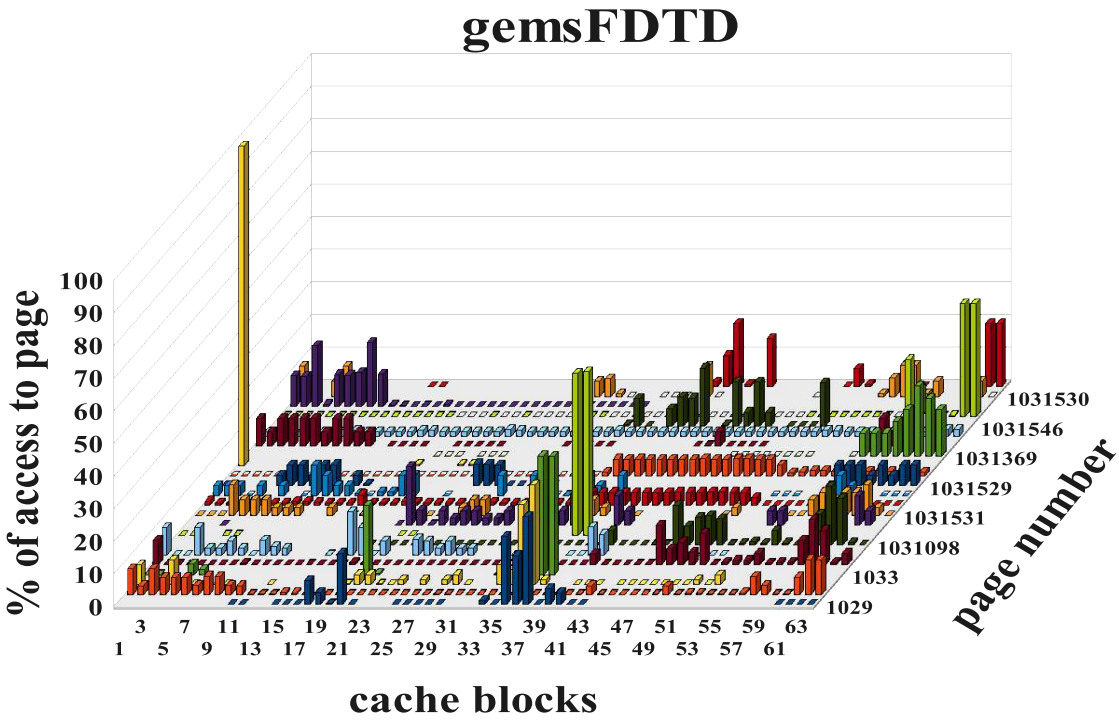


Figure 3.4: gemsFDTD

To make sure that results were not unduly influenced by configuration parameters and not limited to certain applications, we varied all the parameters and simulated different benchmark applications from PARSEC, SPEC, NPB [16], and BioBench benchmark suites. The execution interval was also varied over various phases of the application execution, and increased cache sizes and associativity to make sure that the conflict misses were reduced. Figures 3.3 and 3.4 show 3D graphs for only two representative experiments because the rest of the workloads exhibited similar patterns. The accesses were always clustered around a few blocks in a page, and very few pages accounted for most accesses in a given interval (less than 1% of OS pages account for almost $1/4^{th}$ of the total accesses in the 2 billion instruction interval; the exact figures for the shown benchmarks are: *sphinx3* - 0.1%, *gemsFDTD* - 0.2%).

These observations lead to the central theme of this work - *co-locating clusters of contiguous blocks with similar access counts, from different OS pages, in a row-buffer to improve its utilization.*

In principle, one can imagine taking individual cache blocks and co-locating them in a row-buffer. However, such granularity would be too fine to manage a DRAM memory system and the overheads would be huge. For example, in a system with 64 B wide cache lines and 4 GB of DRAM memory, the scheme would need to track nearly 67 million blocks! To overcome these overheads, we instead focus on dealing with clusters of contiguous cache blocks. In this chapter, these clusters are referred to as *"micro-pages"*.

3.3 Proposed Mechanisms

A common aspect of all the schemes described in this chapter is the identification and subsequent co-location of frequently accessed micro-pages in row-buffers to increase row-buffer hit rates. This chapter describes two mechanisms to this effect - decreasing OS page size and performing page migration for co-location, and hardware assisted migration of segments of a conventional OS page (hereafter referred to as a micro-page). As explained in Section 3.1, row-buffer hit rates can be increased by populating a row-buffer with those chunks of a page which are frequently accessed in the same window of execution. These chunks from different OS pages are referred to as "hot" micro-pages if they are frequently accessed during the *same* execution epoch.

For all the schemes, a common mechanism to identify hot micro-pages at run-time is proposed. This mechanism relies on new counters implemented at the memory controller that keep track of accesses to micro-pages within different OS pages. For a 4 KB OS page size, 1 KB micro-page size, and assuming application memory footprint in a 50 million cycle

epoch to be 512 KB (in Section 3.4 the results show that the footprint of hot micro-pages is actually lower than 512 KB for most applications), the total number of such counters required is 512. This is a small overhead in hardware and since the update of these counters is not on the critical path while scheduling requests at the memory controller, it does not introduce any latency overhead. However, an associative look-up of these counters (for matching micro-page number) can be energy inefficient. To mitigate this, techniques like hash-based updating of counters can be adopted. Since very few micro-pages are touched in an epoch, the probability of hash-collision is small. If the memory footprint of the application exceeds 512 KB, then some errors in the estimation of hot micro-pages can be expected.

At the end of an epoch, an OS daemon inspects the counters and the history from the preceding epoch to rank micro-pages as "hot" based on their access counts. The daemon then selects the micro-pages that are suitable candidates for migration. Subsequently, the specifics of the proposed schemes take over the migrations and associated actions required to co-locate the hot micro-pages. By using the epoch-based statistics collection, and evaluating hotness of a page based on the preceding epoch's history, this scheme implicitly uses both temporal and spatial locality of requests to identify hot micro-pages. The counter values and the preceding epoch's history are preserved for each process across a context switch. This is a trivial modification to the OS' context switching module that saves and restores application context. It is critical to have such a dynamic scheme so it can easily adapt to varying memory access patterns. Huang et al. [54] describe how memory access patterns are not only continuously changing within an application, but also across context switches, thus necessitating a dynamically adaptive scheme. By implementing this in software, we gain flexibility not afforded by hardware implementations.

3.3.1 Reducing OS Page Size (ROPS)

This section first describes the basic idea of co-locating hot micro-pages by reducing the OS page size. It then discusses the need to create superpages from micro-pages to reduce bookkeeping overhead and mitigate the reduced Translation Lookaside Buffer (TLB) reach due to smaller page size.

Basic Idea: In this scheme, the objective is to reduce the OS page size such that frequently accessed contiguous blocks are clustered together in the new reduced size page (a micro-page). The hot micro-pages are then migrated using DRAM copy to co-locate frequently accessed micro-pages in the same row-buffer. The innovation here is to make the OS' Virtual Address (VA) to Physical Address (PA) mapping cognizant of row-buffer uti-

lization. This is achieved by modifying the original mapping assigned by the OS' underlying physical memory allocation algorithm such that hot pages are mapped to physical addresses that are co-located in the same row-buffer. Every micro-page migration is accompanied by an associated TLB shoot-down and change in its page table entry.

Baseline Operation: To understand the scheme in detail, it is useful to first describe the sequence of events from the first time some data are accessed by an application. The application initially makes a request to the OS to allocate some amount of space in the physical memory. This request can be explicit via calls to *malloc( )*, or implicit via compiler indicated reservation in the stack. The OS in turn assigns a virtual to physical page mapping for this request by creating a new page table entry. Subsequently, when the data are first accessed by the application, a TLB miss fetches the appropriate entry from the page table. The data are then either fetched from the disk into the appropriate physical memory location via a Direct Memory Access (DMA) copy, or the OS could copy another page (copy-on-write), or allocate a new empty page. An important point to note from the perspective of our schemes is that in either of these three cases: 1) data fetch from the disk, 2) copy-on-write, and 3) allocation of an empty frame, the page table is accessed. The impact of this is discussed shortly.

Overheads: For the schemes described in this chapter, a minor change to the above sequence of events is suggested. The idea is to reduce the page size across the entire system to 1 KB, and instead of allocating one page table entry the first time the request is made to allocate physical memory, the OS creates four page table entries (each equal to page size of 1 KB). The scheme therefore leverages a "reservation-based" [111, 91] allocation approach for 1 KB base-pages, i.e., on first-touch, contiguous 1 KB virtual pages are allocated to contiguous physical pages. This ensures that the overhead required to move from a 1 KB base-page to a 4 KB superpage is only within the OS and does not require DRAM page copy. These page table entries will therefore have contiguous virtual and physical addresses and will ensure easy creation of superpages later on. The negative effect of this change is that the page table size will increase substantially, at most by $4X$ compared to a page table for 4 KB page size. However, since we will be creating superpages for most of the 1 KB micro-pages later (this will be discussed in more detail shortly), the page table size will shrink back to a size similar to that for a baseline 4 KB OS page size. Note that in this scheme, 4 KB pages would only result from superpage creation and the modifications required to the page table have been explored in the literature for superpage creation. The smaller page size also impacts TLB coverage and TLB miss rate. The impact of this change,

however, is minimal if most 1 KB pages are eventually coalesced into 4 KB pages.

The reason to choose 1 KB micro-page size was dictated by a trade-off analysis between TLB coverage and a manageable micro-page granularity. This resulted from an experiment performed to determine the benefit obtained by having a smaller page size and performance degradation due to drop in TLB coverage. For almost all applications, 1 KB micro-page size offered a good trade-off.

Reserved Space: Besides the above change to the OS' memory allocator, the technique also reserves frames in the main memory that are never assigned to any application. These reserved frames belong to the first 16 rows in each bank of each DIMM and are used to co-locate hot micro-pages. The total capacity reserved for our setup is 4 MB and is less than 0.5% of the total DRAM capacity. Results are presented later that show most of the hot micro-pages can be accommodated in these reserved frames.

Actions per Epoch: After the above-described one-time actions of this scheme, the scheme identifies "hot" micro-pages every epoch using the OS daemon as described earlier. This daemon examines counters to determine the hot micro-pages that must be migrated. If deemed necessary, hot micro-pages are subsequently migrated by forcing a DRAM copy for each migrated micro-page. The OS' page table entries are also changed to reflect this. To mitigate the impact of reduced TLB reach due to smaller page size, superpages are created. Every contiguous group of four 1 KB pages that do not contain a migrated micro-page are promoted to a 4 KB superpage.

Thus, the sequence of steps taken for co-locating the micro-pages for this scheme are as follows:

- Look up the hardware counters in the memory controller and designate micro-pages as "hot" by combining this information with statistics for the previous epoch - performed by an OS daemon described earlier.
- To co-locate hot micro-pages, force DRAM copies causing migration of micro-pages. Then, update the page table entries to appropriate physical addresses for the migrated micro-pages.
- Finally, create as many superpages as possible.

Superpage Creation: The advantages that might be gained from enabling the OS to allocate physical addresses at a finer granularity may be offset by the penalty incurred due to reduced TLB reach. To mitigate these effects of higher TLB misses, we incorporate the

creation of superpages [91, 101]. Superpage creation [91, 101, 109, 45] has been extensively studied in the past and we omit the details of those mechanisms here. Note that in our scheme, superpages can be created only with "cold" micro-pages since migrated hot micro-pages leave a "hole" in the 4 KB contiguous virtual and physical address spaces. Other restrictions for superpage creation are easily dealt with as discussed next.

TLB Status Bits: We allocate contiguous physical addresses for four 1 KB micro-pages that were contiguous in virtual address space. This allows us to create superpages from a set of four micro-pages which do not contain a hot micro-page that has been migrated. When a superpage is created, a few factors need to be taken into account, specifically the various status bits associated with each entry in the TLB.

- The included base-pages must have identical protection and access bits since the superpage entry in the TLB has only one field for these bits. This is not a problem because large regions in the virtual memory space typically have the same state of protection and access bits. This happens because programs usually have large segments of densely populated virtual address spaces with similar access and protection bits.

- Base-pages also share the dirty-bit when mapped to a superpage. The dirty-bit for a superpage can be the logical OR of the dirty-bits of the individual base-pages. A minor inefficiency is introduced because an entire 4 KB page must be written back even when only one micro-page is dirty.

- The processor must support a wide range of superpage sizes (already common in modern processors), including having a larger field for the physical page number.

Schemes proposed by Swanson et al. [109] and Fang et al. [45] have also shown that it is possible to create superpages that are noncontiguous in physical address space and unaligned. We mention these schemes here but leave their incorporation as future work.

The major overheads experienced by the ROPS scheme arise from two sources:

- DRAM copy of migrated pages and associated TLB shoot-down, and page table modifications.

- Reduction in TLB coverage.

In Section 3.4, we show that DRAM migration overhead is not large because on an average, only a few new micro-pages are identified as hot every epoch and moved. As a

result, the major overhead of DRAM copy is relatively small. However, the next scheme proposed below eliminates the above-mentioned overheads and is shown to perform better than ROPS. The performance difference is not large, however.

3.3.2 Hardware Assisted Migration (HAM)

This scheme introduces a new layer of translation between physical addresses assigned by the OS (and stored in the page table while allocating a new frame in main memory) and those used by the memory controller to access the DRAM devices. This translation keeps track of the new physical addresses of the hot micro-pages that are being migrated and allows migration without changing the OS' page size, or any page table entries. Since the OS page size is not changed, there is no drop in TLB coverage.

Indirection: In this scheme, we propose look-up of a table at the memory controller to determine the new address if the data have been migrated. We organize the table so it consumes acceptable energy and area and these design choices are described subsequently. The address translation required by this scheme is usually not on the critical path of accesses and the translated addresses are generated as shown in Figure 3.5. Memory requests usually wait in the memory controller queues for a long time before being serviced. The above translation can begin when the request is queued and the delay for translation can be easily hidden behind the long wait time. The notion of introducing a new level of indirection has been widely used in the past, for example, within memory controllers to aggregate distributed locations in the memory [24]. More recently, it has been used to control data placement in large last-level caches [14, 27, 50].

Similar to the ROPS schemes, the OS daemon is responsible for selecting hot micro-pages fit for co-location. Thereafter, this scheme performs a DRAM copy of "hot" micro-pages. However, instead of modifying the page table entries as in ROPS, this scheme involves populating a "Mapping Table" (MT) in the memory controller with mappings from the old to the new physical addresses for each migrated page.

Request Handling: When a request (with physical address assigned by the OS) arrives at the memory controller, it searches for the address in the MT (using certain bits of the address as described later). On a hit, the new address of the micro-page is used by the memory controller to issue the appropriate commands to retrieve the data from their new location - otherwise, the original address is used. The MT look-up happens the moment a request is added to the memory controller queue and does not extend the critical path in the common case because queuing delays at the memory controller are substantial.

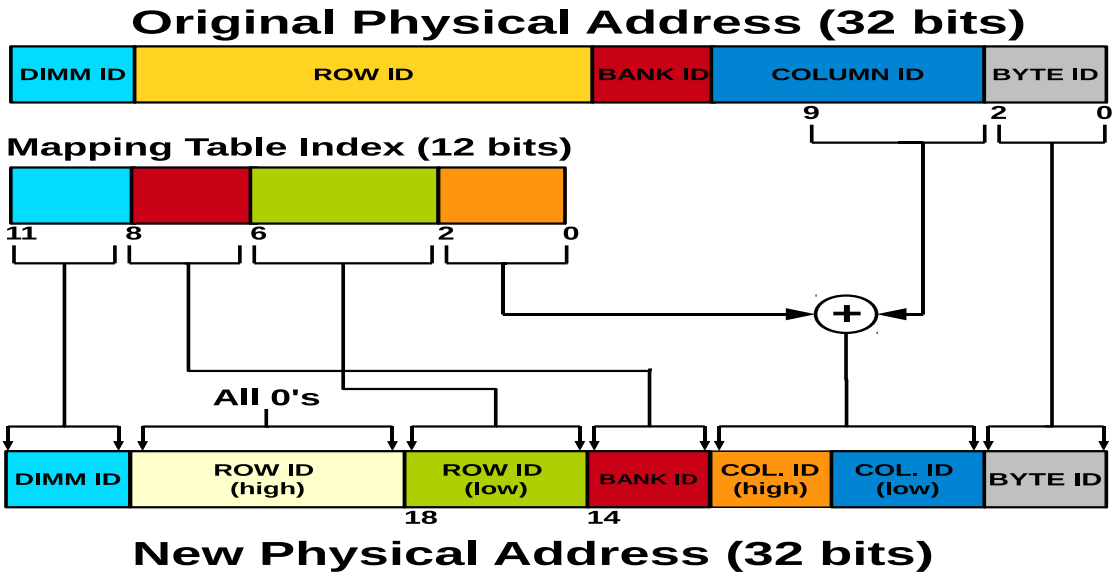


Figure 3.5: Re-Mapped DRAM Addressing

Micro-page Migration: Every epoch, micro-pages are rated and selected for migration. The first 16 rows in each bank are reserved to hold the hot micro-pages. The OS' VA to PA mapping scheme is modified to make sure that no page gets mapped to these reserved rows. The capacity lost due to this reservation (4 MB) is less than 0.5% of the total DRAM capacity and in Section 3.4, we show that this capacity is sufficient to hold almost all hot micro-pages in a given epoch.

At the end of each epoch, hot micro-pages are moved to one of the slots in the reserved rows. If a given micro-page is deemed hot, and is already placed in the reserved row (from the previous epoch), we do not move it. Otherwise, a slot in the reserved row is made empty by first moving the now "cold" micro-page to its original address. The new hot micro-page is then brought to this empty slot. The original address of the cold page being replaced is derived from the contents of the corresponding entry of the MT. After the migration is complete, the MT is updated accordingly.

This migration also implies that the portion of the original row from which a hot micro-page is migrated now has a "hole" in it. This does not pose any correctness issue in terms of data look-up for a migrated micro-page at its former location since an access to a migrated address will be redirected to its new location by the memory controller.

Mapping Table (MT): The mapping table contains the mapping from the original OS-assigned physical address to the new address for each migrated micro-page. The size and organization of the mapping table can significantly affect the energy spent in the address translation process. We discuss the possible organizations for the table in this subsection.

The scheme reserves 16 rows of each bank in a DIMM to hold the hot micro-pages. The reservation of these rows implies that the total number of slots where a hot micro-page might be relocated is 4096 (8 DIMMS $*$ 4 banks/DIMM $*$ 16 rows/bank $*$ 8 micro-page per row).

The MT is designed as a banked fully-associative structure. It has 4096 entries, one for each slot reserved for a hot micro-page. Each entry stores the original physical address for the hot micro-page resident in that slot. The entry is populated when the micro-page is migrated. Since it takes 22 bits to identify each 1 KB micro-page in the 32-bit architecture, the MT requires a total storage capacity of 11 KB.

On every memory request, the MT must be looked up to determine if the request must be redirected to a reserved slot. The original physical address must be compared against the addresses stored in all 4096 entries. The entry number that flags a hit is then used to construct the new address for the migrated micro-page. A couple of optimizations can be attempted to reduce the energy overhead of the associative search. First, one can restrict a hot micro-page to only be migrated to a reserved row in the same DIMM. Thus, only $1/8^{th}$ of the MT must be searched for each look-up. Such a banked organization is assumed in our quantitative results. A second optimization can set a bit in the migrated page's TLB entry. The MT is looked up only if this bit is set in the TLB entry corresponding to that request. This optimization is left as future work. In terms of access latency of the associative search, note that the look-up occurs when a memory request is first added to the memory controllers request queue. Since memory requests typically spend a considerable fraction of their access time in the queue, the associative lookup of the MT is not on the critical path.

When a micro-page must be copied back from a reserved row to its original address, the MT is looked up with the ID (0 to 4095) of the reserved location, and the micro-page is copied into the original location saved in the corresponding MT entry.

3.4 Results

3.4.1 Methodology

Our detailed memory system simulator is built upon the Virtutech Simics [83, 4] platform and important parameters of the simulated system are shown in Table 3.1. Out-of-order timing is simulated using Simics' *sample-micro-arch* module and the DRAM memory subsystem is modeled in detail using a modified version of Simics' *trans-staller* module. It closely follows the model described by Gries in [84]. The memory controller keeps track of each DIMM and open rows in each bank. It schedules the requests based on open-

Table 3.1: Simulator Parameters.

CMP Parameters

ISA	UltraSPARC III ISA
CMP Size and Core Frequency	4-core, 2 GHz
Re-Order-Buffer	64 entry
Fetch, Dispatch, Execute, and Retire	Maximum 4 per cycle
L1 I-cache	32 KB/2-way, private, 1-cycle
L1 D-cache	32KB/2-way, private, 1-cycle
L2 Cache	128 KB/8-way, shared, 10-cycle
L1 and L2 Cache Line Size	64 Bytes
Coherence Protocol	Snooping MESI

DRAM Parameters

DRAM Device Parameters	Micron MT47H64M8 DDR2-800 Timing parameters [84], t_{CL}=t_{RCD}=t_{RP}=20ns(4-4-4 @ 200 MHz) 4 banks/device, 16384 rows/bank, 512 columns/row,8-bit output/device
DIMM Configuration	8 Non-ECC un-buffered DIMMs, 1 rank/DIMM, 64 bit channel, 8 devices/DIMM
DIMM-Level Row-Buffer Size	8 KB
Active Row-Buffers per DIMM	4
Total DRAM Capacity	512 MBit/device $\times$ 8 devices/DIMM $\times$ 8 DIMMs = 4 GB

page and close-page policies. To keep the memory controller model simple, optimizations that do not directly affect the schemes described in this chapter, like finite queue length, critical-word-first optimization, and support for prioritizing reads, are not modeled. Other major components of Gries' model adopted are: the bus model, DIMM and device models, and most importantly, simultaneous pipelined processing of multiple requests. The last component allows hiding activation and precharge latency using pipelined interface of DRAM devices. The CPU is modeled to allow nonblocking load/store execution to support overlapped processing.

DRAM address mapping parameters for the simulator were adopted from the DRAMSim framework [118]. A single-channel basic SDRAM mapping, as found in memory systems similar to Intel 845G chipsets' DDR Synchronous DRAM (SDRAM) mapping [5], was implemented. Some platform-specific implementation suggestions were taken from the VASA framework [117]. The DRAM energy consumption model is built as a set of counters that keep track of each of the commands issued to the DRAM. Each precharge, activation, CAS, write-back to DRAM cells, etc. is recorded and total energy consumed reported

using energy parameters derived from Micron MT47H64M8 DDR2-800 datasheet [84]. This simulator does not model the energy consumption of the data and command buses as the schemes described here do not affect them. The energy consumption of the mapping table (MT) is derived from CACTI [87, 114] and is accounted for in the simulations.

These schemes are evaluated with full system simulation of a wide array of benchmarks. Multithreaded workloads from the PARSEC [22], OpenMP version of NAS Parallel Benchmark (NPB) [16], and SPECjbb [6] suites were used. The STREAM [3] benchmark was also used as one of the multithreaded workloads. Single threaded applications from SPEC CPU 2006 [51] and BioBench [12] suites were used for the multiprogrammed workload mixes. While selecting individual applications from these suites, the applications were first characterized for their total DRAM accesses, and then two applications were selected from each suite that had the highest DRAM accesses. For both multithreaded and single-threaded benchmarks, the application was simulated for 250 million cycles of execution. For multithreaded applications, the simulations start at the beginning of the parallel-region/region-of-interest of the application, and for single-threaded applications, the simulations start from 2 billion instructions after the start of the application. Total system throughput for single-threaded benchmarks is reported as weighted speedup [105], calculated as $\sum_{i=1}^{n}(IPC^{i}_{shared}/IPC^{i}_{alone})$, where IPC^{i}_{shared} is the IPC of program i in an "n" core CMP.

The applications from PARSEC suite are configured to run with *simlarge* input set, applications from NPB suite are configured with *Class A* input set and STREAM is configured with an array size of 120 million entries. Applications from SPEC CPU2006 suite were executed with the *ref* inputs and BioBench applications with the default input set. Due to simulation speed constraints, we only simulated each application for 250 million cycles. This resulted in extremely small working set sizes for all these applications. With 128 KB L1 size and 2 MB L2 size, we observed very high cache hit-rates for all these applications. We therefore had to scale down the L1 and L2 cache sizes to see any significant number of DRAM accesses. While deciding upon the scaled down cache sizes, we chose 32 KB L1 size and 128 KB L2 size since these sizes gave approximately the same L1 and L2 hit-rate as when the applications were run to completion with larger cache sizes. With the scaled down cache sizes, the observed cache hit-rates for all the applications are show in Table 3.2.

All experiments were performed with both First-Come First-Serve (FCFS) and First-Ready First-Come First-Serve (FR-FCFS) memory controller scheduling policies. Only results for the FR-FCFS policy are shown since it is the most commonly adopted scheduling

Table 3.2: L2 Cache Hit-Rates and Benchmark Input Sets.

Benchmark	L2 Hit Rate	Input Set
blackscholes	89.3%	simlarge
bodytrack	59.0%	simlarge
canneal	23.8%	simlarge
facesim	73.9%	simlarge
ferret	79.1%	simlarge
freqmine	83.6%	simlarge
streamcluster	65.4%	simlarge
swaptions	93.9%	simlarge
vips	58.45%	simlarge
IS	85.84%	class A
MG	52.4%	class A
mix	36.3%	ref (SPEC), default (BioBench)
STREAM	48.6%	120 million entry array
SPECjbb	69.9%	default

policy. The results for three types of platforms are shown here:

- **Baseline** - This is the case where OS pages are laid out as shown in Figure 2.1. The OS is responsible for mapping pages to memory frames and since we simulate Linux OS, it relies on the buddy system [69] to handle physical page allocations. (For some applications, the experiments use the Solaris OS.)

- **Epoch-Based Schemes** - These experiments are designed to model the proposed schemes. Every epoch an OS subroutine is triggered that reads the access counters at the memory controller and decides if migrating a micro-page is necessary.

- **Profiled Placement** - This is a two-pass experiment designed to quantify an approximate upper bound on the performance of our proposals. It does so by "looking into the future" to determine the best layout. During the first pass, it creates a trace of OS page accesses. For the second pass of the simulation, these traces are preprocessed and the best possible placement of micro-pages for every epoch is determined. This placement is decided by looking at the associated cost of migrating a page for the respective scheme, and the benefit it would entail. An important fact to note here is that since multithreaded simulations are nondeterministic, these experiments can be slightly unreliable indicators of best performance. In fact for some of the experiments, this scheme shows performance degradation. However, it is still an important metric to determine the approximate upper bound on performance.

For all the experimental evaluations, we use a constant overhead of 70,000 cycles per epoch to execute the OS daemon and for DRAM data migrations. For a 5 million cycle epoch, this amounts to 1.4% overhead per epoch in terms of cycles. From initial simulations, it was observed that approximately 900 micro-pages were being moved every epoch for all the simulated applications. From the parameters of the DRAM configuration, this evaluates to nearly 60,000 DRAM cycles for migrations. For all the above-mentioned schemes, the results are presented for different epoch lengths to show the sensitivity of the schemes to this parameter. The simulations use a 4 KB OS pages size with 1 KB micro-page size for all the experiments.

For the ROPS scheme, an additional 10,000 cycle penalty for all 1 KB $\rightarrow$ 4 KB superpage creations in an epoch is used. This overhead models the identification of candidate pages and the update of OS book-keeping structures. TLB shoot-down and the ensuing page walk overheads are added on top of this overhead. Superpage creation beyond 4 KB is not modeled as this behavior is expected to be the same for the baseline and proposed models.

The expected increase in page table size because of the use of smaller 1 KB pages is not modeled in detail in our simulator. After application warm-up and superpage promotion, the maximum number of additional page table entries required is 12,288 on a 4 GB main memory system, a 1.1% overhead in page table size. This small increase in page table size is only expected to impact cache behavior when TLB misses are unusually frequent – which is not the case, thus it does not affect our results unduly.

The decision to model 4 KB page sizes for this study was based on the fact that 4 KB pages are most common in hardware platforms like x86 and x86_64. An increase in baseline page size (to 8 KB or larger) would increase the improvement seen by our proposals, as the variation in subpage block accesses increase. UltraSPARC and other enterprise hardware often employ a minimum page size of 8 KB to reduce the page table size when addressing large amounts of memory. Thus, the performance improvements reported in this study are likely to be a conservative estimate for some architectures.

3.4.2 Evaluation

In this section, it is first shown that reserving 0.5% of our DRAM capacity (4 MB, or 4096 1 KB slots) for co-locating hot micro-pages is sufficient in Figure 3.6. The applications were simulated with a 5 million cycle epoch length. For each application, an epoch that touched the highest number of 4 KB pages was selected and then the percent of total accesses to micro-pages in the reserved DRAM capacity are plotted. The total number of 4 KB pages touched is also plotted in the same figure (right hand Y-axis). For all but

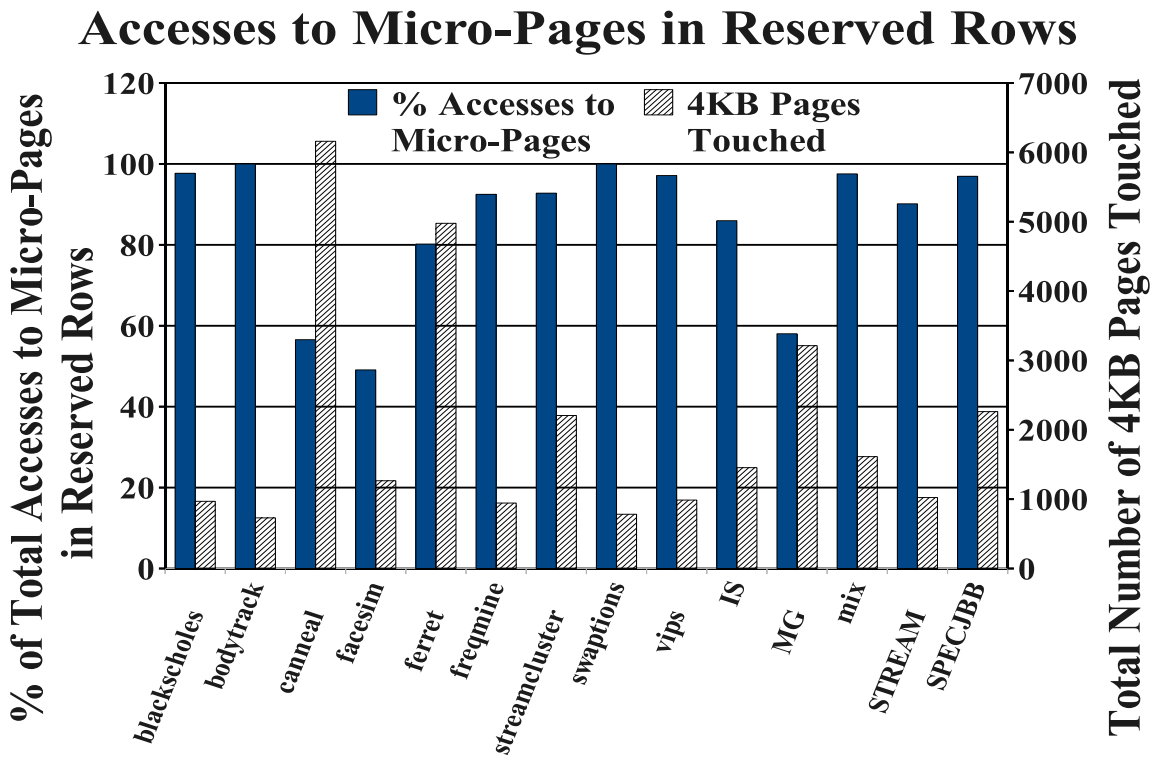


Figure 3.6: Accesses to Micro-Pages in Reserved DRAM Capacity.

3 applications (*canneal*, *facesim*, and *MG*), on an average, 94% of total access to DRAM in that epoch are to micro-pages in the reserved 4 MB capacity. This figure also shows that application footprints per epoch are relatively small and our decision to use only 512 counters at the DRAM is also valid. We obtained similar results with simulation intervals several billions of cycles long.

The next set of results present the impact of the reduced OS page size (ROPS) scheme, described in Section 3.3.1, in Figure 3.7. The results are shown for 14 applications - eight from PARSEC suite, two from NPB suite, one multiprogrammed mix from SPEC CPU2006 and BioBench suites, STREAM, and SPECjbb2005.

Figure 3.7 shows the performance of the proposed scheme compared to baseline. The graph also shows the change in TLB hit-rates for 1 KB page size compared to 4 KB page size, for a 128-entry TLB (secondary Y-axis, right-hand side). Note that for most of the applications, the change in TLB hit-rates is very small compared to baseline 4 KB pages. This demonstrates the efficacy of superpage creation in keeping TLB misses low. Only for three applications (*canneal*, multiprogrammed workload mix, and *SPECjbb*) does the change in TLB hit-rates go over 0.01% compared to baseline TLB hit-rate. Despite sensitivity of application performance to TLB hit-rate, with our proposed scheme, both

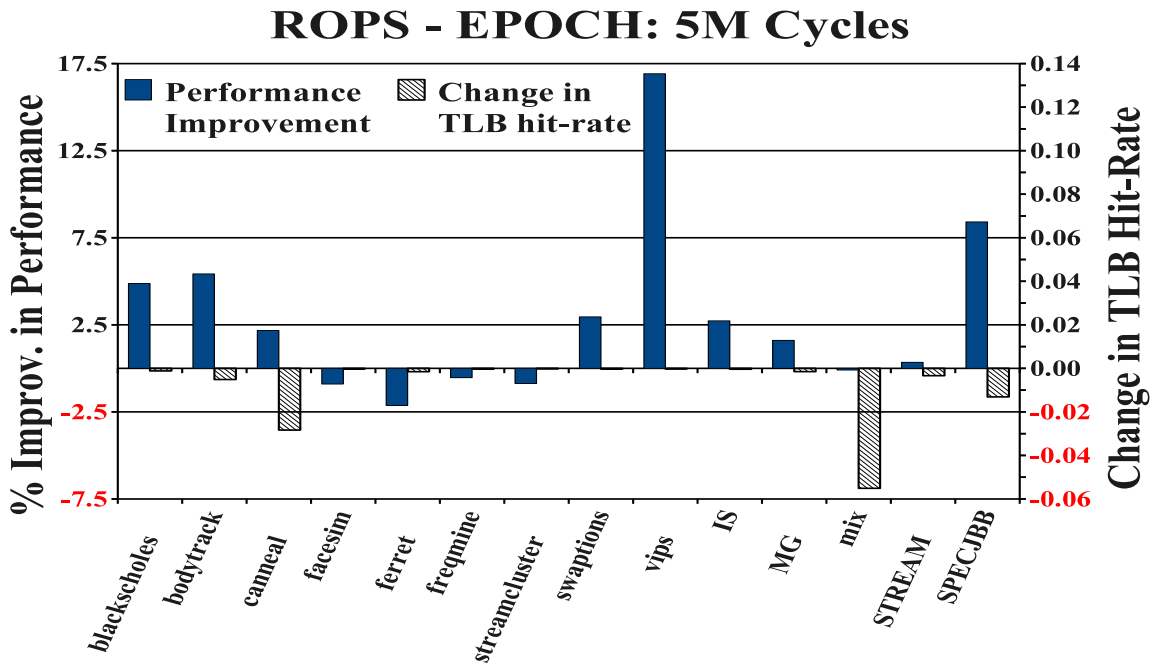


Figure 3.7: Performance Improvement and Change in TLB Hit-Rates (w.r.t. Baseline) for ROPS with 5 Million Cycle Epoch Length

canneal and *SPECjbb* show notable improvement. This is because application performance tends to be even more sensitive to DRAM latency. Therefore, for the Reduced OS Page Size (ROPS) proposal, a balance must be struck between reduced DRAM latency and reduced TLB hit-rate. Out of these 14 applications, 5 show performance degradation. This is attributed to the overheads involved in DRAM copies, increased TLB miss-rate, and the daemon overhead, while little performance improvement is gained from co-locating micro-pages. The reason for low performance improvements from co-location is the nature of these applications. If applications execute tight loops with regular access pattern within OS pages, then it is hard to improve row-buffer hit-rates due to fewer, if any, hot micro-pages. We talk more about this issue when we discuss results for the HAM scheme below.

Figure 3.8 presents energy-delay-squared ($E * D^2$) for the ROPS proposal with epoch length of 5 million cycles, normalized to the baseline. E refers to the DRAM energy (excluding memory channel energy) per CPU load/store. D refers to the inverse of throughput. The second bar (secondary Y-axis) plots the performance for convenient comparison. All applications, except *ferret*, have lower (or as much) $E * D^2$ compared to baseline with the proposed scheme. A higher number of row-buffer hits leads to higher overall efficiency: lower access time *and* lower energy. The average savings in energy for all the applications is nearly 12% for ROPS.

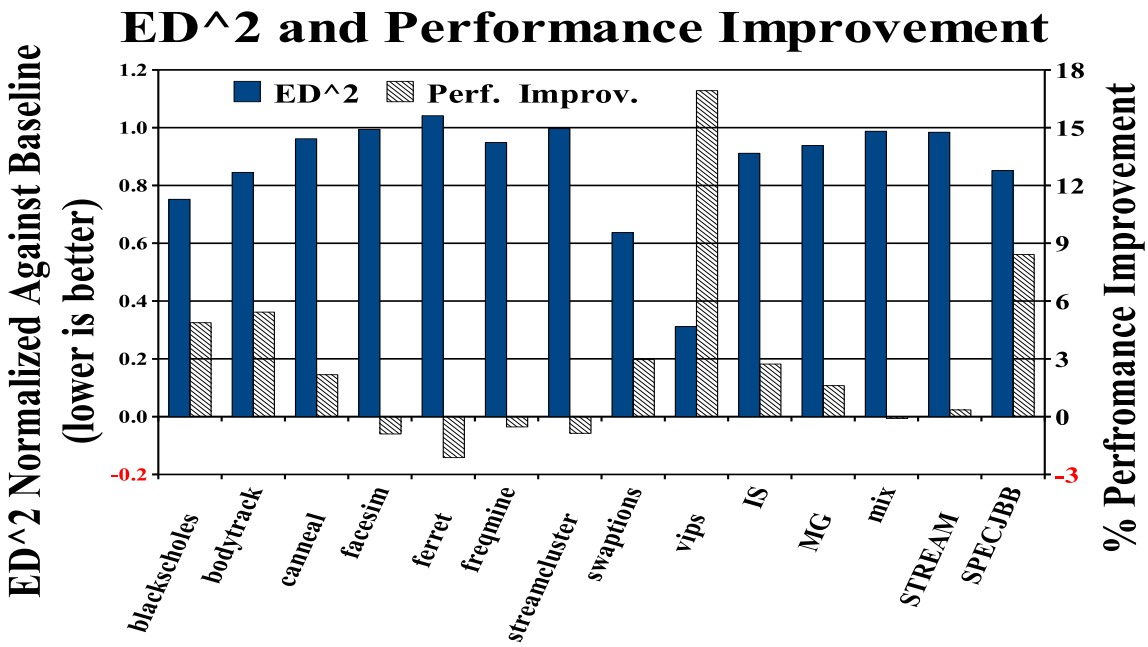


Figure 3.8: Energy Savings ($E * D^2$) for ROPS with 5 Million Cycle Epoch Length

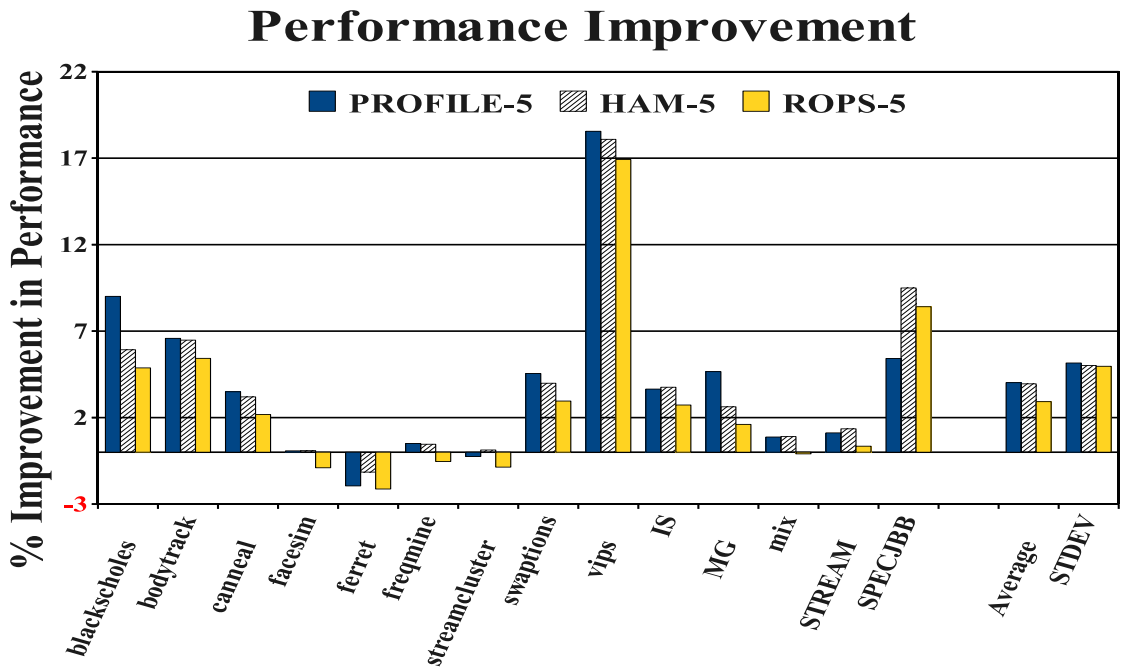


Figure 3.9: Profile, HAM, and ROPS - Performance Improvement for 5 Million Cycle Epoch Length

Figure 3.9 shows the results for Hardware Assisted Migration (HAM) scheme, described in Section 3.3.2, along with ROPS and the profiled scheme, for a 5 million cycle epoch

length. Only two benchmarks suffer performance degradation. This performance penalty occurs because a quickly changing DRAM access pattern does not allow HAM or ROPS to effectively capture the micro-pages that are worth migrating. This leads to high migration overheads without any performance improvement, leading to overall performance degradation. For the Profile scheme, the occasional minor degradation is caused due to nondeterminism of multithreaded workloads, as mentioned earlier.

Compute applications, like those from the NPB suite, some from the PARSEC suite, and the multiprogrammed workload (SPEC-CPU and BioBench suites), usually show low or negative performance change with our proposals. The reason for these poor improvements are the tight loops in these applications that exhibit a very regular data access pattern. This implies there are fewer hot micro-pages within an OS page. Thus, compared to applications like *vips* - that do not stride through OS pages - performance improvement is not as high for these compute applications. The prime example of this phenomenon is the *STREAM* application. This benchmark is designed to measure the main memory bandwidth of a machine, and it does so by loop-based reads and writes of large arrays. As can be seen from the graph, it exhibits a modest performance gain. Applications like *blackscholes*, *bodytrack*, *canneal*, *swaptions*, *vips*, and *SPECjbb* which show substantial performance improvement, on the other hand, all work on large data sets while they access some OS pages (some micro-pages, to be precise) heavily. These are possibly code pages that are accessed as the execution proceeds. The average performance improvement for these applications alone is approximately 9%.

Figure 3.10 plots the $E * D^2$ metric for all these 3 schemes. As before, lower is better. Note that while accounting for energy consumption under the HAM scheme, we take into account the energy consumption in DRAM, and energy lost due to DRAM data migrations and MT look-ups. All but one application perform better than the baseline and save energy for HAM. As expected, Profile saves the maximum amount of energy while HAM and ROPS closely track it. ROPS almost always saves less energy than HAM since TLB misses are costly, both in terms of DRAM access energy and application performance. On an average, HAM saves about 15% on $E * D^2$ compared to baseline with very little standard deviation between the results (last bar in the graph).

Epoch length is an important parameter for the proposed schemes. To evaluate the sensitivity of results to the epoch length, we experimented with many different epoch durations (1 M, 5 M, 10 M, 50 M, and 100 M cycles). The results from experiments with epoch length of 10 M cycles are summarized in Figures 3.11 and 3.12. A comparison

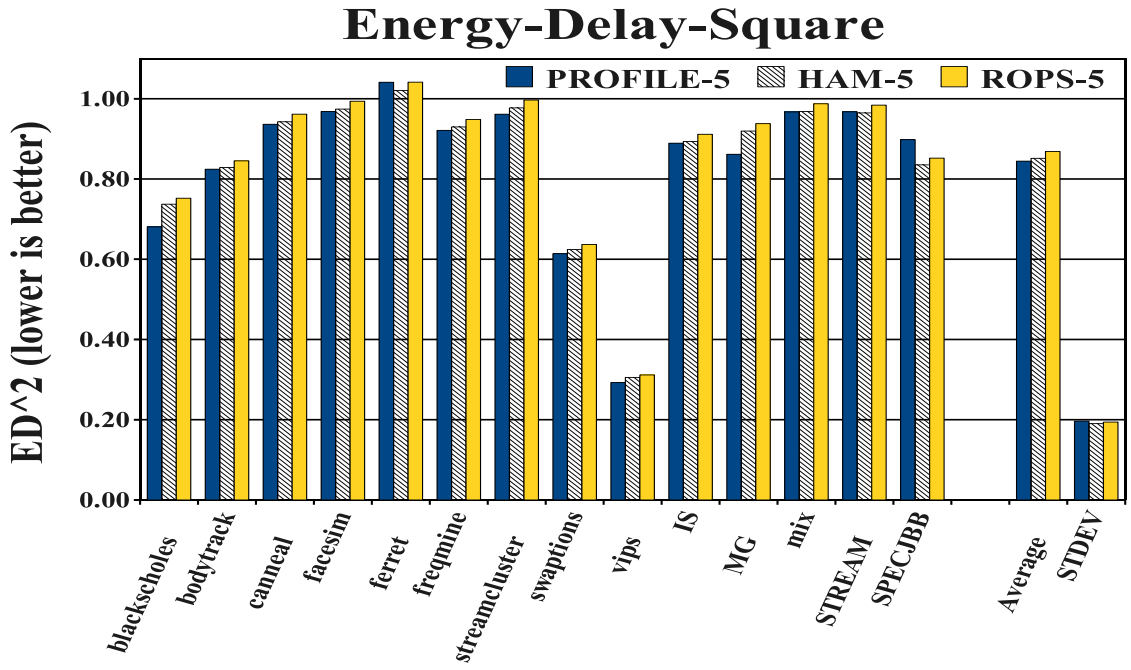


Figure 3.10: Profile, HAM, and ROPS - Energy Savings ($E * D^2$) for 5 Million Cycle Epoch Length

of Figures 3.9, 3.10, 3.11, and 3.12 shows that some applications (such as *SpecJBB*) benefit more from the 5 M epoch length, while others (*blackscholes*) benefit more from the 10 M epoch length. The epoch length is envisioned to be a tunable parameter in software. Since performance and energy consumption are also sensitive to the memory controller's request scheduling policy, we experimented with First-Come First-Serve (FCFS) access policy. As with variable epoch length experiments, the results show consistent performance improvements (which were predictably lower than FR-FCFS policy).

Finally, to show that the simulation interval of 250 million cycles was representative of longer execution windows, the schemes evaluated for a 1 billion cycle simulation window. Figure 3.13 presents the results for this sensitivity experiment. Note that only the performance improvement over baseline is shown as $E*D^2$ results are similar. As can be observed, the performance improvement is almost identical to 250 million cycle simulations.

Results Summary: For both ROPS and HAM, consistent improvement in energy and performance was observed. The higher benefits with HAM are because of its ability to move data without the overhead of TLB shoot-down and TLB misses. For HAM, since updating the MT is not expensive, the primary overhead is the cost of performing the actual DRAM copy. The energy overhead of MT look-up is reduced due to design choices and is marginal compared to page table updates and TLB shoot-downs and misses associated with ROPS.

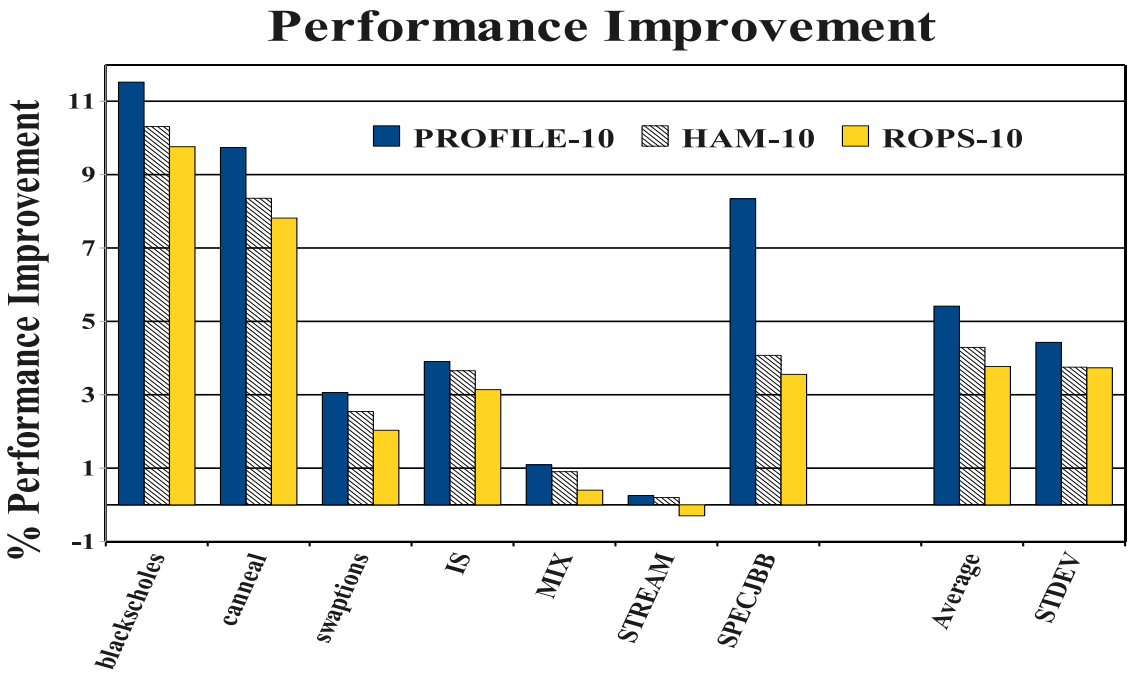


Figure 3.11: Profile, HAM, and ROPS - Performance Improvement for 10 Million Cycle Epoch Length

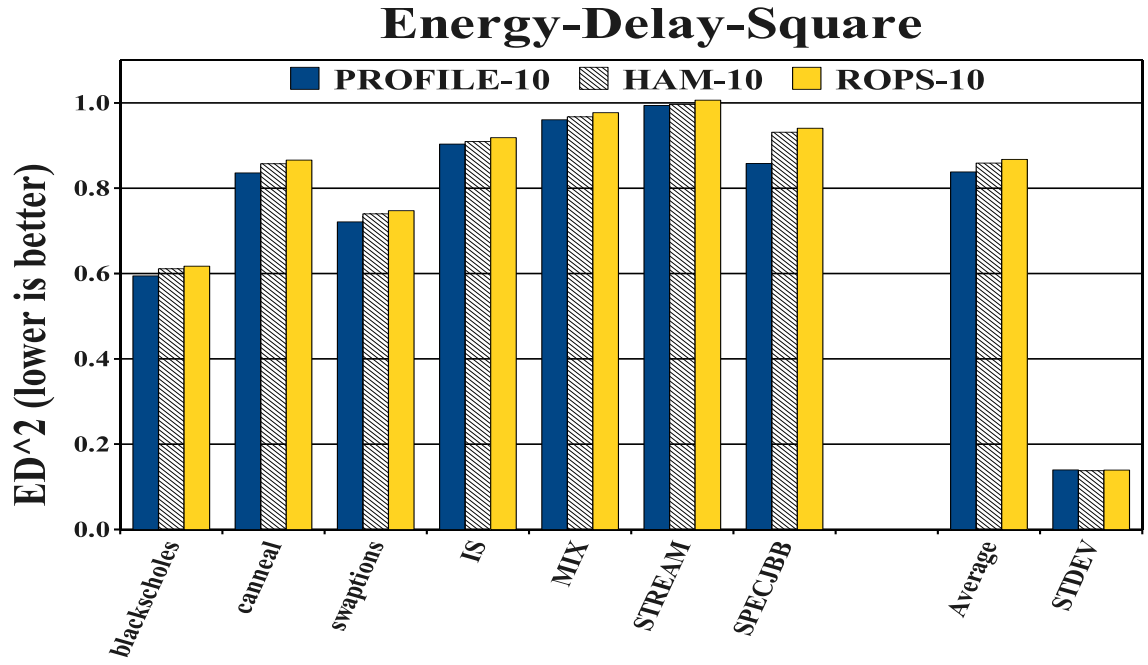


Figure 3.12: Profile, HAM, and ROPS - Energy Savings ($E * D^2$) Compared to Baseline for 10 Million Cycle Epoch.

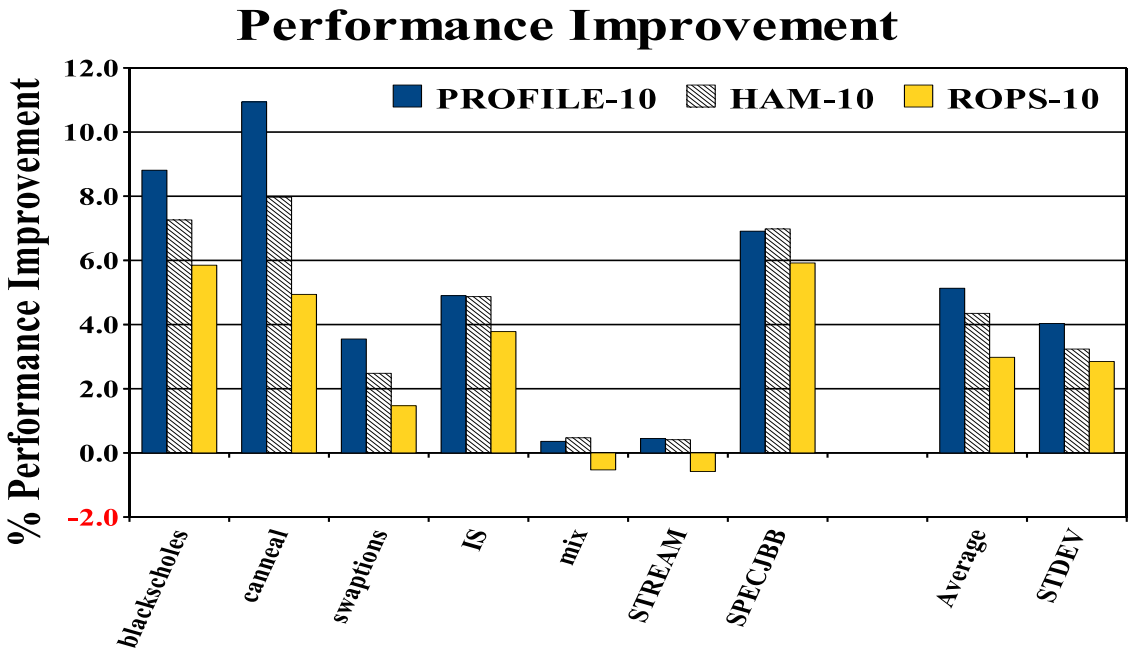


Figure 3.13: Profile, HAM, and ROPS - Performance Improvements for 10M Cycle Epoch Length, and 1 Billion Execution Cycles.

Due to these lower overheads, the HAM scheme performs slightly better than ROPS (1.5% in terms of performance and 2% in energy for best performing benchmarks). The two schemes introduce different implementation overheads. While HAM requires hardware additions, it exhibits slightly better behavior. ROPS, on the other hand, is easier to implement in commodity systems today and offers flexibility because of its reliance on software.

3.5 Related Work

Cuppu et al. [33] first showed that performance is sensitive to DRAM data mapping policy and Zhang et al. [127] proposed schemes to reduce row-buffer conflicts using a permutation-based mapping scheme. Delaluz et al. [36] leveraged both software and hardware techniques to reduce energy while Huang et al. [54] studied it from OS' virtual memory subsystem perspective. Recent work [128, 129] focuses on building higher performance and lower energy DRAM memory systems with commodity DRAM devices. Schemes to control data placement in large caches by modifying physical addresses have also been studied recently [14, 27, 50, 126]. The schemes presented here build on this large body of work and leverage the observations that DRAM accesses to most heavily accessed OS pages are clustered around few cache-line sized blocks.

Page allocation and migration have been employed in a variety of contexts. Several

bodies of work have evaluated page coloring and its impact on cache conflict misses [21, 37, 67, 86, 103]. Page coloring and migration have been employed to improve proximity of computation and data in a NUMA multiprocessor [26, 30, 72, 73, 74, 115] and in a NUCA caches [14, 28, 97]. These bodies of work have typically attempted to manage capacity constraints (especially in caches) and communication distances in large NUCA caches. Most of the NUMA work predates the papers [33, 32, 100] that shed insight on the bottlenecks arising from memory controller constraints. Here, the presented schemes not only apply the well-known concept of page allocation to a different domain, they also extend the policies to be cognizant of the several new constraints imposed by DRAM memory systems, particularly row-buffer re-use.

A significant body of work has been dedicated to studying the effects of DRAM memory on overall system performance [32, 80] and memory controller policies [44, 100]. Recent work on memory controller policies studied effects of scheduling policies on power and performance characteristics [128, 89, 92, 132] for CMPs and SMT processors. Since the memory controller is a shared resource, all threads experience a slowdown when running in tandem with other threads, relative to the case where the threads execute in isolation. Mutlu and Moscibroda [89] observe that the prioritization of requests to open rows can lead to long average queuing delays for threads that tend to not access open rows. This leads to unfairness with some threads experiencing memory stall times that are ten times greater than that of the higher priority threads. That work introduces a Stall-Time Fair Memory (STFM) scheduler that estimates this disparity and over-rules the prioritization of open row access if the disparity exceeds a threshold. While this policy explicitly targets fairness (measured as the ratio of slowdowns for the most and least affected threads), minor throughput improvements are also observed as a side-effect. Such advances in scheduling policies can easily be integrated with our policies to give additive improvements.

The schemes presented here capture locality at the DRAM row-buffer; however, they are analogous to proposals like victim caches [65]. Victim caches are populated by recent evictions, while our construction of an efficient "DRAM region" is based on the detection of hot-spots in the access stream that escapes whatever preceding cache level. The schemes presented in this chapter take advantage of the fact that co-location of hot-spots leads to better row-buffer utilization, while the corresponding artifact does not exist in traditional victim caches. The optimization is facilitated by introducing another level of indirection. Victim caches, on the other hand, provide increased associativity for a few sets, based on application needs. Therefore, it is easy to see that micro-pages and victim caches are

comparable in terms of their design or utility.

3.6 Summary

In this chapter, we attempt to address the issues of increasing energy consumption and access latency being faced by modern DRAM memory systems. Two schemes were proposed that control data placement for improved energy and performance characteristics. These schemes are agnostic to device and signaling standards and therefore, their implementation is not constrained by standards. Both schemes rely on DRAM data migration to maximize hits within a row-buffer. The hardware-based proposal incurs less run-time overhead, compared to the software-only scheme. On the other hand, the software-only scheme can be easily implemented without major architectural changes and can be more flexible. Both schemes provide overall performance improvements of 7-9% and energy improvements of 13-15% for our best performing benchmarks.

CHAPTER 4

TIERED MEMORY

This chapter describes how data placement mechanisms can be used to improve main memory capacity while keeping the memory power budget constant. These mechanisms place frequently accessed data in "active" DRAM ranks while placing infrequently accessed data in DRAM ranks that are predominantly maintained in "low-power" mode.

4.1 Introduction

Main memory capacity and bandwidth are fast becoming dominant factors in server performance. Due largely to power and thermal concerns, the prevailing trend in processor architecture is to increase core counts rather than processor frequency. Multicore processors place tremendous pressure on memory system designers to increase main memory capacity and bandwidth at a rate proportional to the increase in core counts.

The pressure to grow memory capacity and bandwidth is particularly acute for mid- to high-end servers, which often are used to run memory-intensive database and analytics applications or to consolidate large numbers of memory-hungry virtual machine (VM) instances. Server virtualization is increasingly popular because it can greatly improve server utilization by sharing physical server resources between VM instances. In practice, virtualization shares compute resources effectively, but main memory is difficult to share effectively. As a result, the amount of main memory needed per processor is growing at or above the rate of increase in core counts. Unfortunately, DRAM device density is improving at a slower rate than per-processor core counts, and per-bit DRAM power is improving even slower. The net effect of these trends is that servers need to include more DRAM devices and allocate an increasing proportion of the system power budget to memory to keep pace with the server usage trends and processor core growth.

4.1.1 Memory Power Wall

Since DRAM energy efficiency improvements have not kept pace with the increasing demand for memory capacity, servers have had to significantly increase their memory subsystem power budgets [25]. For example, Ware et al. [120] report that in the shift from IBM POWER6 processor-based servers to POWER7-based servers, the processor power budget for a representative high-end server shrank from 53% to 41% of total system power. At the same time, the memory power budget increased from 28% to 46%. Most servers today are forced to restrict the speed of their highest capacity memory modules to accommodate them within the server's power budget [49]. This trend is not sustainable — overall system power budgets have remained constant or decreased and many data centers already operate close to facility power and/or server line-cord power limits. These power constraints are now the primary factor limiting memory capacity and performance, which in turn limits server performance. We call this emerging problem the *memory power wall*. To overcome this memory power wall, system designers must develop methods to increase memory capacity and bandwidth for a fixed power budget.

We propose to attack the problem by exploiting DRAM low-power, idle modes. DDR3 DRAM supports *active power-down*, *fast precharge power-down*, *slow precharge power-down*, and *self-refresh* [7] modes. Each mode progressively reduces DRAM device power by gating additional components within the device, but increases the latency to return to the active state and start servicing requests, introducing a tradeoff between idle mode power reduction and potential increase in access latency.

DRAM low-power modes are not aggressively exploited due to two factors: (i) the coarse granularity at which memory can be put in a low-power mode and (ii) application memory access patterns. Most DRAM-based memory systems follow the JEDEC standard [62]. Memory is organized as modules (DIMMs) composed of multiple DRAM devices. When a processor requests a cache line of data from main memory, the physical address is used to select a channel, then a DIMM, and finally a rank within the DIMM to service the request. All devices in a rank work in unison to service each request, so the smallest granularity at which memory can be put into a low-power mode is a rank. For typical systems built using 4GB DIMMs, a rank is 2 GB in size, a substantial fraction of main memory even for a large server.

The large granularity at which memory can be placed into low-power modes is a serious problem because of the increasingly random distribution of memory accesses across ranks. To increase memory bandwidth, most servers interleave consecutive cache lines in the phys-

ical address space across ranks. Further, without a coordinated effort by memory managers at all levels in the system (library and operating system), application data are effectively randomly allocated across DRAM ranks. Virtualization only exacerbates this problem [60] because it involves yet another memory management entity, the hypervisor. Consequently, frequently accessed data tend to be spread across ranks, so no rank experiences sufficient idleness to warrant being placed in a low-power mode. As a result, few commercial systems exploit DRAM low-power modes and those that do typically only use the deep low-power modes on nearly idle servers.

4.2 An Iso-Powered Tiered Memory Design

To address the memory wall problem, this chapter proposes an *iso-powered tiered memory architecture* that exploits DRAM low-power modes by creating two tiers of DRAM (*hot* and *cold*) with different power and performance characteristics on the same DRAM channel. The tiered memory configuration has the same total memory power budget (iso-power) but allows for larger total capacity. The hot tier is comprised of DRAM in the *active or precharge standby* (full power) mode when idle, while the cold tier is DRAM in *self-refresh* (low power) mode when idle. The cold tier retains memory contents and can service memory requests, but memory references to ranks in the cold tier will experience higher access latencies. To optimize performance, we dynamically migrate data between tiers based on their access frequency. We can also dynamically adjust the amount of DRAM in the hot and cold tiers based on aggregate workload memory requirements.

If one is able to keep most of main memory in a low-power mode, servers with far more memory than traditional designs for the same power budget can be built. An idle rank of DRAM in self-refresh mode consumes roughly one-sixth of the power of an idle rank in active-standby mode, so an iso-powered tiered memory design can support up to six ranks in self-refresh mode or one rank in active-standby mode for the same power budget. The actual ratio of cold ranks per hot rank is dependent on the distribution of memory requests to the hot and cold tiers, and is derived in Section 4.4.1.1. For example, the same power budget can support eight hot ranks and no cold ranks (16GB), four hot ranks and twelve cold ranks (32GB), or two hot ranks and twenty-two cold ranks (48GB). Which organization is optimal is workload-dependent. For workloads with small aggregate working sets and/or high sensitivity to memory access latency, the baseline configuration (16GB of DRAM in the active mode) may perform best. For workloads with large aggregate working sets, an organization that maximizes total memory capacity (e.g., 4GB in active mode and 44GB

in self-refresh mode) may perform best. VM consolidation workloads typically are more sensitive to memory capacity than memory latency, and thus are likely to benefit greatly from the larger capacity tiered configurations. *Our iso-powered tiered memory architecture exploits this ability to trade off memory access latency for memory capacity under a fixed power budget.*

A key enabler of our approach is that for many applications, a small fraction of the application's memory footprint accounts for most of its main memory accesses. Thus, if hot data can be identified and migrated to the hot tier, and cold data to the cold tier, we can achieve the power benefits of tiering without negatively impacting performance.

To ensure hot and cold data are placed in the appropriate tier, we modify the operating system or hypervisor to track access frequency. We divide execution into 20M-100M long instruction epochs and track how often each DRAM page is accessed during each epoch. At the end of each epoch, system software migrates data to the appropriate tier based on its access frequency.

To limit the latency and energy overhead of entering and exiting low-power modes, we do not allow memory requests to wake up cold ranks arbitrarily. Instead, we delay servicing requests to cold ranks when they are in self-refresh mode, which allows multiple requests to a cold rank to be queued and handled in a burst when it is made (temporarily) active. To avoid starvation, we set an upper bound on how long a request can be delayed. The performance impact of queuing requests depends on both the arrival rate of requests to ranks in self-refresh mode and the latency-sensitivity of the particular workload. Our primary focus is on mid- to high-end servers used as virtualization platforms. In this environment, memory capacity is generally more important than memory latency, since DRAM capacity tends to be the primary determinant of how many VMs can be run on a single physical server. Our experiments, discussed in Section 4.5, indicate that occasionally queuing memory requests decreases individual application performance by less than 5%, while being able to support 3X as much total DRAM, increasing aggregate server throughput by 2.2-2.9X under a fixed power budget.

The main contributions of the proposed schemes are the following:

- We propose a simple, adaptive two-tier DRAM organization that can increase memory capacity on demand within a fixed memory power budget.
- We demonstrate how variation in page access frequency can be exploited by the tiered memory architecture.

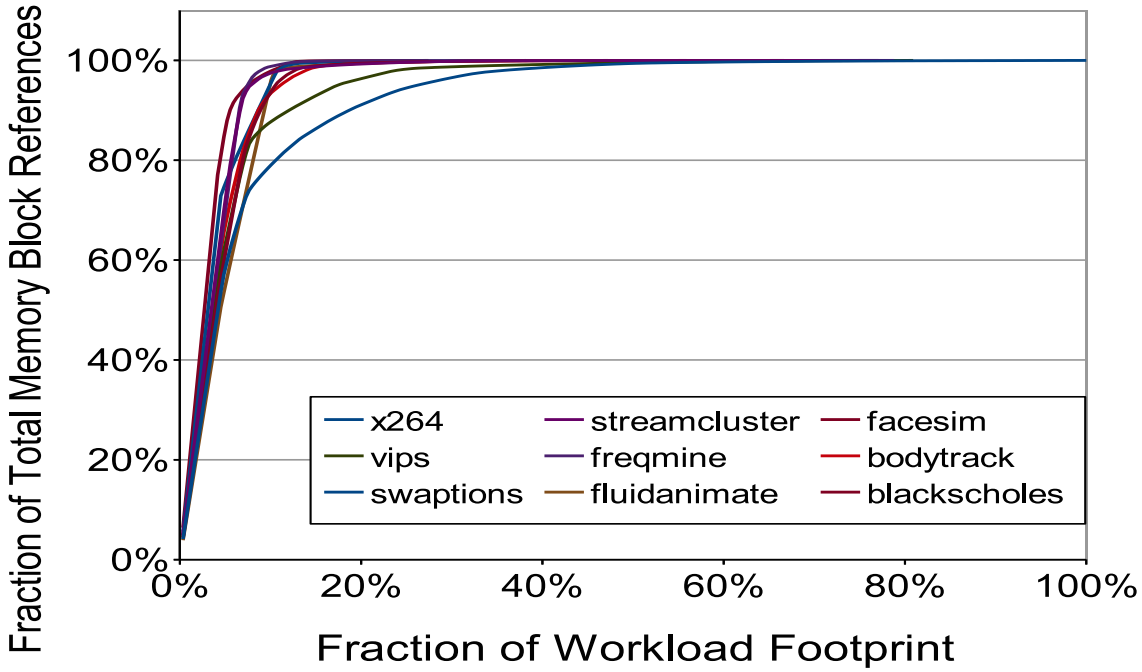
- We analyze the proposed iso-powered tiered memory architecture to quantify its impact on server memory capacity and performance.

4.3 Memory Access Characteristics of Workloads

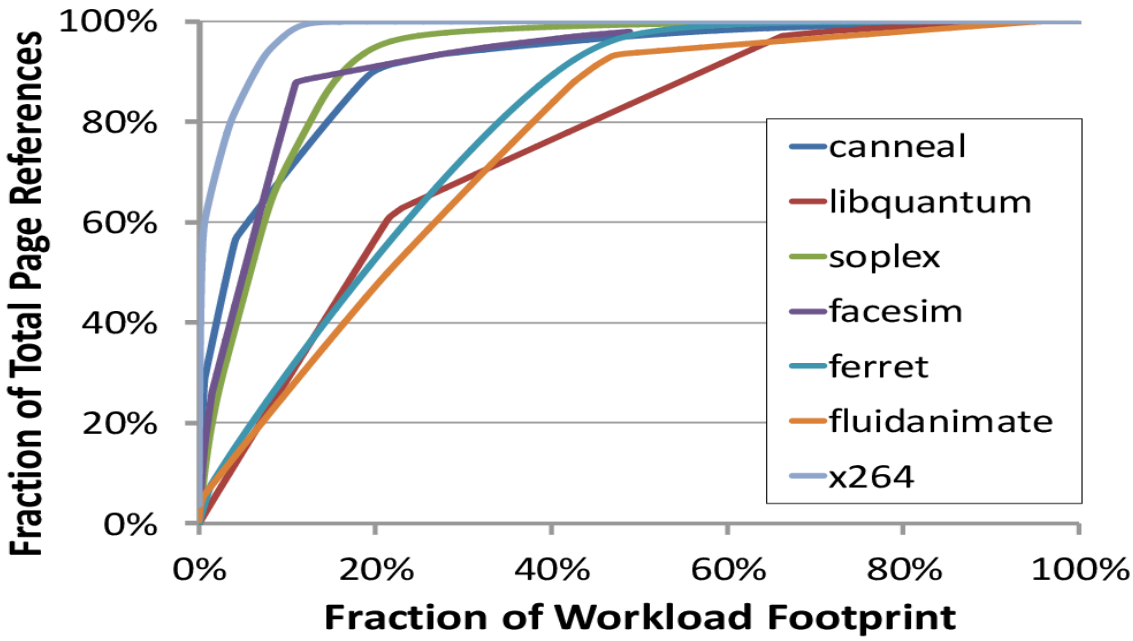
Our iso-powered tiered memory architecture identifies hot/cold pages in an application's address space, migrates them into the appropriate tiers, and exploits this differentiated access characteristics to increase total memory capacity. This is only feasible if applications exhibit strong access locality, wherein a relatively small fraction of memory is responsible for a large fraction of dynamic accesses. To determine if that is so, we first examined a set of applications from the PARSEC suite using their *native* inputs. We instrumented the OS to track references to memory blocks every 10 msecs at a 128 MB granularity - the first access to a block in a 10 ms interval would increment the reference count for that block. Figure 4.1(a) presents a cumulative distribution function (CDF) of the block reference counts showing the fraction of workload footprint needed to cover a given fraction of block reference counts for each workload. While these data are collected at a fairly coarse spatial (128 MB) and temporal granularity (10 msec), it is evident that there is substantial memory access locality. In all cases, 10-20% of each application's memory accounts for nearly all accesses, which is sufficient locality for iso-powered tiered memory to be effective.

To determine whether tracking reference frequency at a small granularity would demonstrate similar locality, we reran our experiments in an event-driven system simulator (described in detail in Section 4.5.2) and tracked accesses at a 4 KB OS page granularity. Figure 4.1(b) plots the resulting memory access CDFs for seven applications. The finer granularity of simulator-measured reference locality data shows a similar spatial locality in the memory region references as we found with the coarse grained measurements on existing hardware.

Figure 4.2 shows the distribution of accesses to individual 4 KB pages sorted from the most frequently accessed page to the least, for two of the most memory-intensive PARSEC and SPEC CPU2006 benchmarks, `canneal` and `libquantum`. Both applications exhibit traditional "stair step" locality, with sharp knees in their access CDF graphs that indicate the size of individual components of each application's working set. The cumulative working set sizes where steep transitions occur are good candidate hot tier sizes if our hot/cold data migration mechanism can effectively migrate hot/cold data to the appropriate tier.

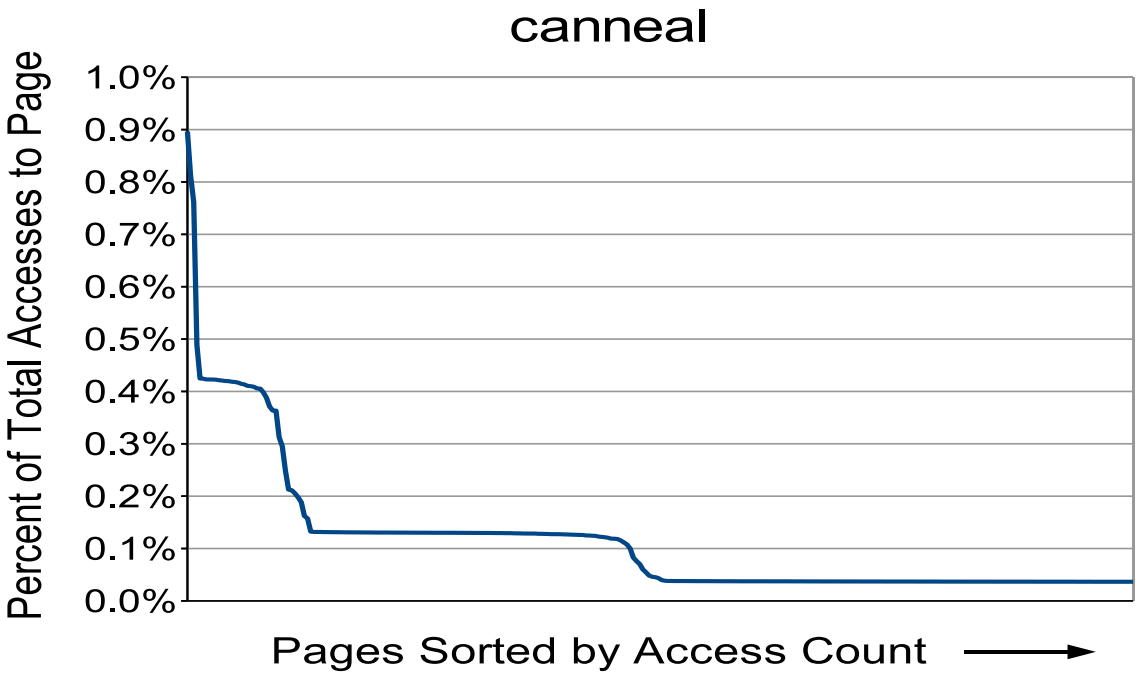


a. *Real-system memory block reference distribution at 128MB granularity for select PARSEC workloads*

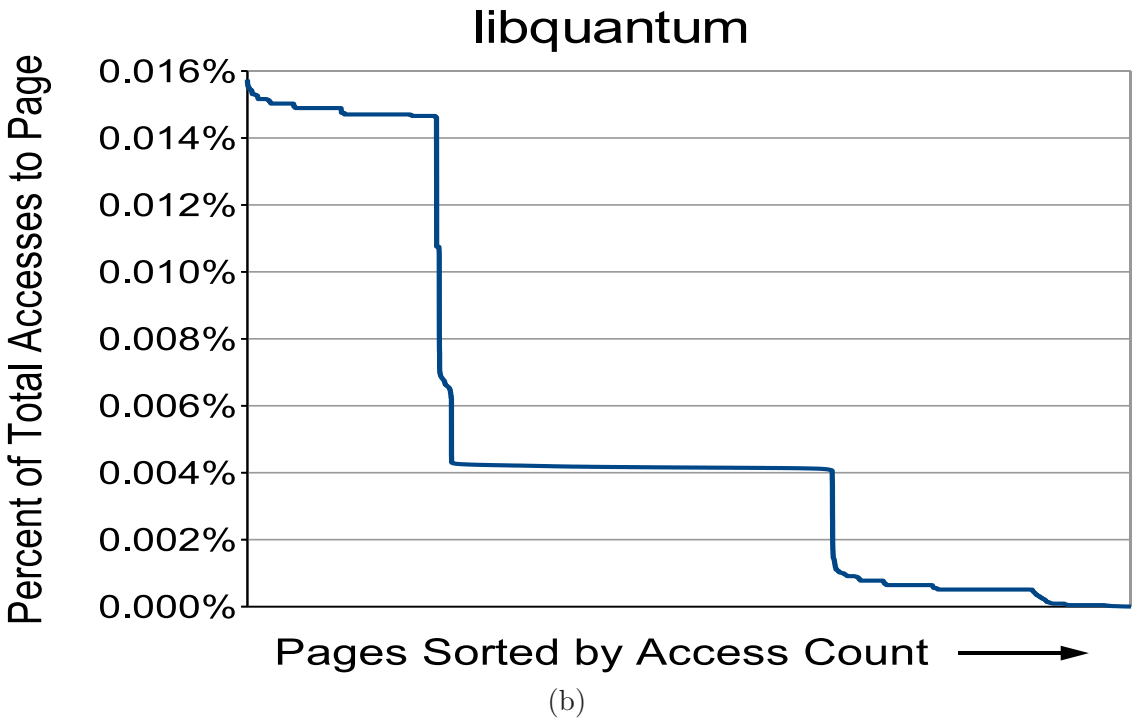


b. *Simulated memory access count distribution at 4 KB page granularity for select PARSEC and SPEC CPU2006 workloads*

Figure 4.1: Cumulative Access Distribution of DRAM Memory Requests for Different Workloads.



(a)



(b)

Figure 4.2: Page Access Distribution at 4 KB Granularity Measured on Our Simulator.

4.4 Implementation

Tiered memory is a generalizable memory architecture that leverages the heterogeneous power-performance characteristics of each tier. For instance, there is a large body of recent work on hybrid memory systems that mix multiple memory technologies (e.g., DRAM and PCM) [15, 38]. Our work introduces the notion of DRAM-only tiered designs, which can be implemented with existing memory technologies. We use the self-refresh mode, which provides the maximum reduction in idle DRAM power, but also the highest exit latency, to create our cold tier of memory. While we could in theory support more than two tiers, the incremental benefit of using higher power idle modes is marginal and increases implementation complexity.

4.4.1 Implementing Tiered Memory in a DRAM Memory System

We implement iso-powered tiered memory by dividing memory into two tiers, a *hot tier* in the active state and a *cold tier* in self-refresh mode. To balance the goal of maximizing power savings while mitigating the performance impact, we can adjust the size of the hot and cold tiers based on the size and locality of the working sets of running applications. For some application mixes, optimal iso-power performance is achieved by placing as many ranks as possible given the power budget in the hot tier, and turning off the remainder of DRAM. Referring to our example from Section 4.1, this would be the organization with eight ranks (16 GB) in the hot tier and no ranks in the cold tier. Other application mixes might have large footprints but very high spatial locality, and thus achieve the best performance by prioritizing capacity, e.g., the two hot rank (4 GB) and twenty-two cold rank (44 GB) example from Section 4.1.

In our proposed design, servers are configured with enough DIMMs to store the maximum-sized iso-power configuration supported, which in the example above would be 48 GB. For the purposes of this work, we assume that these ranks (DIMMs) are allocated evenly across available memory channels. The operating system (or hypervisor) determines how many ranks should be in each power mode given the memory behavior of the current application mix, subject to a total memory system power budget. Since we only consider one application mix at a time, we do not dynamically resize the tiers midrun, but rather evaluate the performance of several iso-power hot/cold tier configurations. In Section 4.6, we discuss other possible organizations, e.g., modifying the number of memory channels or memory controllers, but they do not have a substantial impact on our conclusions.

4.4.1.1 Iso-Powered Tiered Memory Configurations

As a first step, we need to determine what mix of hot and cold ranks can be supported for a given power budget. Assume a baseline (all hot ranks) design with N ranks. An iso-powered tiered configuration can support a mix of N_{hot} hot ranks and N_{cold} cold ranks ($N_{hot} \leq N \leq N_{hot} + N_{cold}$). Whenever we adjust the number of active ranks, we do so evenly across all memory channels and memory controllers, so the number of hot/cold/off ranks per memory channel increases/decreases in direct proportion to changes in tier size and aggregate capacity. This design imposes the fewest changes to current memory controller designs, but other designs are discussed in Section 4.6.

Note that the number of hot/cold ranks is not the only factor that impacts power consumption — power is also consumed by actual memory operations. To avoid exceeding the power budget, we employ well-known memory throttling techniques [49] to limit the rate of accesses to the memory system. We can adjust the active power budget up/down by decreasing/increasing the memory throttle rate. The power allocated for memory accesses impacts the number of hot and cold ranks that can be supported for a given power budget. Note that no actual throttling was needed during our evaluations as the aggregate bandwidth demand for the memory tiers was low enough to be met within the memory system power budget.

We compute iso-powered tiered memory configurations based on the desired peak memory request rate (r_{peak}), average service time (t_{svc}), the idle power of hot (p_{hot}) and cold (p_{cold}) ranks of DRAM, the fraction of requests expected to be serviced by the hot tier (μ), the amount of time we are willing to queue requests destined for the cold tier (t_q), and the latency of entering and exiting the low-power mode cold tier (t_e).

Using p_{svc} for average power to service a request, we can represent the power equivalence as follows;

$$
\begin{aligned}
\textit{Tiered memory power} &= \textit{Conventional memory power} \\
p_{hot}\ N_{hot} \ +\ (p_{svc} - p_{hot})\ \mu\ r_{peak}\ t_{svc} \ &+\ p_{cold}\ N_{cold} \ + \\
(p_{svc} - p_{cold})\ (1-\mu)\ r_{peak}\ t_{svc} \ &+\ \frac{(p_{hot} - p_{cold})\ N_{cold}\ t_e}{t_q\ (1-(1-\mu) r_{peak} t_{svc}/N_{cold})} \\
&= p_{hot}\ N \ +\ p_{svc}\ r_{peak}\ t_{svc}
\end{aligned} \tag{4.1}
$$

After canceling out of the active power components (terms with p_{svc}), the baseline idle power can redistributed among a greater number of ranks by placing idle cold ranks in low-power mode:

$$p_{hot}N_{hot} \; - \; p_{hot}\mu r_{peak}t_{svc} \; + \; p_{cold}\; N_{cold} \; - \; p_{cold}(1-\mu)r_{peak}t_{svc} + \frac{(p_{hot}-p_{cold})\; N_{cold}\; t_e}{t_q(1-(1-\mu)r_{peak}t_{svc}/N_{cold})} \;\; = \;\; p_{hot}\; N \quad (4.2)$$

If we define P_{ratio} to be the ratio of the idle power of hot and cold ranks (p_{hot}/p_{cold}) and T_{ratio} to be the ratio of the transition latency to the maximum queuing delay (t_e/t_q), Equation 2 can be reduced to an expression of the number of cold ranks (N_{cold}) that can be supported for a given number of hot ranks (N_{hot}):

$$N_{cold} = \frac{P_{ratio}\;(N-N_{hot}) - (P_{ratio}-1)\;(1-\mu)\;r_{peak}\;t_{svc}\;(1-T_{ratio})}{(1+(P_{ratio}-1)\;T_{ratio})} \quad (4.3)$$

P_{ratio}, r_{peak}, are constants and t_{svc} values are in a fairly narrow range for any given system design. μ is application-dependent. Given their values, we can solve for N_{cold} as a function of N_{hot} for a particular point in the design space. Figure 4.3 depicts the resulting set of possible memory configurations that can be supported for a particular access pattern (μ) for various DRAM technologies (P_{ratio}). For this analysis, we assume a baseline design with eight ranks ($N_{hot} = 8,\; N_{cold} = 0$) and a DRAM service time of one-quarter the DRAM activate-precharge latency. The range of hot-to-cold idle power ratios shown along the x-axis (6-10) is typical for modern DRAM devices [7]. As can be seen in Figure 4.3, we can support larger tiered memory sizes by using deeper low power modes (increasing P_{ratio}), and when more accesses hit the hot tier (increasing μ).

We use this model to identify iso-powered tiered memory configurations of interest. For our analysis, we consider two configurations: (1) one with four hot ranks and twelve cold ranks ($N_{hot}, N_{cold} = (4, 12)$) and (2) one with two hot ranks and twenty-two cold ranks ($N_{hot}, N_{cold} = (2, 22)$). Both of these configurations fit in the same power budget even when only 70% of memory accesses hit the hot rank ($\mu = 0.7$) and when the ratio of hot-to-cold idle power is only six. Note that throttling ensures that even applications with particularly poor locality (μ under 0.7) operate under the fixed power budget, although they will suffer greater performance degradation.

4.4.1.2 Servicing Requests from Tiered Memory

Memory requests to hot tier ranks are serviced just as they would be in a conventional system. For ranks in the cold tier, requests may be queued briefly to maximize the time spent in DRAM low-power mode and amortize the cost of changing power modes. Figure 4.4 shows how requests to the cold tier are handled. When there are no pending requests destined for a particular cold rank, it is immediately put in self-refresh mode. If a request arrives at a rank in self-refresh mode, e.g., at time t_0 in Figure 4.4, it is queued for up

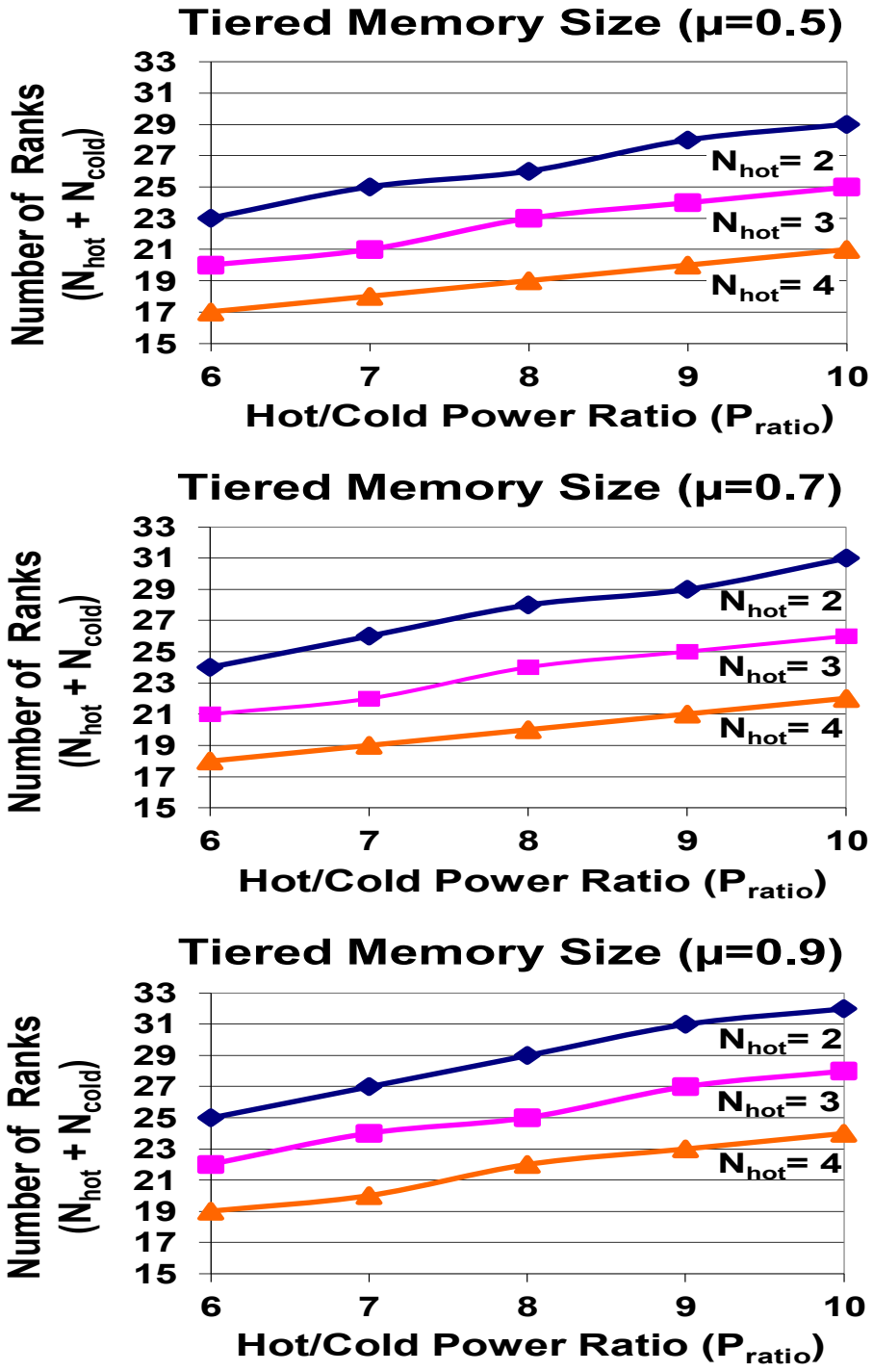


Figure 4.3: Iso-Powered Tiered Memory Capacity Relative to an 8-rank Conventional System.

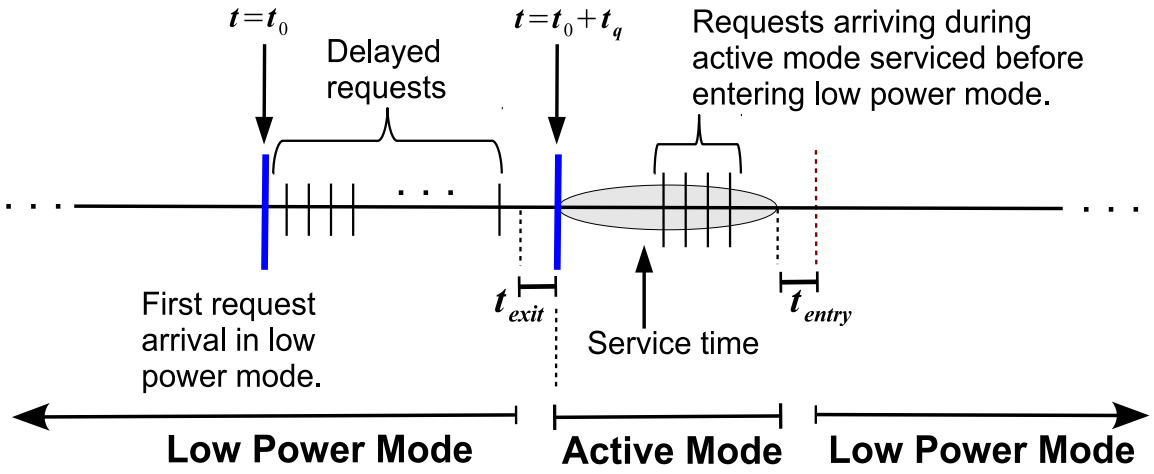


Figure 4.4: Cold Tier Rank Mode Changes with Request Arrival.

to t_q seconds. Any other requests that arrive during this interval are also queued. After experimenting with different values of t_q, we use a fixed value of $t_q = 10 * t_{exit} + t_{entry}$ for our experiments. We found this value to be long enough to amortize entry/exit overhead, but also small enough to not trigger software error recovery mechanisms.

To support this mechanism, we augment the memory controller DRAM scheduler with per-rank timers. When a request arrives at a rank in self-refresh mode, the timer is started. When it reaches $t_q - t_{exit}$, the controller issues the DRAM required to begin transition of the rank from self-refresh to active mode. The rank remains in the active mode as long as there are requests pending for it. Once all the pending requests are serviced, the rank is transitioned back into low-power mode.

4.4.1.3 Identifying Hot and Cold Pages

Our tiered memory mechanism is most effective when most accesses are made to the hot tier, so we track page access frequency and periodically migrate pages between the hot and cold tiers based on their access frequency. To identify hot/cold pages, we assume that the memory controller is augmented to support a version of the access frequency tracking mechanism proposed by Ramos et al. [99], which builds on the Multi Queue algorithm [130] (MQ) first proposed for ranking disk blocks.

The algorithm employs M Least Recently Used (LRU) queues of page descriptors (q_0-q_{M-1}), each of which includes a page number, reference count, and an expiration value. The descriptors in q_{M-1} represent the pages that are most frequently accessed. The algorithm also maintains a running count of total DRAM accesses ($Current$). When a page is first accessed, the corresponding descriptor is placed at the tail of q_0 and its expiration value

initialized to $Current + Timeout$, where $Timeout$ is the number of consecutive accesses to other pages before we expire a page's descriptor. Every time the page is accessed, its reference count is incremented, its expiration time is reset to $Current + Timeout$, and its descriptor is moved to the tail of its current queue. The descriptor of a frequently used page is promoted to from q_i to q_{i+1} (saturating at q_{M-1}) if its reference count reaches 2^{i+1}. Conversely, the descriptor of infrequently accessed pages are demoted if their expiration value is reached, i.e., they are not accessed again before $Timeout$ requests to other pages. Whenever a descriptor is promoted or demoted, its reference count and expiration value are reset. To avoid performance degradation, updates to these queues are performed by the memory controller off the critical path of memory accesses, using a separate queue of updates and a small on-chip Synchronous Random Access Memory (SRAM) cache.

4.4.1.4 Migrating Pages Between Hot and Cold Tiers

To ensure that frequently accessed pages are kept in the hot tier, we periodically migrate pages between tiers based on their current access frequency. We employ an epoch-based scheme wherein the hypervisor uses page access information to identify pages that should be migrated between the tiers. Using this access information at the end of every epoch, a migration daemon moves pages between the tiers such that the most frequently accessed pages among both the tiers are placed in the hot tier for the subsequent epoch, pushing out less frequently accessed pages if needed to the cold tier.

To preserve data coherence, the hypervisor first protects any pages being migrated by restricting accesses to them for the duration of the migration and updates the corresponding page table entries. Once migration is complete, the hypervisor shoots down the stale TLB entries for the migrated pages and flushes any dirty cache lines belonging to that page. The first subsequent access to a migrated page will trigger a TLB fault and the first subsequent access to each cache line in a migrated page will suffer a cache miss, both of which we faithfully model in our simulations. A hardware block copy mechanism could reduce much of the overhead of migration, but we do not assume such a mechanism is available.

An important consideration is how long of an epoch to employ. Long epochs better amortize migration overhead, but may miss program phase changes and suffer when suddenly "hot" data are kept in the cold tier too long. Another consideration is whether to limit the amount of migration per epoch. We found that a 40M-cycle epoch length and a limit of 500 (4KB) page migrations per epoch achieved a good balance. We discuss the impact of these choices (epoch length and per-epoch migration cap) in Section 4.5.

4.5 Results

4.5.1 Evaluation Metric

Our focus is on power- and memory-constrained VM consolidation scenarios, which are increasingly common in data centers. We assume that all other resources are plentiful and do not impact VM performance. Memory bandwidth, CPU resources, and other shared resources are under-utilized in typical consolidation scenarios; e.g., Kozyrakis et al. found that large on-line services consume less than 10% of provisioned bandwidth, but close to 90% of memory capacity [71]. Thus, in our analysis, we assume that power limits how much memory a server can support and the amount of memory available is the main constraint on the number of VMs that can be run effectively on a server.

The goal of our work is to maximize aggregate server performance for a given power budget, which involves balancing the total amount of memory present in a server, which determines how many VMs can be loaded, against the amount of "hot" memory present, which affects per-VM performance. We evaluate iso-powered tiered memory on its ability to boost aggregate server performance in memory-constrained virtualized environments by increasing the available memory within a given memory system power budget. We evaluate the effectiveness of iso-powered tiered memory by simulating the performance of individual VMs when they are allocated different amounts of memory, which corresponds to allocating different numbers of VMs on a server with a fixed memory size. We also simulate the impact of memory being divided into hot and cold tiers and the overhead of managing the tiers, both of which impact individual VM performance.

As a baseline, we first measure the memory footprint and performance of a single VM in a virtualized environment. In the following discussion, let X be the memory footprint of a particular VM instance, measured in 4 KB sized pages. If the VM instance is allocated at least X pages, it fits entirely in main memory and never suffers from a page fault. When we run more VMs than can fit entirely in a server's main memory, we allocate each VM a proportional fraction of memory. For example, if we run 2 VMs, one with an X-page memory footprint and one with a 2X-page memory footprint, on a server with X pages of memory, we allocate $\frac{1}{3}X$ to the first VM and $\frac{2}{3}X$ to the second.

We report aggregate system performance when running N VMs as:

$$Aggregate\ System\ Performance = Individual\ VM\ Performance * Number\ of\ VMs$$

Thus, to determine the aggregate system performance for a given tiered memory configuration, we first determine the corresponding per-VM memory capacity and then employ a cycle-accurate simulator to determine the performance of a single VM with this allocation.

Note again that we assume that memory capacity is the primary limit on the number of VMs that a server can support. Thus, we simulate VMs running in isolation and do not model contention for cores, memory controllers, or memory channels. This model is representative of typical consolidation scenarios, where the number of VMs ready to execute at any given moment is less than the number of cores and contention for memory controller and memory channel resources does not substantively impact performance. This approach allows us to evaluate points in the design space that are infeasible to simulate on a high fidelity full system simulator, e.g., very large numbers of VMs running on large tiered memory configurations.

To simplify the discussion and enable easier cross-experiment comparison, our baseline configuration entails 100 VMs running on server with a nontiered memory of size of 100X. The specific value of X is workload-dependent but unimportant — it simply denotes a baseline scenario where we allocate as many VMs as fit in the server's physical memory without paging. All other results are normalized to this baseline configuration.

4.5.2 Experimental Infrastructure and Methodology

We evaluate our proposal using the Mambo full system simulator [23]. Each VM is assigned a single out-of-order core. We employ a cycle-accurate power-performance model of a high-performance single-channel memory system with fully-buffered, DDR3 memory. Details of the core and memory system modeled are listed in Table 4.1.

We evaluate applications from the PARSEC [22] and SPEC-CPU 2006 benchmark suites [51]. We use the *simlarge* inputs for PARSEC and the *ref* inputs for SPEC. For the SPEC benchmarks, we simulate 5 billion instructions after appropriate warmup. For each PARSEC benchmark, we use the specified Region-of-Interest (ROI) [22].

To determine the baseline performance of each workload, we model just enough memory to hold the entire workload image without paging. Each workload's pages are distributed uniformly over the available ranks. We model a 4KB interleave granularity so each page fits within a single rank. To model scenarios with less memory, we simply reduce the simulated number of rows in each DRAM device without altering any other memory system parameters, as explained below. As an example, consider an application with an 80 MB (20,000 4KB pages) memory footprint. In the baseline configuration (100 VMs running on a server with 100X DRAM capacity), the VM's 80MB footprint would be uniformly distributed across 8 ranks of active DRAM. To model 125 VMs executing on the same memory configuration, we model the VM running on a system with 64MB of DRAM (80MB divided by 1.25) equally distributed across the 8 ranks of active DRAM. The VM would

Table 4.1: Simulator Parameters

Core Parameters:

ISA	Power ISA
Core Freq.	4 GHz
L1 I-cache	32KB/4-way, 1-cycle
L1 D-cache	32KB/8-way, 1-cycle
L2 Cache	256KB/8-way, 5-cycle
L3 Cache	4 MB 26-cycle

DRAM Device Power Parameters:

DRAM cycle_time	1.5 ns	V_{DD}	1.5 V
I_{DD0}	130 mA	I_{DD2P} (Slow)	12 mA
I_{DD2P} (Fast)	40 mA	I_{DD2N}	80 mA
I_{DD3P}	65 mA	I_{DD3N}	95 mA
I_{DD4R}	255 mA	I_{DD4W}	330 mA
I_{DD5A}	305 mA	I_{DD6}	10 mA
watt per DQ	6.88 mW	num DQS	1

Memory System Parameters:

Memory Type	Fully-Buffered, DDR3-1333
Memory Channel Bandwidth	6.4 GHz (2B read + 1B write) channel
Buffer-Chip	Dual ported, each supporting 4, 64-bit ranks
DRAM Device Parameters	Micron MT4741J512M4 DDR3-1333 x4 part [7] t_{CL}=t_{RCD}=t_{RP}=14ns (9-9-9) 8 banks/device, 32K rows/bank, 2K cols/row
Page Fault Penalty	1 msec
Baseline organization	8, 2-port ranks
Tiered memory config 1	24, 2-port ranks (2-hot, 22-cold)
Tiered memory config 2	16, 2-port ranks (4-hot, 12-cold)

suffer page faults whenever it touched a page of memory not currently in its 64MB DRAM allocation, in which case we model LRU page replacement.

The simulator models every DRAM access to determine if it is to a hot rank, a cold rank, or a page not currently present in memory (thereby causing a page fault). The simulator emulates the tiered memory architecture described in Section 4.4, including the migration of data between tiers based on usage. Every DRAM rank is associated with a queue of pending requests. Requests at a hot rank are serviced as soon as the channel and rank are free. Requests to a cold rank that is currently in the low-power state are queued, but as described in Section 4.4, no request is delayed more than t_q DRAM cycles. The simulator also models cross-tier data migration. The migration epoch length is configurable, but unless otherwise specified, all experiments used a 40M CPU cycle (10ms) epoch. Sensitivity to epoch length is presented below.

To track which pages are resident in memory, the simulator emulates OS page allocation and replacement. We model random page allocation, which corresponds to situations where the OS free page list is fragmented, as is common on servers that have been executing for more than a trivial amount of time. We conservatively model a low (1 msec) page fault service time, e.g., by using a Solid State Disk (SSD), but our results are not sensitive to higher service times, e.g., due to using a disk paging device, since performance degradation is severe even with a 1 msec service time.

4.5.3 Evaluation

Using the analytic model described in Section 4.4.1.1, we identified three candidate iso-power memory configurations:

- **Baseline**: 8 hot ranks and 0 cold ranks
- **2X-capacity**: 4 hot ranks and 12 cold ranks
- **3X-capacity**: 2 hot ranks and 22 cold ranks

As seen in Figure 4.3, even with only a 6:1 hot:cold idle power ratio, all three of these configurations are iso-power when at least 70% of requests are serviced by the hot tier ($\mu \geq 0.7$). The 2X-capacity configuration is iso-power even down to $\mu = 0.5$, while the 3X-capacity configuration needs $\mu > 0.5$. For the 2X- and 3X-capacity configurations, the additional ranks on the same channels as the baseline ones will see higher access latency [59], an effect that our simulator models. Recall that the total memory capacity of the baseline model is enough to hold 100 VMs in their entirety; the 2X- and 3X-capacity configurations

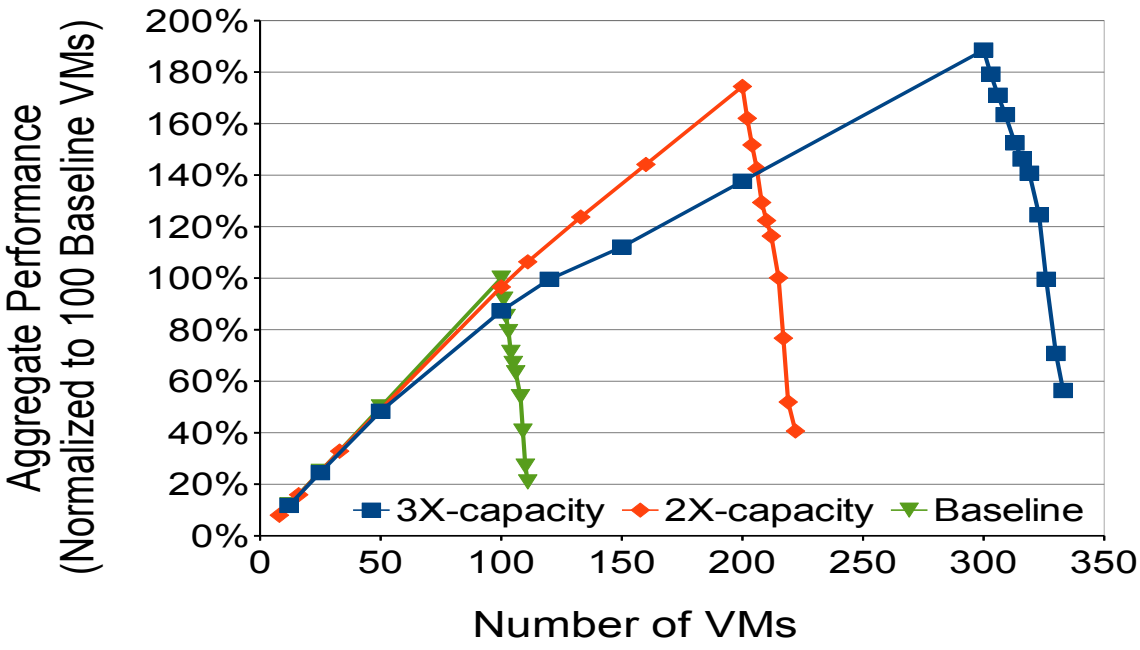
have enough memory to hold 200 and 300 VMs, respectively, but a large portion of this memory is in self-refresh mode.

Figure 4.5 presents the results for one representative workload, *libquantum*, for each modeled memory configuration. Figure 4.5(a) plots normalized aggregate performance. As expected, *baseline* performance scales linearly up to 100 VMs, after which performance drops off dramatically due to the high overhead of page faults. The performance of the 2X-capacity configuration scales almost perfectly up to 200 VMs, where it achieves 176% of the aggregate performance of the baseline organization, after which it also suffers from high page fault overhead. The 3X-capacity configuration is able to squeeze out a bit more aggregate performance, 188% when 300 VMs are loaded.

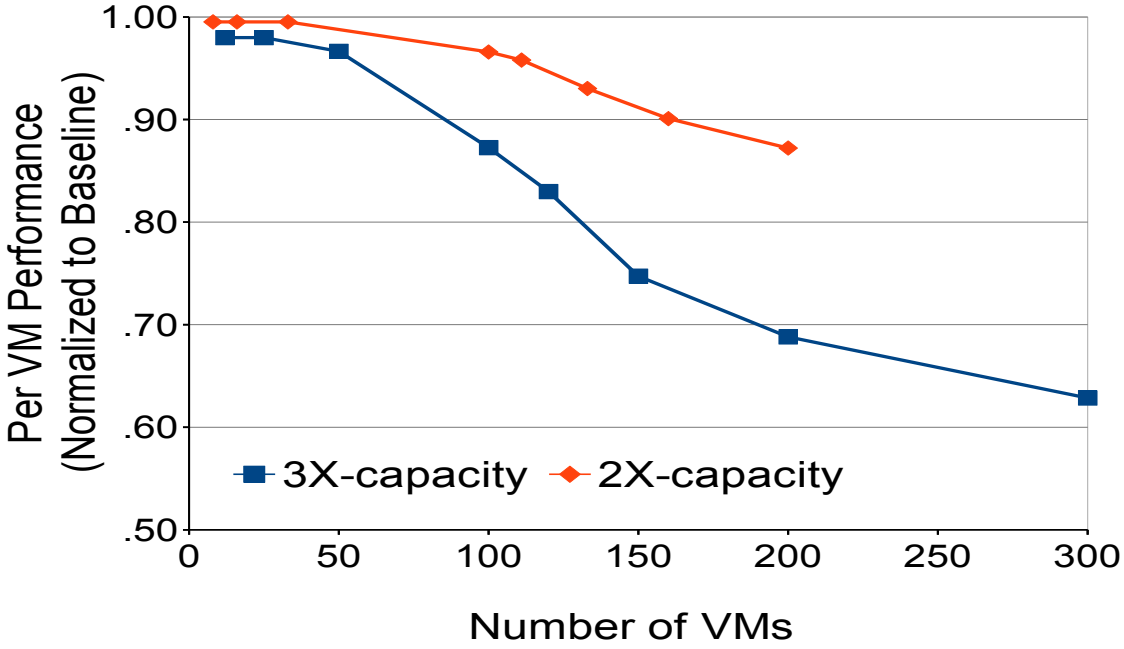
There are several important trends to note in these results. First, there is a clear drop in the relative benefit of tiering going from the 2X- to 3X-capacity organizations. Figure 4.5(b) illustrates this effect — it plots per-VM performance for the 2X- and 3X-capacity configurations. The drop-off in per-VM performance in the 2X-capacity configuration is modest, never more than 15%, because 4 hot ranks are enough to capture most accesses. In contrast, with only 2 hot ranks, per-VM performance for the 3X-capacity configuration drops as much as 37% at 300 VMs. The impact of slower accesses to cold tiers causes a clear drop in benefit as we move to more aggressive tiering organizations.

Another trend to note in Figure 4.5 is the gap between the baseline and 3X-capacity results at 100 VMs and the gap between the 2X- and 3X-capacity results at 200 VMs. These gaps motivate a tiered memory architecture that adjusts the size of the hot and cold tiers as demand for memory capacity grows and shrinks. Under light load, baseline performance is best and 8 ranks should be kept hot, with the remainder turned off. As load increases, the system should shift to the 2X-capacity configuration, which continues to scale well up until 200 VMs are present. Finally, the system should shift to the 3X-capacity configuration under the heaviest loads. Overall, system performance with such a design should track the convex hull of the three curves. While we do not simulate the ability to shift aggregate capacity based on demand, it can be realized by using (capacity) page fault misses to trigger increasing memory capacity by adding more cold ranks and suitably reducing the number of hot ranks..

Figure 4.6 presents the results for the remaining applications that we evaluated. The relatively smaller gaps between the baseline and 3X-capacity results at 100 VMs indicate that these applications are not as sensitive as `libquantum` to slower cold tier accesses. Figure 4.7 presents the peak aggregate performance for all seven benchmarks for the 2X-



a. *Aggregate Performance (libquantum)*



b. *Per VM Performance (libquantum)*

Figure 4.5: Behavior of `libquantum` on a Tiered Memory System

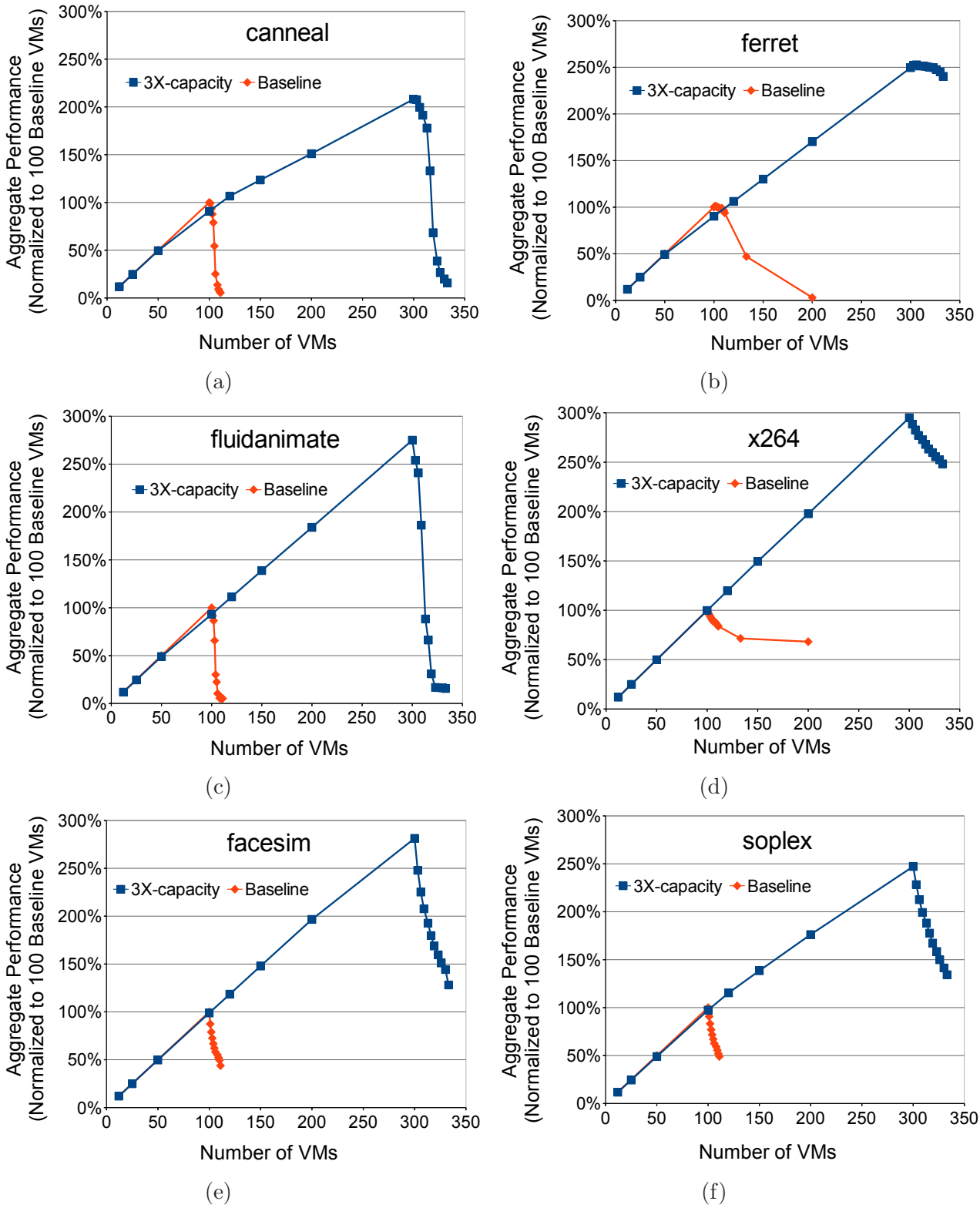


Figure 4.6: Aggregate Performance for Baseline, and 3X-capacity Configurations

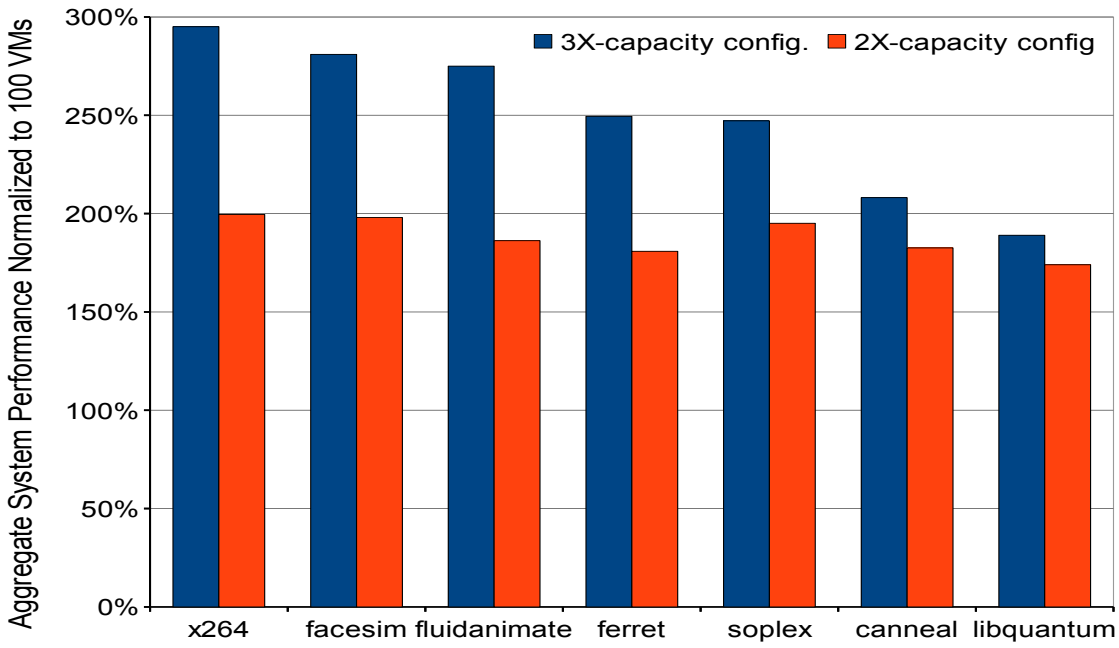
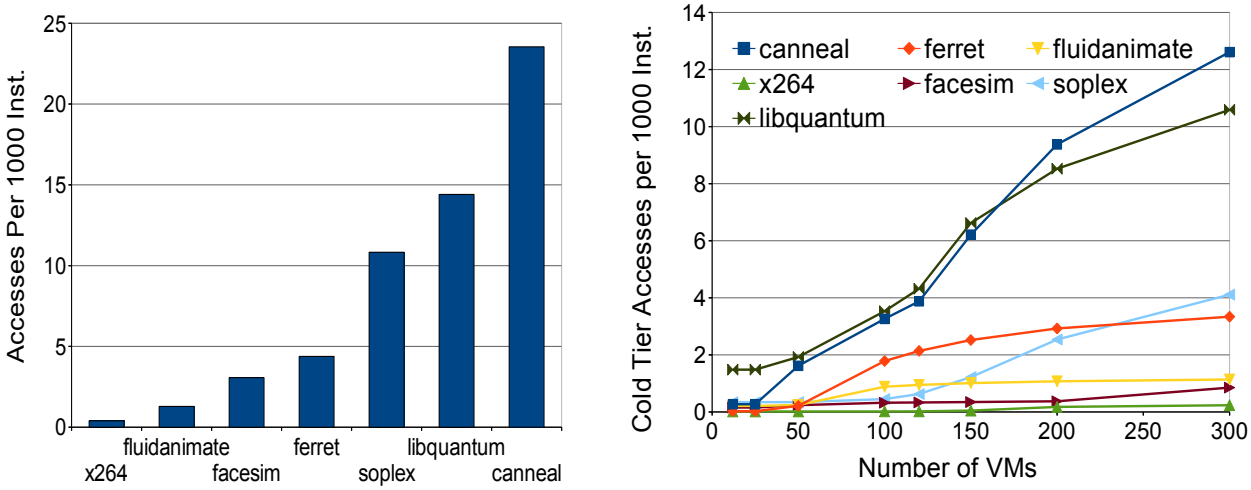


Figure 4.7: Peak Aggregate Performance for 2X- and 3X-capacity Memory Configurations.

and 3X-capacity configurations relative to the baseline design. The aggregate performance of `x264`, `facesim`, and `fluidanimate` scale almost linearly with memory capacity, achieving over 186% speedup on the 2X-capacity configuration over 275% on the 3X-capacity configuration. `soplex` and `ferret` scale fairly well, achieving over 180% and 250% on the 2X- and 3X-capacity configurations, respectively. Finally, `canneal` and `libquantum` both scale well on the 2X-capacity configuration, but experience fairly small benefits when going to the 3X-configuration.

To understand why individual applications benefit differently from the tiered organizations, we investigated how they access memory. Figure 4.8 presents the memory access characteristics of all seven benchmark programs. Figure 4.8(a) presents the total number of memory accesses per 1000 instructions, which indicates how memory-bound each application is. Figure 4.8(b) plots the number of cold tier accesses per 1000 instructions for the 3X-capacity configuration as we vary the number of VMs. These results confirm our theory of why the various benchmarks perform as they do on the various iso-powered tiered memory configurations. The two applications whose performance scales least well, `canneal` and `libquantum`, access memory more often in general and the cold tier in particular far more frequently than the other applications. In contrast, `fluidanimate`, `facesim`, and `x264` access the cold tier very rarely even in the 3X-capacity configuration, and thus experience



a. Memory Ops per 1000 instructions

b. Cold Tier Ops per 1000 instructions

Figure 4.8: Memory Behavior of Benchmark Applications

near linear performance increase. In general, the benefit of using very large iso-power memory configurations is dependent on the spatial locality of a given application.

In addition to slower accesses to the cold tier, another potential overhead in our tiered memory architecture is the cost of migrating of pages between tiers after each epoch. Short epochs can respond to application phase changes more rapidly, and thus reduce the rate of cold tier accesses, but the overhead of frequent migrations may overwhelm the benefit. To better understand the tradeoffs, we reran our experiments with four different epoch lengths: (i) 20M CPU cycles, (ii) 40M CPU cycles, (iii) 50M CPU cycles, and (iv) 100M CPU cycles — recall that our baseline evaluation described earlier used a 40M-cycle epoch. We found that while there was minor variation in performance for individual applications (e.g., `soplex`, `canneal`, and `facesim` performed best with 20M-cycle epochs, while `ferret`, `fluidanimate`, and `x264` performed best with 100M-cycle epochs), the performance variance for all applications across all epoch lengths was less than 5%. Thus, we can conclude that for reasonable choices of epoch length, performance is insensitive to epoch length and there is no need to adjust it dynamically.

For our final sensitivity study, we evaluated the impact of limiting the maximum number of migrations per epoch. Our baseline results, reported earlier, limited the number of migrations per VM to 500 per epoch, which corresponds to a maximum of 1% runtime overhead for migration given the available memory bandwidth. Figure 4.9 presents normalized performance for four applications on the 3X-capacity configuration for two other migration limits: (i) 100 migrations per VM per epoch (20% of the original limit) and (ii)

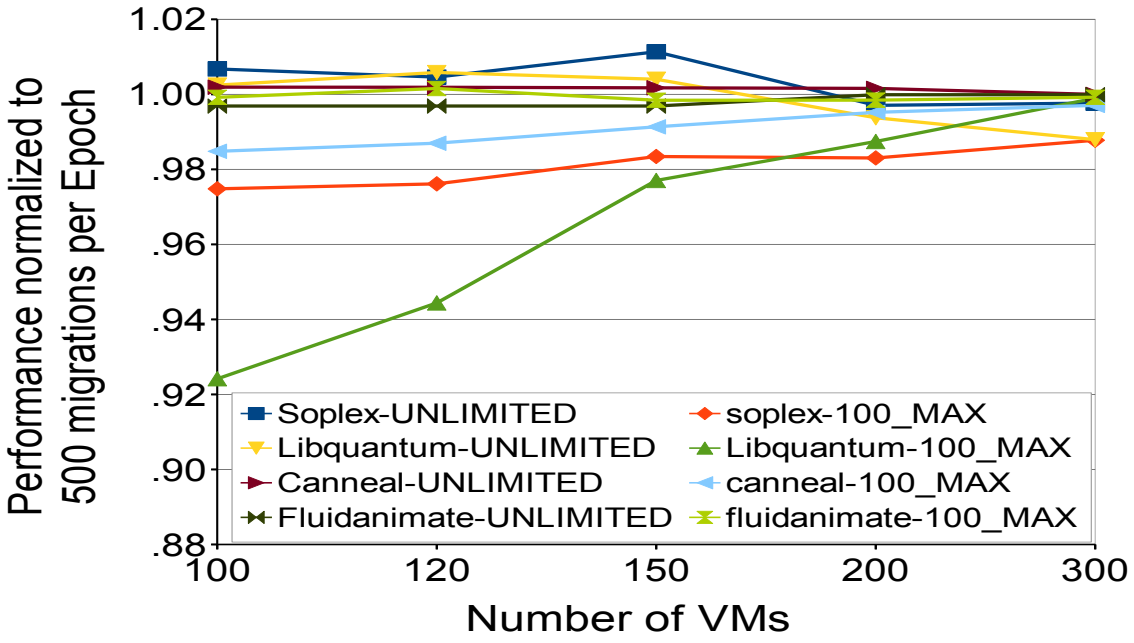


Figure 4.9: Impact of Limiting Number of Migrations per Epoch.

UNLIMITED migrations. In general, performance is insensitive to any limits on migrations, with the exception of `libquantum` when migrations were constrained to only 100 per VM per epoch. Consequently, we can conclude that limiting migrations per epoch to limit overhead has little impact on performance, and it is better to allow many migrations than to try to artificially limit them.

4.6 Discussion

This section elaborates on some of the important experimental and implementation aspects of tiered memory.

4.6.1 Contention for Shared Resources Among VMs

The iso-powered tiered memory architecture does not consider or address contention among VMs for shared resources other than main memory capacity. In most server consolidation scenarios, memory capacity is the primary factor limiting the number of VMs that a particular server can host, not memory bandwidth or compute capability, and memory capacity is increasingly limited by power. We evaluated the memory bandwidth needs of our workloads, indirectly illustrated in Figure 4.8(a). In all cases, the memory bandwidth needs of these applications is significantly below the per-core bandwidth available on mid- to high-end servers, which are designed to run very memory-intensive workloads. When we

modeled multiple VMs running in parallel sharing a single memory channel, the performance impact of contention for the channel (and associated memory controller) was negligible.

When memory bandwidth is a bottleneck, e.g., when the VMs being consolidated are running memory-intensive applications, the performance impact of increased memory bandwidth contention and iso-powered tiered memory latency overheads will tend to occur at the same time. As can be seen in Figures 4.7 and 4.8(a), the workloads with the highest memory bandwidth requirements, `libquantum` and `canneal`, experience the least benefit from tiering, at least for the 3X-capacity configuration. These kinds of memory-intensive applications are the same ones that will suffer (or cause) the most memory bandwidth contention problems in current consolidation scenarios, so a smart consolidation platform will less aggressively consolidate these workloads. A memory bandwidth-aware VM placement mechanism will naturally do a good job of iso-powered tiered memory VM placement.

Another factor that we do not address is limited compute capacity. Our models assume that if there is enough memory to hold N VMs, then there is sufficient compute capacity to execute them. In our experience, this is almost always the case for modern servers, at least for commercial applications that tend to have large memory footprints. As discussed earlier, per-processor core counts are growing much faster than memory capacity is, so memory capacity (power) is going to become an even more important factor limiting performance in the future. For situations where single-threaded performance is more important than aggregate performance, an iso-power tiered memory system may want to be less aggressive about using (slow) cold ranks.

4.6.2 Iso-power Validation

Iso-powered tiered memory organizations are guaranteed to fit under the same maximum power cap as the baseline design. This power cap is determined by the physical power delivery constraint to the system. The specific power consumed by each application on each configuration will vary, but will never exceed the baseline memory power budget, regardless of memory access pattern and migration frequency. The baseline power cap is a server-specific value that depends on the total DRAM capacity, which determines the idle-state power consumption of DRAM, and the maximum rate of memory requests, which determines the additional active power component.

Our iso-powered tiered architecture does not change the maximum bandwidth available, so the active power component of a particular tiered organization cannot exceed that of a worst-case memory-intensive application running on the baseline configuration, regardless of hot-cold access patterns or migration frequency. Migration bandwidth competes with

demand request bandwidth (from different cores), and thus cannot cause the total bandwidth consumed by a channel to exceed its limit. We should note that even for the most migration-intensive combination of application and tiered memory configuration, migration bandwidth was only 200MB/S, which is roughly 1% of available channel bandwidth (19.2GB/s). Thus, both the power and performance impact of migration is very small.

The static power component of tiered memory power is determined by the configuration. We choose iso-power configurations using the analytic model derived in Section 4.4.1.1. The power equivalence of a particular tiered configuration and the baseline design derives from the ratio of hot:cold idle power (P_{ratio}) and the hit rate of access to the hot tier (μ). P_{ratio} is a constant for a given system, while μ can be measured and the impact of the rare situation of a poor μ on power controlled via request throttling. The configurations identified by our analytic model are conservative, but a real implementation of iso-powered tiered memory would likely include hardware-firmware mechanisms to enforce the power cap despite unexpected changes in DRAM device efficiency, memory access behavior, and ambient temperature.

4.6.3 Supporting Additional Memory

The goal of our iso-powered tiered memory architecture is to increase memory capacity within a fixed power constraint. DRAM ranks can be added to an existing on-board DRAM channel with FB-DIMMs. Memory can also be added in the form of memory extensions (e.g., IBM eX5) and dedicated memory extension enclosures (e.g., IBM MAX5). One could also consider enabling more channels, but this would require modifying the processor design to incorporate more memory controllers and allocate additional I/O pins. All such approaches could incorporate variants of our proposed tiered memory design.

An alternative way to increase memory capacity at a fixed power budget is to use LR-DIMMS (load-reduced DRAM DIMMS) in place of traditional DDR3 DIMMs [70]. We consider this to be an appealing approach, but it is complimentary to our proposal.

4.7 Related Work

Although there were some notions of memory tiering in very early computing systems, the modern development of multilevel main memory appears to start with the work of Ekman and Stenström [40, 41]. They propose a two-tier scheme similar to ours and make the same observation that only a small fraction of data is frequently accessed. They manage the hot and cold tiers by using the TLB to mark cold pages as inaccessible — when a cold tier is accessed, a page fault occurs which causes the OS to copy the accessed data to the hot

tier. In contrast, our scheme never experiences a page fault for data in the cold tier. Instead, we periodically reorganize data based on access frequency to bring critical data into the hot tier. Their work focuses on reducing the data transfer overhead of moving data from the slow tier to the faster tier during a page fault. Our approach requires (modest) additional hardware support to track page access frequency and manage the tiers, but in return is able to provide faster access to the cold tier and a more gradual drop in performance when application footprints exceed the hot tier size. Also, Ekman and Stenström do not attempt to maintain an iso-power guarantee with their memory implementation.

Liu et al. [81] propose mechanisms to reduce idle DRAM energy consumption. They employ refresh rates lower than recommended by the manufacturer, which can cause data loss. They propose techniques to identify the critical data of an application by introducing new programming language constructs along with OS and runtime support. Liu et al. claim that applications can cope with data loss for noncritical data. This approach to reducing memory power is orthogonal to ours, even though both entail introducing heterogeneity into the DRAM system. We employ heterogeneity to provide a power-performance trade-off, while they trade-off power and data integrity.

Jang et al. [60] propose methods to reduce main memory energy consumption using VM scheduling algorithms in a consolidation environment. They propose to intelligently schedule VMs that access a given set of DRAM ranks to minimize the number of ranks in Active-Standby mode. We believe this approach is orthogonal to our proposal and can be combined with our scheme to provide increased aggregate memory power savings.

Lebeck et al. [75] propose dynamically powering down infrequently used parts of main memory. They detect clusters of infrequently used pages and map them to powered-down regions of DRAM. Our proposal provides a similar mechanism while guaranteeing we stay under a fixed memory power budget.

Ahn et al. [10] proposed an energy-efficient memory module, called a Multicore DIMM, built by grouping DRAM devices into multiple "virtual memory devices" with fine-grained control to reduce active DRAM power at the cost of some serialization overhead. This approach is orthogonal to our proposal.

Huang et al. proposed Power-Aware Virtual Memory [54] to reduce memory power using purely software techniques. They control the power-state of DRAM devices directly from software. Their scheme reconfigures page allocation dynamically to yield additional energy savings. They subsequently proposed extensions that incorporate hardware-software cooperation [55, 56] to further improve energy efficiency. Their mechanisms reduce the

power consumption of a given memory configuration, whereas we use low-power DRAM modes to enable greater memory capacity in a fixed power budget.

Waldspurger [116] introduced several techniques — page sharing, reclaiming idle memory, and dynamic reallocation — to stretch the amount of memory available to enable more virtual machines to be run on a given system. The Difference Engine approach by Gupta et al. [48] and Satori by Murray et al. [88] tackles the same problem of managing VM memory size. These proposals are complimentary to our approach and leveraging them would lead to even higher aggregate numbers of VMs per server.

Ye et al. [123] present a methodology for analyzing the performance of a single virtual machine using hybrid memory. Using a custom virtualization platform, they study the performance impact of slower, second-level memory on applications. Their motivation is to leverage technologies like flash storage to reduce disk accesses. In contrast, our scheme aims to reduce the aggregate impact of tiering by managing data at a more fine-grained level while strictly conforming to a fixed power budget.

Heterogeneous DRAM-based memory designs were recently studied by Dong et al. [39] to improve total memory capacity. They propose using system-in-package and 3D stacking to build a heterogeneous main memory. They also move data between memory tiers using hardware- or OS-assisted migration. The heterogeneity in their design is along the dimensions of bandwidth and capacity. They focus on application performance and attempt to provide the lowest access latency and highest bandwidth to frequently accessed data. We also strive to provide the lowest access latency to frequency accessed data, but our work is different than their approach since we focus on power budget and leverage existing low-power DRAM modes rather than emerging packaging technologies.

Flash-based hybrid memory schemes have also been researched extensively lately. Badam and Pai [15] developed a scheme to use flash memory in a flat address space along with DRAM while introducing minimal modification to user applications. Their scheme allows building systems with large memory capacity while being agnostic to where in the system architecture the flash capacity resides (PCIe bus, SATA interface, etc.). The performance characteristics of flash are quite different than that of DRAM placed in a low-power mode, so the architecture and performance tradeoffs of the two approaches are quite different. Our work could be combined with a flash-based hybrid memory scheme to create a memory architecture with an additional tier.

Finally, researchers recently proposed replacing traditional DRAM (DDR3) with low-power DRAM chips (LP-DDR2) designed for mobile and embedded environments [70]. LP-

DDR2 devices consume less power than server DDR3 DRAM, but they have lower peak bandwidth and their design severely limits the number of devices that can be supported on a single DRAM channel, both of which run counter to the need to increase memory capacity and bandwidth. Iso-powered tiered memory is nonetheless equally applicable to LP-DDR2 main memories also and may even be more effective with LP-DDR2 since LP-DIMMs support even deeper low-power states than DDR3.

4.8 Summary

The goal of the tiered memory scheme is to increase a server's memory capacity for a fixed memory power budget. The design presented here builds a two-tier iso-powered memory architecture that exploits DRAM power modes to support 2X or 3X as much main memory for a constant power budget. In the design, the main memory is divided into a relatively smaller hot tier, with DRAM placed in the standby/active state, and a larger cold tier, with DRAM placed in the self-refresh state. Data placement is aggressively used to ensure that most of the memory accesses are serviced from the hot tier. Given the relative idle power of DRAM in active and self-refresh modes, the design is also able to provide a range of iso-power configurations that trade off performance (in the form of "hot tier" size) for total capacity. In a server consolidation environment, supporting 3X as much memory, even when much of it must be maintained in a slow low-power state, allows a single server to achieve 2-2.5X the aggregate performance of a server with a traditional memory architecture.

CHAPTER 5

INTELLIGENT NETWORK OF MEMORIES

In this chapter, we describe a future memory system built using 3D stacked DRAM devices. One of the major objectives of this design is to realize a memory system with larger capacity using point-to-point link technology for the memory channel. In this context, the schemes described in this chapter discuss how data placement can be leveraged to reduce memory access delay and power consumption for such a system.

5.1 Introduction

The contribution of memory system power has been rising in the past few years because of the following reasons: (1) Processor cores have become simpler and more energy-efficient, while the microarchitecture of memory chips has seen little change; (2) Processor clock frequencies and processor power have been relatively constant, while memory channel frequencies have been steadily increasing to boost pin bandwidth and feed data to a growing number of cores; (3) Buffer chips [58], that consume nearly 10 W each, are typically required to provide high memory capacities.

A high-end server, such as the IBM p795 [57] or the HP DL580 G7 [1], provides memory capacities in the tera-byte range. In such systems, the limited pins on a processor are required to provide high-bandwidth access to this large memory capacity. Since DDRx DRAM chips are high-density commodities, they cannot incorporate circuits for high-speed signaling. They therefore communicate via 72-bit DDR3 channels at modest frequencies. The DDR3 standard channel also accommodates only two DIMMs because of signal integrity concerns. With a DDR3 interface, a processor would only be able to support four channels, eight DIMMs, and a pin frequency of at most 933 MHz. Solutions such as LR-DIMM [2] can provide a one-time boost to memory capacity.

However, a substantial boost to memory capacity and bandwidth requires the processor to move away from the DDR3 interface. Processors can implement high-speed signaling

and many narrow memory channels (links). This increases the bandwidth and memory capacity per pin. The processor uses these many high-speed serial links to communicate with buffer chips on the motherboard (or DIMM). The buffer chips, in turn, use the DDR3 interface to communicate with DRAM chips via wide and slow buses. An example of such a motherboard architecture for the high-end Intel Nehalem platform is shown in Figure 5.1. The Nehalem design uses Scalable Memory Buffer (SMB) chips that are rated at 10 W [58], much higher than the power consumption of most DIMMs (nearly 5 W for DDR3-1333 R-DIMMs). The SMB consumes a large amount of power because it has to convert between the DDR3 and SMI protocols, and because of its high-speed SerDes circuits. The high power of a similar AMB chip in the FB-DIMM architecture [90] contributed to its relative nonadoption. In fact, primarily because of power constraints, the Nehalem-EX approach has lower scalability than FB-DIMM. The AMBs of the FB-DIMM could be daisy-chained to boost capacity (up to eight DIMMs on a daisy-chained link), while the SMBs cannot be daisy-chained and only allow up to four DIMMs per processor link.

The emergence of 3D stacking may finally lead to a viable and power-efficient approach to memory capacity scaling. Of the many recent 3D memory products [95, 112, 42], Micron's Hybrid Memory Cube (HMC) [95] is a compelling design point. We will use it as a representative example throughout this chapter. The HMC prototype design stacks four DRAM chips on a logic chip. The logic chip incorporates the circuits for high-speed serial links that can connect to the processor or other HMCs. The HMC package therefore provides all the functionality of a traditional DIMM and its buffer chip. The HMC's logic layer is much more power efficient than a buffer chip because there is no need for an off-chip DDR3 interface and no need for conversion between DDR3 and (say) SMI protocols. The logic layer receives requests on the high-speed links and accesses the DRAM banks via low-power TSVs. Because of the logic layer's relative power-efficiency, Micron envisions that multiple HMCs will be connected to grow memory capacity. However, a wide open problem before the research community is how should the HMCs be connected and managed to build an efficient network of memories? To our knowledge, this is the first body of public-domain work that attempts to address a portion of the above problem.

Specifically, we first focus on estimating an optimal network topology. We explore the design space and show that a tapered-tree topology offers the best performance because of relatively low hop counts and low link contention. Most topologies offer nonuniform hop counts to different HMCs; to reduce the average hop count, we propose and evaluate a number of page placement strategies that retain popular pages at nearby HMCs. With

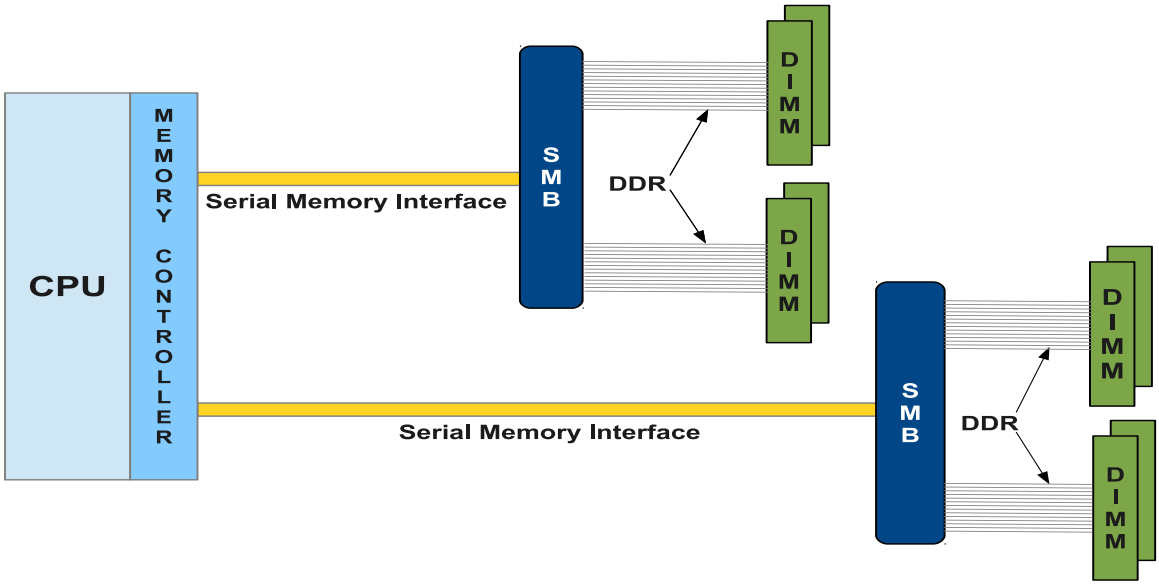


Figure 5.1: The Intel Scalable Memory Interface and Buffer

intelligent page placement, link contention is low enough that router bypassing techniques can be very effective. This alleviates the overheads of introducing routers in the HMCs with relatively complex functionality. The intelligent page placement also increases idle times in many of the HMCs, thus allowing frequent use of deep-sleep modes in these HMCs. The proposed iNoM architecture yields a 49.3% performance improvement and a 42% energy reduction, relative to a daisy-chained memory network with random page placement.

5.2 Intelligent Network of Memories (iNoM)

This section considers a number of design issues and innovations when constructing a network of memories. We first describe many candidate topologies and their trade-offs. We then argue for page placement mechanisms that can impact hop counts, memory queuing delays, and traffic patterns on the network. Because of the resulting traffic patterns, significant energy savings are possible by employing router bypassing and aggressive low-power memory modes.

5.2.1 Channel Topology

There are two primary constraints that need to be kept in mind for a network-of-memories: *(1)* memory node (HMC) pin count, and *(2)* traffic pattern. We explore the impact of these constraints on topology choices. First, to keep the packaging cost of the memory node low, we fix the I/O pin count of each node. Dally [35] and Agarwal [9] show that for fixed packaging constraints, lower radix networks offer lower packet latency due to

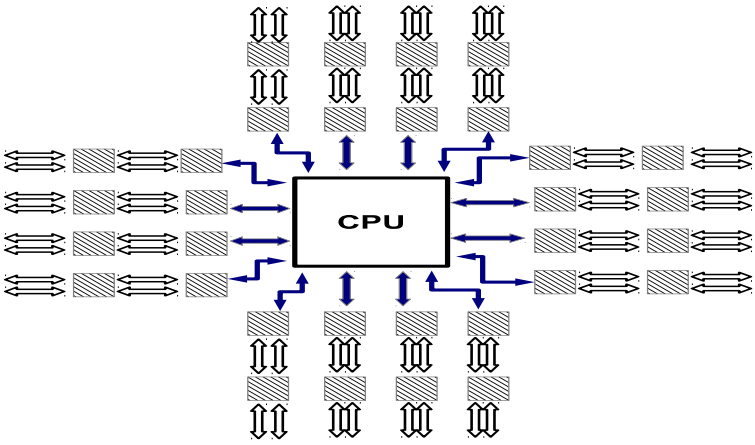
lower router complexity and hence fewer router delays. We therefore focus on low radix topologies [34] for the iNoM. Next, since these topologies will be primarily used for two-way traffic between the CPU and the memory, we only study topologies that are suitable for such a traffic pattern. Therefore, topologies like hypercube [34] that are aimed at low-latency communication between any two network nodes are not attractive for an iNoM. Hierarchical topologies like the fat-tree [77], on the other hand, are especially well suited for this traffic pattern.

Following these two design constraints, we examine the following five topologies: daisy chain, mesh, binary tree, fat tree, and tapered tree. These topologies are shown in detail in Figures 5.2 and 5.3. In all these topologies, the blue colored boxes represent the minimum repeatable unit that must be added when extending the memory system. In the case of daisy-chain and mesh topologies, the minimum repeatable unit is simply the memory node (one HMC) itself; hence, there are no blue boxes in Figure 5.2(a) and 5.2(b).

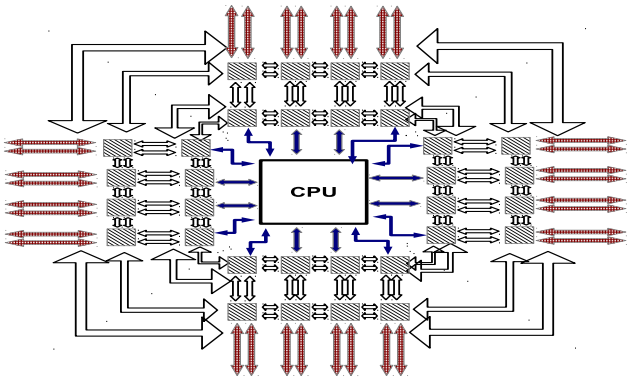
The mesh topology is a simple extensible network, but like the daisy chain topology, it can suffer from congestion based on the routing algorithm and the traffic pattern. The binary tree (BTree) is the simplest hierarchical topology; we show later that hierarchical topologies are well suited for some memory traffic patterns. A two-level fat tree (FTree) topology is another hierarchical topology that delivers high throughput even for random traffic patterns. This, however, requires a large number of links, all of which may not be used effectively. The tapered tree (TTree) topology reduces this cost by reducing the link count in the upper levels of the hierarchy.

For all the evaluations presented in this chapter, cacheline striping policy that maps consecutive cache lines from the same OS page across all the memory channels is used. This striping is also compatible with the choice of a close-page policy. Each topology can be thought of as consisting of "rings" (or "sets") defined as HMC nodes that are equidistant in hop count from the CPU. An OS page, while being allocated, has its cache lines spread over all the HMCs in one ring only. Since the topologies like daisy chain, mesh, and FTree are symmetrical, they have an equal number of HMCs on all the rings. BTree and TTree are asymmetrical and are organized among rings, as described in Table 5.1.

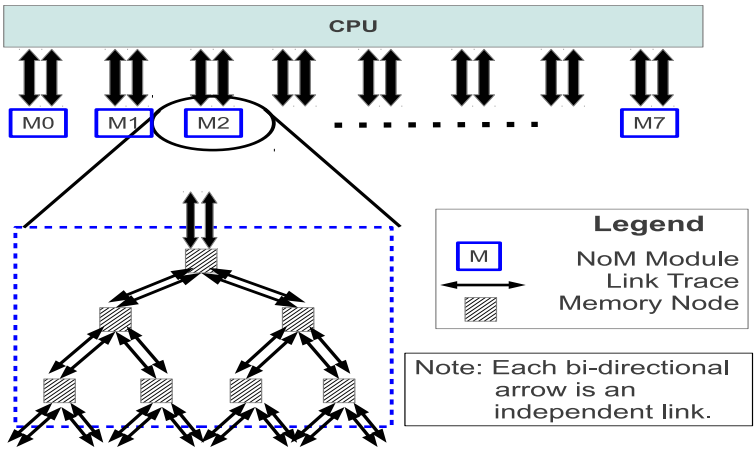
With an upper limit of 64 HMCs imposed by a memory power budget of 675 W (details in Section 5.3.1), the daisy-chain and mesh topologies are constructed using 64 HMCs, the BTree using 56, the FTree using 48, and the TTree using 44 HMCs. The organization of these HMCs on the rings is dictated by the topology and the number of HMCs that need to be directly connected to the CPU. Daisy-chain and mesh each have 4 rings, BTree and



(a) Daisy Chain

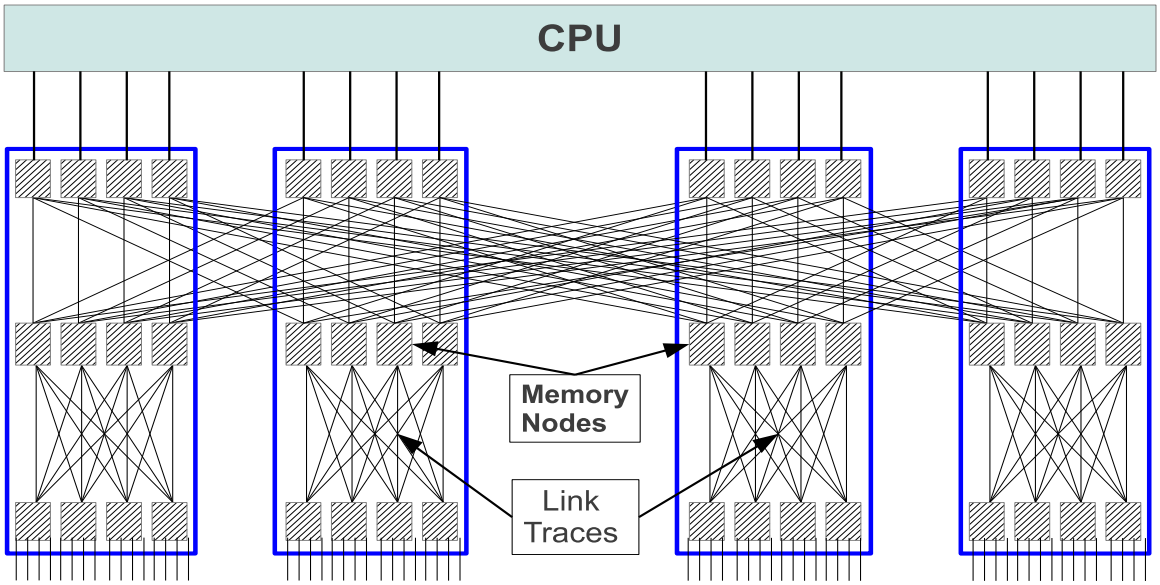


(b) Mesh

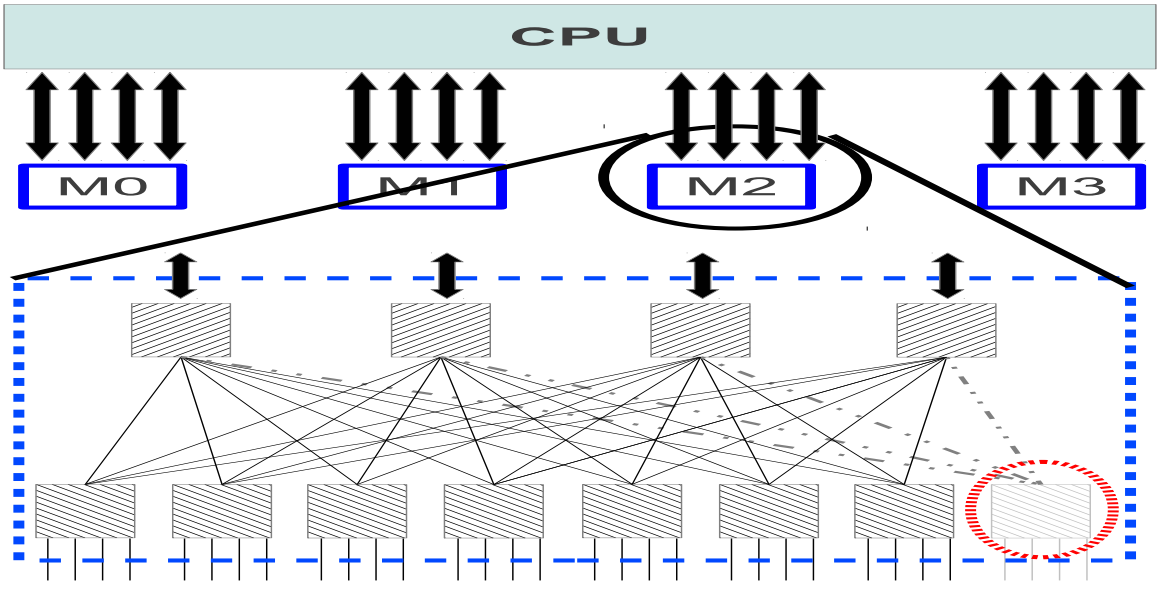


(c) Binary Tree

Figure 5.2: Daisy Chain, Mesh, and Binary Tree Topologies



(a) FTree



(b) TTree

Figure 5.3: Fat Tree, and Tapered Tree Topologies

Table 5.1: Topology Configurations

Topology	HMC Devices	Organization
Daisy Chain	64	4 rings; 16 HMC per ring
Mesh	64	4 rings; 16 HMCs per ring
Binary Tree	56	3 rings; 8,16,32 HMCs per ring
Fat Tree	48	3 rings; 16 HMCs per ring
Tapered Tree	44	2 rings; 16,28 HMCs per ring

FTree have 3, and TTree has 2 rings.

To maximize the benefits offered by the topologies with more links, we assume that the router implements fully adaptive routing with nonminimal routes. It handles deadlock based on a deadlock recovery approach (described next), which eliminates the need for virtual channels [34]. The routing algorithm works in two different modes. In the normal mode, any packet at the routing stage within a router selects the output port based on the congestion level at the downstream routers, while selecting output links on the minimal paths with a higher probability. The congestion level is measured based on the approach suggested by Gratz et al. [47].

To avoid deadlocks, each packet is permitted to reroute for a limited times and has to follow the minimal path when the allowed number of reroutes reaches zero. The number of possible reroutes per request is assigned based on the distance between the source and the destination, to control the power and latency overhead of rerouting. In the case when a request waits for more than 1000 cycles, it broadcasts a deadlock alarm forcing all the routers to be in the deadlock mode. In the deadlock mode, all routers first move all the requests from the router buffers to special deadlock prevention (DP) buffers and then route them using a deadlock free routing scheme such as dimension order routing in the mesh topology. Once all the requests in the DP buffers have been sent, the routers transition back to normal mode operation. The required DP buffers can easily be realized by using some of the DRAM capacity of the HMC. Virtual channel-based deadlock prevention schemes can also be used, but we experiment only with simple schemes like the one described above.

5.2.2 OS Support for Data Allocation

The networks of memories described in the previous subsection yield nonuniform latencies for memory accesses. For example, HMCs in ring-1 can be reached quicker than HMCs in ring-2. The average memory access time can be reduced by placing frequently accessed pages in rings closer to the processor. This also reduces the average hop count

and traffic on the network. The problem is similar to that of data placement in NUCA caches [68, 50, 14, 64] and we develop solutions that extend the ideas in prior work [68, 107]. However, there are some key differences: *(1)* We are dealing with data at page granularity, *(2)* There is no need for a search mechanism to locate data because the page tables and TLBs track page migrations, and *(3)* DRAM queuing delays impact our choice of policy thresholds. We observe that moving most popular data to the first ring can lead to long queuing delays when accessing the banks of the HMCs. It is therefore sometimes better to place a page in the next ring and incur the latency for the extra hop than be stuck in a long DRAM Access Queue. In the outer ring, the OS-page is striped among more HMCs, thereby decreasing the contention at any given HMC. As the HMC or the network topology options are varied, we experiment with different policy thresholds and observe that the optimal thresholds vary substantially. This is an important insight that is peculiar to a nonuniform memory system because of the long DRAM bank access time, and has not been reported in prior work.

We develop and evaluate four different page placement mechanisms, in addition to the baseline *random* policy in use today: *dynamic*, *percolate-up*, *percolate-down*, *profiled*.

The *dynamic* scheme allocates new pages randomly while monitoring memory accesses at run-time. It then periodically migrates heavily used OS pages to HMCs that are closer to the CPU. The periodic time interval is a tunable parameter called *Epoch*. The scheme identifies heavily accessed OS pages using the methods proposed in [108, 122, 107]. These schemes augment the CPU memory controller and use the Multi-Queue algorithm [131] to rank OS pages according to their access frequency. This mechanism is described in detail shortly. Based on this information, the *dynamic* scheme migrates the most heavily accessed N pages closer to the CPU while moving the N infrequently accessed pages away from the CPU. This is achieved using an OS daemon that is activated every Epoch. As described above, the optimal value of N varies with our platform assumptions.

The *percolate-down* policy allocates new pages accessed by the application to the HMCs that are closer to the CPU. It also periodically pushes out N OS pages from the HMCs closest to the CPU, once the access rate to these pages falls below a tunable threshold. As a result, this policy requires fewer data movements during run-time, compared to the *dynamic* scheme where a minor reshuffle of page assignment is performed every epoch. In Section 5.3.2, we quantify these fewer data movements and provide the sensitivity of our results to the tunable thresholds. The *percolate-up* policy is similar to the *percolate-down* policy, but instead of allocating new pages closer to the CPU, it allocates new pages to

HMCs farthest from the CPU. Over the course of execution, heavily accessed OS pages are "promoted" to HMCs closer to the CPU, while infrequently touched pages are demoted. The pages to be promoted are identified by the Multi Queue (MQ) tracking algorithm described shortly. The *percolate-up/down* policies can be thought of as insertion policies for caches, where *percolate-down* is equivalent to placing a new cache line in the Most Recently Used (MRU) position, and *percolate-up* amounts to placing the new cache line in the Least Recently Used (LRU) position.

Finally, the *profiled* scheme leverages the information of an application's memory access pattern from a prior run to allocate virtual to physical page translations. The scheme allocates $N\%$ of the most frequently accessed pages to memory nodes closest to the CPU. This policy, while impractical, is designed to evaluate the maximum potential for improvement with a static data placement scheme.

The MQ algorithm to identify hot pages employs LRU queues of page descriptors, each of which includes a page number, reference count, and an expiration value. The descriptors in a given queue represents the pages that are most frequently accessed. Our implementation of this mechanism maintains 16 hierarchical queues with 256 entries each, i.e., 4096 total entries corresponding to a total 16 MB of DRAM working set being tracked at any given time. The decision to track 16 MB of DRAM working set was taken to minimize the overhead of tracking too many pages while still capturing a reasonable working set during an epoch. This was determined by performing a design space study to measure the typical working set size during an epoch. Every access to a page is recorded by incrementing the reference count for that page, and updates the expiration value which is used to identify when the page can be demoted to a lower queue. Conversely, a page is promoted to the next higher queue when its reference count reaches a set threshold – 8192 accesses in our evaluations. To avoid performance degradation, updates to these queues are performed by the memory controller off the critical path of memory accesses, using a separate queue of updates and a small on-chip SRAM cache. When pages are selected for promotion/migration to HMCs in ring-0, pages not resident in ring-0 that feature in any of these hierarchical queues are candidates for promotion to ring-0. Conversely, all pages not in the queues are candidates for demotion to outer rings.

In the evaluations, a page migration is modeled to occupy the link for 12 CPU cycles because a page is striped across at least 8 HMCs and it takes 1 CPU cycle to read one cache lines per HMC. The application execution is delayed until all the page migrations for an epoch are complete. This introduces overheads but note that a page migration is relatively

fast because a page is striped across all HMCs in a ring; i.e., many high-bandwidth links are engaged for every page transfer. Furthermore, only a small number of pages are moved every epoch. The CPU-side memory controller simply issues commands to copy cache lines among a pair of source and destination HMCs. This does not add any new complexity to the controller and is similar to the OS copying data from the kernel memory space to the user memory space during DMA operations in current systems. Note that the HMC logic layer can be used to track page activities and to buffer pages as they are slowly migrated.

5.2.3 Supporting Optimizations

With the proposed topologies and data mapping policy, we observe that much of the network traffic is localized to the first ring. Therefore, contention on links beyond the first ring is relatively low. We also observe that the topologies with high radix (fat tree and tapered tree) do engage adaptive routing to yield high performance. However, a steep energy price is being paid to implement a high-radix router. We therefore believe that a customized topology can be constructed to selectively use additional links in parts of the network. We leave this for future work. In this study, we employ another simple technique to alleviate the power overhead of a lightly-loaded high-radix router. Prior work [119, 34] has proposed the use of buffer and router bypassing under low load. After the packets have been decoded by the SerDes circuits, we can examine the destination addresses of incoming packets to determine if an output port collision will result. If there is no known impending collision, the packet moves straight to the SerDes block at the output port, without consuming power in the remaining router stages. We also considered SerDes encode/decode bypassing if we could somehow propagate the destination address with the encoded packet. However, most SerDes power is within the analog circuits for the output drivers [106], so the power saving from bypassing the encode/decode would have been small.

The second important side-effect of our proposed topology and page placement is that DRAM banks and links in distant rings experience long idle times. To reduce the power in distant HMCs, the entire device can be placed in deep-sleep low-power modes. Power-down functionality is crucial to any large memory system since large capacity could mean wasting power if the extra capacity is not used by the applications. For the iNoM proposal, we consider two low-power modes of operation for each HMC. The logic layer implements the logic for transitioning the HMC into low-power states based on the memory traffic each HMC experiences. Note that this model is different from the current designs where the CPU-side memory controller is responsible for putting DIMMs in low-power states. The first low-power mode transitions the 3D DRAM layers to the Precharge Power Down

Mode [85] along with most of the router functionality, except for the bypassing logic. We call this mode PD-0.

The next power-down mode, PD-1, transitions the SerDes link circuitry on the HMC logic layer into a low-power state [106, 110]. The difference between these two power-down modes is that an HMC in PD-0 can still propagate requests to its connected HMCs, although it cannot reroute packets based on network load. In the PD-1 mode, the HMC cannot forward or handle any request without first powering up. A beacon SerDes lane is kept active for the HMCs in PD-1 mode to wake up the rest of the circuitry when the transmitter on the other end of the link is ready to send data. This is a well-known technique used for low-power SerDes designs and is also implemented in commercial PCIe links [110]. We conservatively assume that the wake up time from this PD-1 state to be twice the wake up time from the L1 state of the PCIe. An HMC moves from power-up state to the PD-0 state after I idle cycles and to PD-1 state after J idle cycles. Because of the longer idle times in our proposed design, HMCs remain powered down for long stretches, while minimally impacting performance.

5.3 Methodology and Results

5.3.1 Methodology

To evaluate the iNoM design, we use a combination of trace-based detailed cycle-accurate simulation and long non-cycle-accurate simulation using Pin [82]. The latter is only employed to verify that our page access statistics are consistent even when we run our programs to completion and experience memory capacity pressures. Unless otherwise specified, all the results shown are with the trace-based cycle-accurate simulator. This simulator processes multithreaded traces generated by Pin and models an in-order, 128-core chip-multiprocessor (CMP) system with parameters listed in Table 5.2(a).

We experiment with eight applications from the NAS Parallel Benchmark (NPB) suite [16]. For these applications, the memory footprint can be controlled by varying the input set class. We chose the input class for each application such that we were able to instrument the applications using Pin and that resulted in the largest memory footprint. Since the complete memory trace for an application could be a several terabyte large file, we imposed an upper limit of 1 billion memory accesses on the trace length for the cycle accurate simulations. The simulated workloads, their input class, memory footprint, and LLC misses per kilo instructions (MPKI) for each application are listed in Table 5.2(b). The high MPKIs are consistent with those reported by others [125]; such workloads are a good stress test of

Table 5.2: Simulation parameters

Parameter	Value
CPU Core	In-order, 2GHz
CMP size	128-core
L1 I-cache	128KB/2-way, private, 1-cycle
L1 D-cache	128KB/2-way, private, 1-cycle
L2 Cache	4 MB, 8-way, private, 20-cycle
L3 Cache	256 MB, 8-way, shared, 30-cycle
Cache line size	128 Bytes
Memory System Pins	1024

(a) CPU Parameters

Application	Footprint	MPKI
CG.D	17 GB	359.1
BT.E	169 GB	262.3
SP.E	203 GB	238.5
MG.E	211 GB	227.5
IS.D	36 GB	198.9
UA.D	10 GB	187.6
FT.D	82 GB	158.0

(b) Workloads

Parameter	Value
DRAM Capacity	512 MB
Num. Links*	8
Link Bandwidth	40 GB/s
Usable Bandwidth*	256 GB/s
I/O Pins*	512
Power Consumption	11 W
Energy per Bit	10.8 pJ/bit

(c) HMC Parameters
(* = Projected Parameters)

Parameter	Value
Frequency	2 GHz
Num_Links	8
I/O Buffer Depth	8
DRAM Q Depth	8
Response Q Depth	8

(d) Router Parameters

memory bandwidth and capacity for future big-data workloads.

This work targets the design of high memory capacity server systems, for example, the IBM p670 server [76]. Given the 675 W per socket memory power budget in the IBM p670 [76], and the 11 W rating of one HMC, we assume that our memory system can accommodate 44-64 HMCs based on topology, as described in Table 5.1.

HMC device parameters [102, 61] are listed in Table 5.2(c). These parameters match the 2nd-generation HMC numbers disclosed by Sandhu [102]. Instead of assuming 4 links per HMC, as in the 2nd-generation device, 8 links per HMC are modeled. This higher number of links is on the HMC roadmap [102] – the 5th-generation HMC has 8 links per HMC. This choice allows us to experiment with a richer set of topologies.

The current HMC prototype consumes 10.8 pJ/bit of energy. Out of this 10.8 pJ, 6.78 pJ/bit is used by the logic layer, and 3.7 pJ/bit by DRAM layers [61]. The HMC power breakdowns [102] indicate that the prototype design may not include a sophisticated router. We therefore increment the energy-per-bit value for the HMC with the energy-per-bit (3.91 pJ/bit) for a generic 4x4 router pipeline to include functionality like the I/O buffers, routing logic, and the bypass logic. We arrive at this 3.91 pJ/bit number by considering the router implemented by Howard et al. [52] for the Intel 48-core SCC chip, and scaling it for bandwidth using the estimates provided for the active and static power consumption. This router energy is scaled up or down based on the radix for the topology being modeled, i.e., a daisy chain uses a 3x3 router, and a tapered tree uses a 9x9 router – these scaling factors are estimated by modeling routers with different radix in Orion-2.0 [66]. The manufacturing technology for the HMC logic layer is taken as the 45nm process, similar to that for the Intel SCC.

In both the PD-0 and PD-1 states, the DRAM is assumed to be powered down. When powered down, a 1Gb DRAM die takes 18 mW power per DRAM die [85]. In these power-down states, each router consumes 0.1 W [53]. In the PD-1 state, the SerDes links are powered down. In this state, the SerDes links consume 10 micro Watts per lane [110]. We conservatively assume that the wake up time from this PD-1 state to be twice the wake up time from the L1 state of the PCIe [13].

The proposed router on the logic layer of the HMC is modeled in detail with finite length I/O buffers. Each I/O link on the router has eight request deep FIFO buffers associated with it. The router delay is modeled as described in [96] and the router parameters for our evaluations are listed in Table 5.2(d). The DRAM Access Queue dequeues four requests per router cycle – one for each 3D DRAM layer. We model 128 banks across these four

DRAM layers and assume each bank operates independently [95]. A closed-page row buffer policy is modeled for each of these banks. Once the DRAM core accesses are complete, the requests are injected into the Completed Request Queue. The router logic can schedule up to eight completed requests from this queue (one request per output port). It preferentially schedules requests from the completed queue before scheduling requests from the input buffers, unless the requests in the input buffers have been waiting in the buffer for longer than 10 router cycles.

The data allocation policies that migrate OS pages have also been modeled in detail. The optimal thresholds for migration and the number of migrated pages vary across topologies and are experimentally determined. This is because the DRAM queuing delays can rise if too many pages are moved to the first ring. The latency and energy overheads for page migration have also been modeled, assuming 400 ns for each 4 KB OS page that has been migrated.

5.3.2 Results

5.3.2.1 Execution Time

To understand the impact of topology and data allocation policy, we first present results showing their impact on application execution time. Topologies with multiple paths to the CPU (like the mesh, FTree etc.), are expected to perform better compared to topologies with fewer paths like the daisy-chain and the BTree. Similarly, allocation policies with adaptive data placement (*Dynamic, P-*, etc.*) are expected to outperform static allocation schemes. The results for these experiments are plotted in Figure 5.4. The execution times are normalized against the daisy-chain topology with *Random* allocation scheme. The TTree topology with the *Percolate-down* allocation outperforms all other practical allocation schemes with an average improvement of 49.3% compared to the daisy-chain topology with *Random* allocation. This improvement is within 4.2% of the *Profiled* allocation policy for the same topology. With *P-down* allocation, the TTree topology outperforms the daisy chain by 40.1%, the mesh by 30.1%, the BTree by 57%, and the FTree by 17.8%. In the following sections, we delve deeper to understand the cause of these patterns.

5.3.2.2 Memory Access Delay

In this section, we analyze the memory delay incurred in servicing a single memory request. Figure 5.5(a) plots the average memory access delay instead of reporting individual applications for clarity. The average memory delay follows similar trends as application execution time in Figure 5.4. This is to be expected since we model an in-order CPU core.

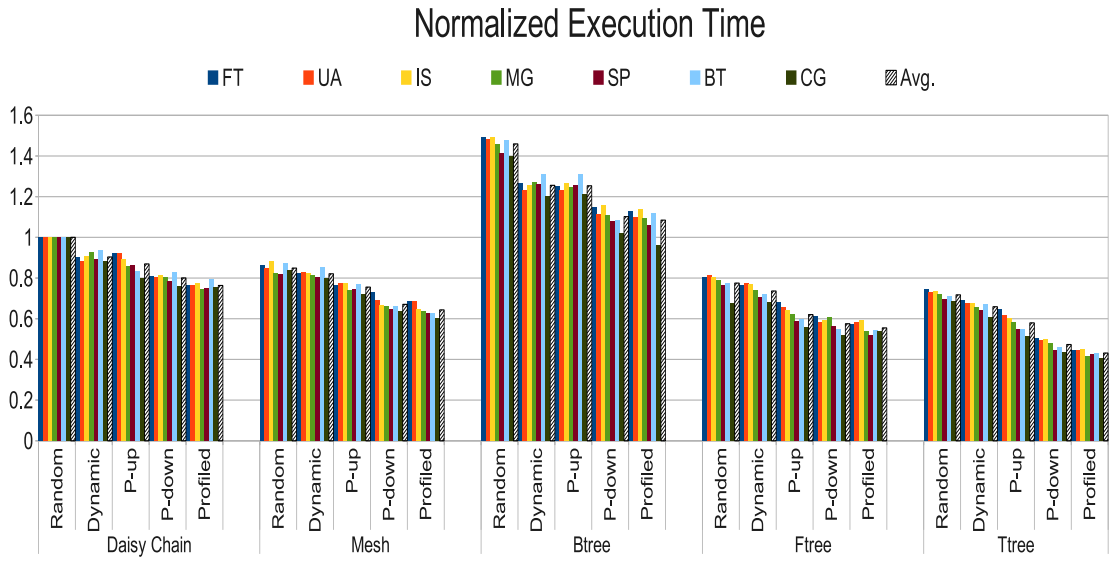
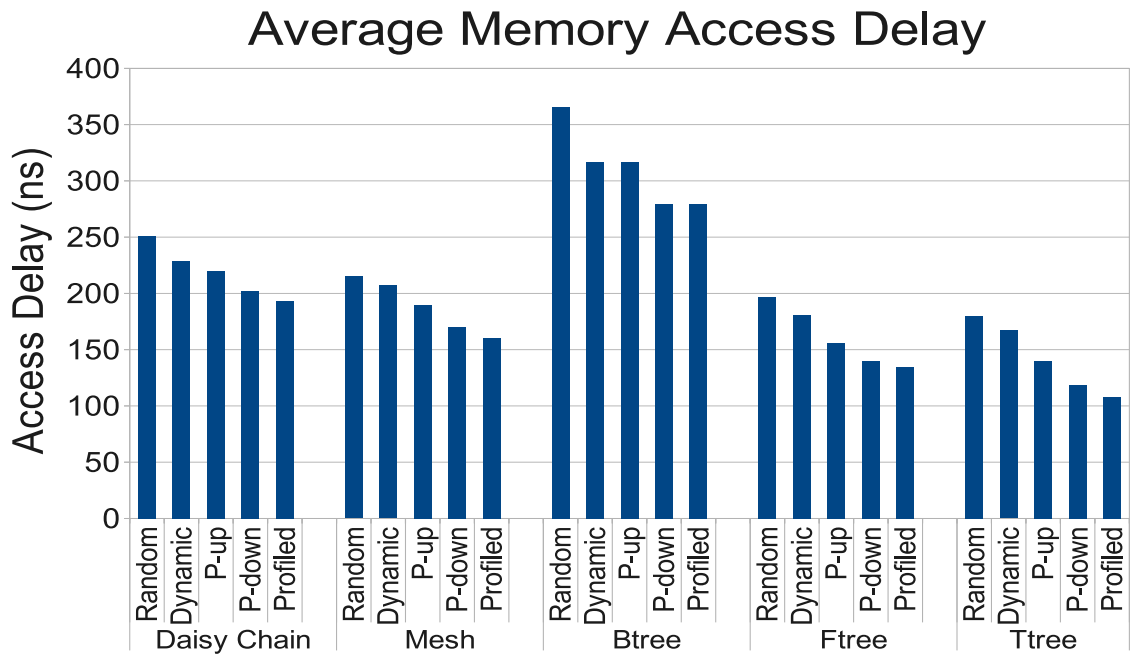


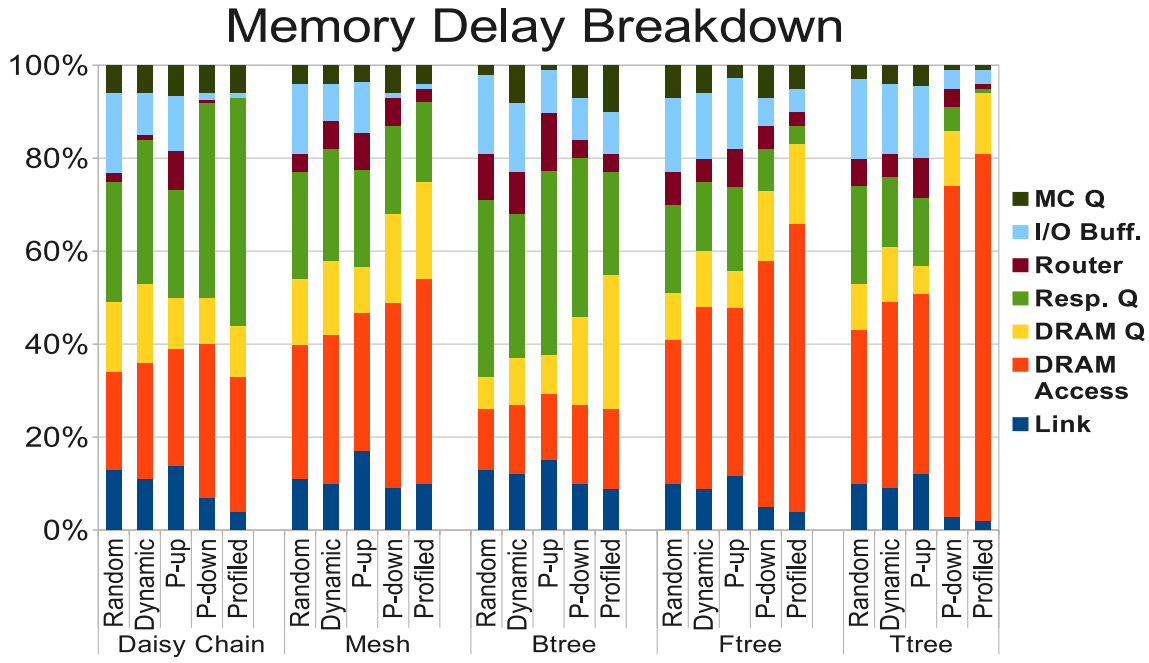
Figure 5.4: Application Execution Time Normalized w.r.t. Daisy Chain Topology with *Random* Allocation Policy

Also note that the access delay is sensitive to the thresholds set for the allocation policies. The rate at which data are migrated for the adaptive policies like *Dynamic, P-** therefore varies for each topology. Figure 5.5(a) plots the result for each topology with the best threshold settings for that topology.

To understand how the topology and data placement policy impact the access delay, Figure 5.5(b) breaks down the access delay into the fraction of time spent among the various components of the system. Each access is broken down into the time spent in the CPU-side controller queue (MC Q in the figure), link delay, HMC input buffer delay (I/O Buff in the figure), router delay, the DRAM access queue delay, the completed request queue delay (Resp. Q in the figure), and the actual DRAM core access time. Ideally, the fraction of time spent accessing the DRAM should be 100% as time spent in all other components is essentially an overhead to access the DRAM core. As can be seen from the figure, moving left-to-right increases the sophistication of the topology and data placement policy. This directly correlates with an increasing fraction of time being spent in DRAM core access, i.e., reduction in overhead to reach the memory. Router bypassing is critical in reducing the access delay, and our results indicate that for the TTree-*P-down*, the input buffers are bypassed for nearly 86.4% of the requests on average. We next explain the reason for change in fraction of time spent in each component as we move towards more sophisticated topologies and data placement.



(a)



(b)

Figure 5.5: Memory Access Delay

The MC Q time is impacted by the contention for the link directly connecting the CPU to an HMC. Multiple paths to reach a destination HMC from the CPU - as in Mesh, FTree,

and TTree topologies - helps reduce this contention. As a result, the fraction of time spent in MC Q with *Profiled* allocation reduces from 9.8% for the BTree topology (which has the most contention due to all HMCs being accessed via few CPU-HMC links), to 4% for the TTree topology which has the highest number of independent paths from the CPU to the HMCs.

The I/O buffer delays on the router are incurred due to the inability of the router to bypass the requests. Since the router gives priority to the injection port, a long buffer delay implies that the given HMC is completing requests at a fast rate. The daisy chain topology sees the most stalls for output ports. When data are not primarily allocated to Ring-0, some of these stalls show up as I/O buffer delays (responses from beyond Ring-0 that get stuck at Ring-0) and some show up as Response queue delays. When data are primarily allocated to Ring-0 (as with P-down and Profiled), the stalls mainly show up as Response queue delays because the Ring-0 HMCs are servicing requests at a fast rate. As path diversity increases with the TTree topology, both delays go down significantly.

The router delays in Figure 5.5(b) are a function of the typical hop count and the ability to bypass the router. Again, these are substantially lower for the TTree with P-down.

The DRAM queue delay increases as we move to sophisticated page mapping policies. For example, the Profiled policy tends to place many pages in Ring-0 and these HMCs tend to have long DRAM access queues. The DRAM core access time is roughly a constant across all designs (since we use a close-page policy), but its contribution grows as we move to the right because all other components shrink. Finally, the link delays are a function of hop count and they are lowest with intelligent page placement.

To further understand how different data allocation schemes impact the number of memory requests served from HMCs in a given topology, we measure the fraction of total requests served from each ring of the iNoM architecture. With intelligent allocation of data, the goal is to serve as many requests as possible from HMCs closest to the CPU. Ideally, all requests should be served from Ring-0; however, due to capacity pressures and dynamically changing placement of data with the *Dynamic* and *Percolate-** policies, not all requests can be served from ring-0. Figure 5.6 plots the request distribution among rings organized by topology. Since different rings of a topology provide varying fraction of total memory capacity, the last bar in each topology group plots the fraction of total capacity supplied by each ring. The trend in Figure 5.6 shows that with intelligent allocation, the fraction of requests served from ring-0 increases significantly. For the TTree topology, the percent of requests served from ring-0 changes from 39% with *Random*, to 88% with *Percolate-down*

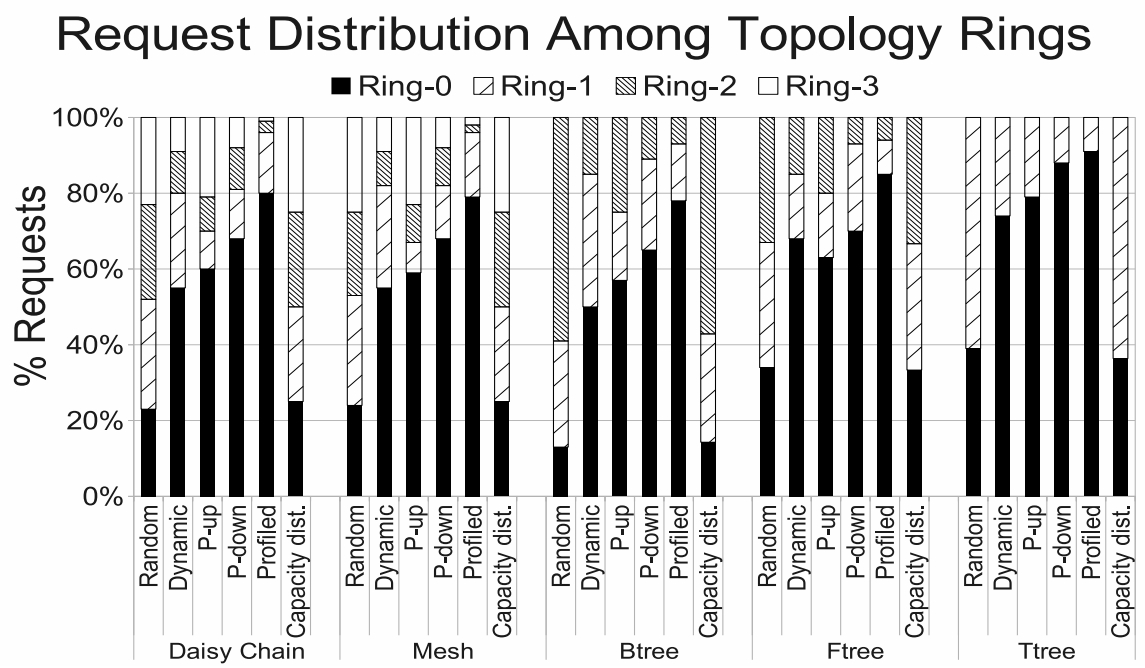


Figure 5.6: Request Distribution

policy. To make sure that the request distribution in Figure 5.6 is not skewed by limiting our simulations to only a billion DRAM accesses, we ran the NPB applications with the D input set to completion, using the non-cycle-accurate Pin-based simulator described in Section 5.3.1. Such an experiment is necessary to confirm that our policies are robust even when there are capacity pressures in each ring and the application goes through different phases. We observed a very similar request distribution to rings even with the complete application simulation (not shown for space reasons).

While the HMC offers 128 independent banks, it is not clear how many requests can be initiated simultaneously in the same cycle. This may be a function of whether DRAM circuits/interconnects are shared, or whether power delivery limits are being exceeded. In most of our evaluation, we assume that 4 requests to different banks can be simultaneously de-queued and initiated. The DRAM queuing delays for our best organizations are non-trivial. These delays grow even more if we assume that only 1 request can be initiated in a cycle. These delays are high enough that it makes sense to sometimes move data to distant rings to reduce DRAM queuing delay. This is a phenomenon not observed by prior work on NUCA caches. As an example case study, Figure 5.7 shows a model where 1 request is de-queued every cycle for the P-down policy and the daisy chain topology. The overall

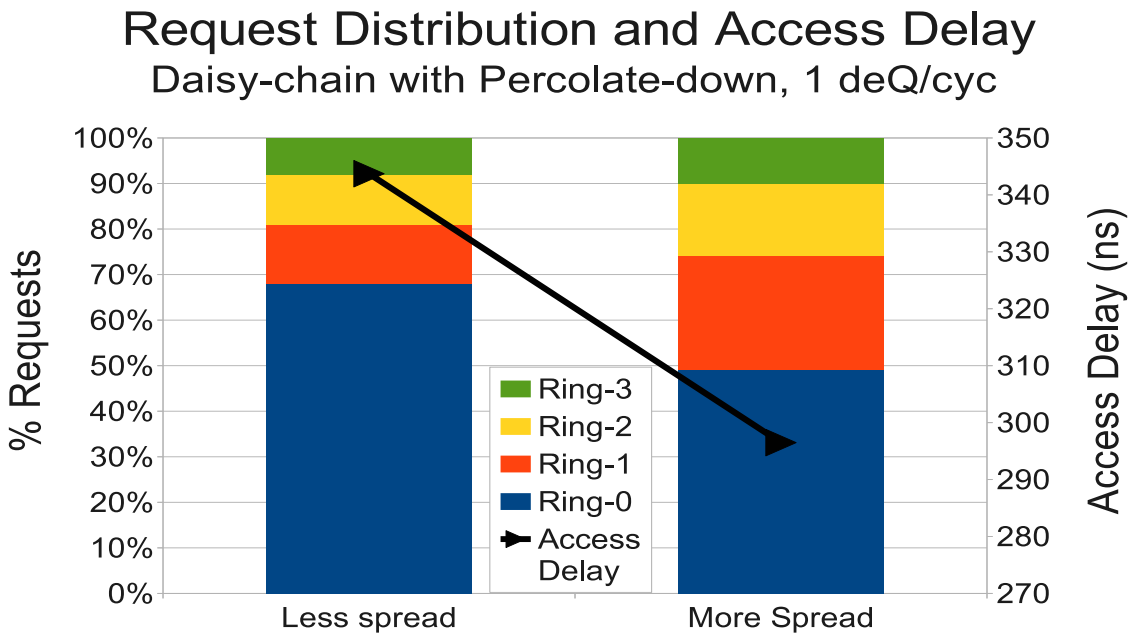


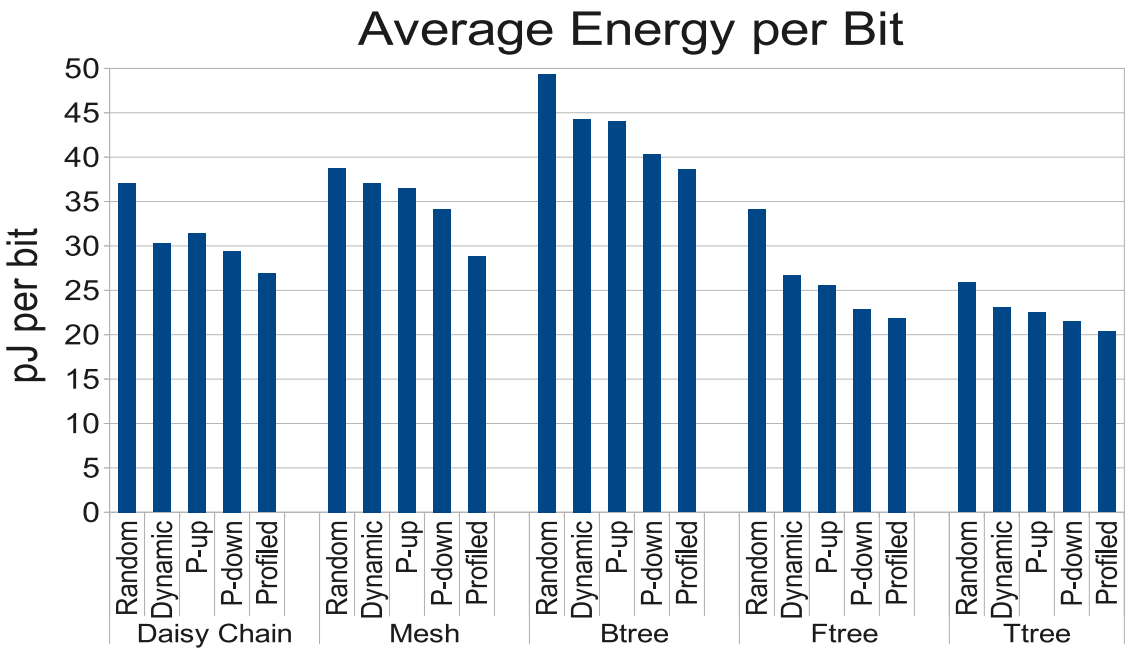
Figure 5.7: Memory Access with Relaxed Thresholds for Data Migration

memory access latency reduces (the line, see right axis) as fewer requests are placed in Ring-0 (by modifying the thresholds and parameters for the P-down policy), shown by the breakdown bar.

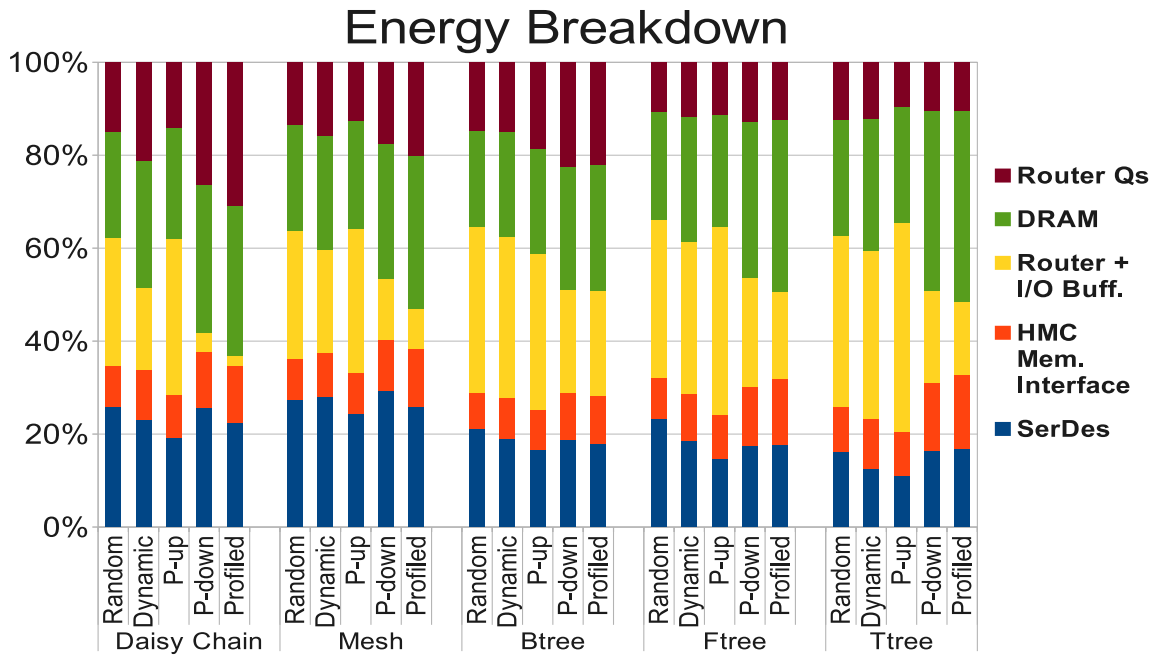
5.3.2.3 Energy Consumption

To understand the energy consumed per memory request by various topologies and allocation schemes, we measured the average energy consumed per bit, and then broke down the energy per bit into the various energy consuming components of our design. This breakdown includes energy consumed by the DRAM core access, the SerDes link energy and HMC memory interface logic, and the energy consumed by the structures introduced as part of the router logic, i.e., the router queues (the DRAM access queue and the completed request queue), I/O buffers, and finally, the routing logic itself. We note that the I/O buffer and response queue occupancies decrease with better topologies and page placements. As a result, we expect the contribution of energy consumed by these structures to decrease, similar to the trend in memory delay. These results are plotted in Figure 5.8.

Figure 5.8(a) shows the average energy per bit consumed and Figure 5.8(b) plots the breakdown. Note that if an HMC is directly connected to the CPU, it consumes 10.48 pJ/bit (assuming no energy for router components). If the HMCs are connected in a daisy chain



(a) Average Energy Consumed per Bit



(b) Energy Breakdown per Memory Request

Figure 5.8: Energy Consumption

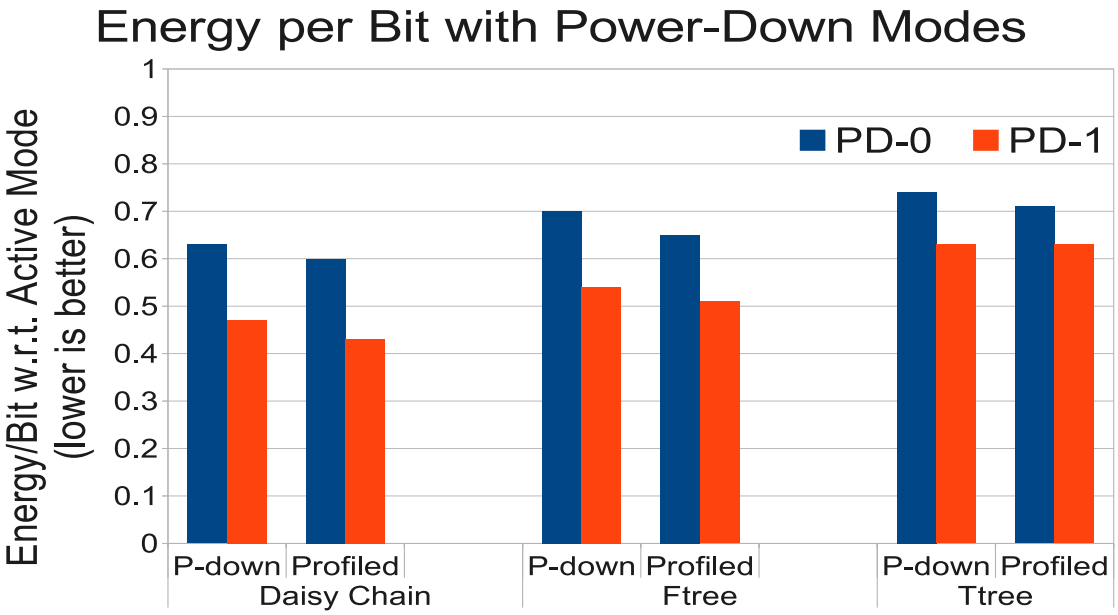
with only two rings, the energy per bit is expected to more than double (because of router components), and scale linearly as more ring are added. The overall impact of this is captured in Figure 5.8(a), with the daisy chain topology with *Random* placement consuming 37.1 pJ/bit. In comparison, the TTree topology with *Percolate-down* placement consumes 42% less energy per bit. It is important to note that the TTree topology has a more complex 9x9 router (10.15 pJ/bit), while the daisy chain topology has a simpler 3x3 router (1.95 pJ/bit). In spite of this disadvantage, the TTree topology reduces energy per bit because of the reduced number of hops. It is also worth noting that energy is saved in the processor because of the shorter execution time afforded by the TTree topology.

Figure 5.8(b) further shows that the TTree topology with *Percolate-down* allocation uses nearly 29% of its energy for the routing logic and buffers, compared to nearly 53% for the daisy chain topology with *Random* allocation. Most of this energy reduction comes from bypassing the router I/O buffers. As noted earlier, for the TTree-*P-down* mode, the I/O buffers are bypassed for nearly 86.4% of the requests on average.

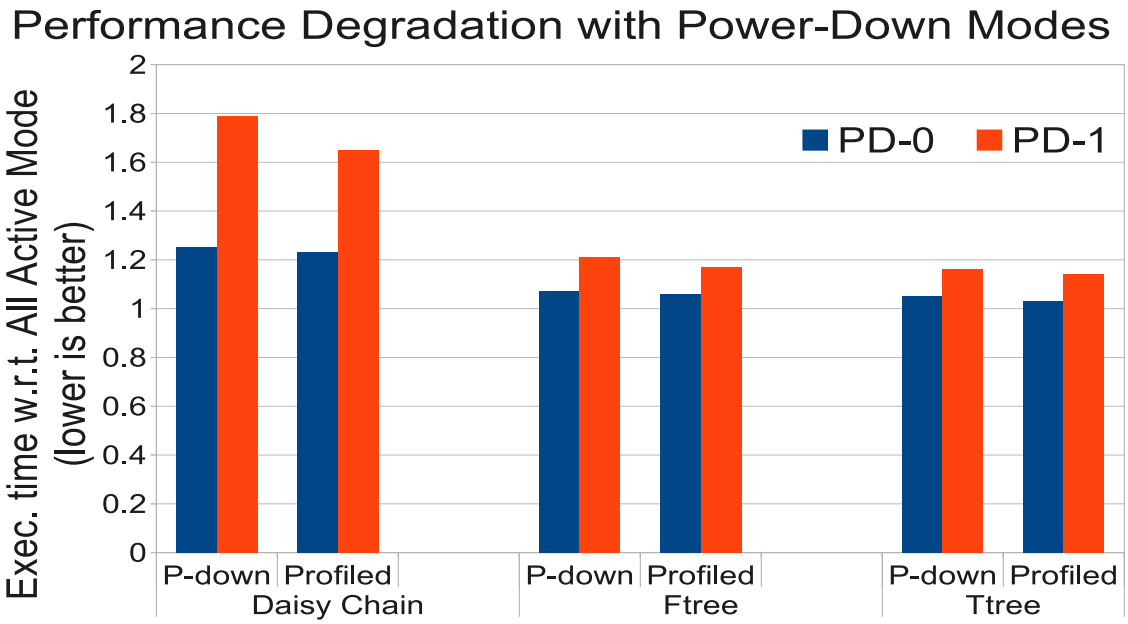
5.3.2.4 Energy Savings with Power-Down Modes

As described in Section 5.2.3, with active data management, the iNoM architecture provides an opportunity to put idle HMCs in low-power states. We consider two deep-sleep modes (PD-0 and PD-1) that progressively turn off functionality to reduce power consumption. In this section, we quantify the reduction in energy per bit, and the associated increase in average execution time due to the overheads of exit latency, to wake up an HMC from low-power mode. We consider two policies, one that never attempts PD-1, and one that first switches to PD-0 and then to PD-1. With PD-1, we expect the increase in execution time to be larger, compared to PD-0 since it takes nearly 1000X longer to exit the PD-1 mode compared to PD-0 (64 ns vs. 64 us).

Figure 5.9 shows the results of these experiments for daisy chain, FTree, and the TTree topologies. We only show the *Percolate-down* and *Profiled* topologies since these allocation schemes are best suited to providing power-down opportunities for distant HMCs. Figure 5.9(a) shows the reduction in energy-per-bit when compared to a system with no power-down mode support. The daisy chain topology yields large energy savings because it has more HMCs in total, thus yielding more idle times per HMC. The TTree topology with Percolate-down yields a 26% reduction in energy per bit, while engaging only the PD-0 mode. In terms of performance, the TTree topology shows the least slowdowns since the distant rings are accessed infrequently. The TTree Percolate-down policy yields a 5% increase in execution time for PD-0. An important point to note is that the Random page



(a)



(b)

Figure 5.9: Impact of Low-Power Modes on Energy per Bit and Execution Time

placement policy (not shown in the figure) provides almost no opportunities for power-down, yielding only a 1% reduction in energy per bit.

5.4 Related Work

There is little prior work that has considered the design of a network of memories, especially for 3D-stacked memory devices. A recent paper explores design considerations when using on-board buffer chips to expand capacity and bandwidth [124, 29]. The RAMCloud project [94] builds an aggregated memory space that is spread across many compute nodes, thus resembling a network of memories. Similarly, memory blades [78, 79] have been used to provide on-demand memory capacity for compute blades, another example where different portions of memory are accessible over a network.

The most related work is the recently proposed DIMM-Tree architecture [113], where DIMMs are connected using a combination of multidrop and point-to-point links to form a tree-like hierarchy. The siblings in the tree are connected using a multidrop bus and the different levels in the tree are connected using point-to-point links. This allows the memory access latency to grow logarithmically with the number of DIMMs rather than linearly as in FB-DIMMs. While that paper demonstrates the need and feasibility of creating a nontrivial network of DIMMs on the motherboard, our work is the first that examines different network topologies and considers the impact of data allocation in such networks in the context of future 3D memory packages.

Our proposed memory architecture involves the placement of critical pages in rings of devices at varying distances from the processor. This is reminiscent of earlier schemes that dictate data placement for NUCA caches [11, 18, 19, 68, 50, 14, 64]. However, as noted in Section 5.2.2, there are many differences from prior NUCA work. In some cases, data must be spread to minimize DRAM queuing delays. Also, there is a much stronger interaction between data placement policies and network/DRAM optimizations, such as router bypassing and DRAM power-down modes.

5.5 Summary

Future high-capacity memory systems are likely to be built using point-to-point serial links instead of parallel buses due to pin count and signal integrity limitations. In this chapter, we show that it is better to invest in more complex topologies (such as tapered tree) with more complex routers to build network-of-memories for high-capacity memory systems. Combined with smart page placement policies and router bypassing, the design yields better performance and better energy per bit by reducing hop count, buffering, and queuing delays.

It also leads to more power-down opportunities. The design space for memory networks is large and we believe that several future innovations are possible, including the design of custom topologies that selectively add links to target network hotspots. With the techniques described in this chapter, a network-of-memory design utilizing the tapered tree topology and intelligent data placement outperforms the daisy chain topology by 49.3% in application execution time over a suite of benchmarks. This design also can reduce energy per-bit by 42% when low-power modes are leveraged along with intelligent data placement.

CHAPTER 6

CONCLUSIONS AND FUTURE WORK

This chapter summarizes the techniques presented in this dissertation and describes future extensions of data management concepts for emerging memory systems.

6.1 Conclusions

In this dissertation, we propose mechanisms that use intelligent data placement to improve DRAM main memory access. The mechanisms range from their ability to improve access latency and efficiency at the same time (micro-pages), reducing power/energy consumption and consequently increasing memory capacity in a fixed power budget (tiered memory), and finally, improving access latency and power efficiency for future memory systems built as a network-of-memories (iNoM). The common theme among all the schemes proposed in this thesis is the use of data placement techniques to minimize DRAM access inefficiencies.

Data management has traditionally been abstracted away at the operating system level. This dissertation demonstrates that with increasingly difficult challenges faced by the memory systems, data management can be used as a powerful tool to optimize future memory systems. The broad impact of data placement in improving memory access characteristics is a critical observation this dissertation makes. Another equally significant observation made by this dissertation is the fact that data management schemes require little to no hardware changes to implement. This is relevant when considering that the DRAM memory industry is extremely cost sensitive and even a minor change to DRAM device architecture faces significant challenges for adoption.

To summarize, we now list the major highlights of each described data management scheme to improve memory access.

- In Chapter 3, we showed data management schemes that first identify frequently accessed segments of an OS page. Using this information, these chunks are coalesced to reside in the same DRAM row-buffer by changing the data allocation on the

fly. This results in increased row-buffer hit rates which have been diminishing due to independent memory access streams generated by multicore CPUs. Increased row-buffer hits reduce memory access latency, while simultaneously reducing memory energy consumption. Our data management schemes show an average 9% increase in performance, while simultaneously reducing memory energy consumption by 15% on average.

- In Chapter 4, we describe a mechanism to leverage DRAM low-power modes aggressively by consolidating actively used data in a few DRAM ranks. This enabled transitioning ranks with low activity to the self-refresh state reducing their standby power consumption. We also design a simple mechanism to batch memory requests destined for the low-power ranks to prevent frequent transitions to/from low-power states. This mechanism guarantees acceptable DRAM access latency increases to limit the adverse impact of batching on applications. With these techniques, we were able to increase DRAM memory capacity by 2-3X in the same power budget. This allowed us to run as many as 3X the number of virtual machines compared to the baseline, while achieving an aggregate 250% improvement in performance.

- In Chapter 5, a possible future memory system design is investigated that uses point-to-point serial links for the memory channel along with 3D integrated memory devices. We first investigate the logic required to implement a network-of-memories using point-to-point interconnects, and then study different topologies suitable for memory systems. We then investigate in detail the data management policies to improve memory access. Our results show that data management can play a significant role for such a network-of-memories. We study various data placement techniques to find that access latency and energy consumption of these memory systems can be significantly improved by intelligent allocation of data. Our experiments show that with data management and a suitable network topology, the application performance can be improved by an average 49% accompanied by a 42% reduction in energy, relative to a simple daisy-chained topology with random data placement.

6.2 Future Work

Main memory design is a rapidly evolving area within computing systems. Consequently, this work can be extended along several lines based on the specifics of the requirements from the main memory system. In this section, we discuss possible extensions to this work

organized by three categories of performance and power trade-offs likely to dominate future memory systems.

6.2.1 High-Performance Memory Systems

These memory systems are typically characterized by their demands on the memory system for low access latency combined with high bandwidth and capacity. These requirements are combined with "power-proportionality" [43], which means that the memory system consumes little idle power and power consumption increases linearly with increasing utilization. Such memory systems are most likely to be used in high-end servers targeting memory intensive workloads, or application consolidation scenarios.

Memory systems targeted for this design space can benefit from 3D DRAM stacking to increase total capacity. Though 3D stacking can help with issues like energy-per-bit, it is however not a panacea. The primary concern for such memory systems would be reliability since these memory systems are likely to use a large number of memory devices. Intelligent data allocation can improve reliability of such systems by actively managing replicated, and/or versioned copies of data. This, however, will add to the storage overhead in already capacity constrained systems and will lead to coherence/versioning problems. Tackling these issues with low-overhead schemes can be an interesting line of investigation. Techniques from the storage domain like de-duplication of data can be leveraged here with modifications.

A number of high-end systems are also emerging with special purpose accelerators aimed at a small subset of applications executing on these systems. Managing the memory interaction of these accelerators with the system CPU is also an interesting problem where data allocation can help.

6.2.2 Low-Power Memory Systems

This memory system design point is best exemplified by designs predominant in handheld computing systems like smart-phones, tablets, embedded systems, etc. The requirements from these designs are extremely efficient memory accesses to conserve energy, and ability to transition from deep power saving states quickly.

For such systems, controlling the rate of memory requests to stay within power and thermal constraints is likely to be a dominating design parameter. It would be necessary to aggressively manage DRAM accesses either by throttling memory requests at the CPU level, or by controlling the rate of requests served from the DRAM devices. Another concern would be managing the bandwidth among multiple applications/subsystems. For example,

many low-power devices (like smart-phones) rely on system-on-chip designs that combine CPU as well as GPU cores on the same die. Since these SoC typically connect to the memory using a shared interface, memory bandwidth partitioning would be necessary to ensure memory requests from the CPU and GPU do not interfere excessively. Devising data management schemes to automatically allocate data in DRAM devices to reduce destructive interference is an interesting and challenging problem for this domain.

6.2.3 Balanced Memory Systems

This class of memory system designs represent memory systems likely to be found in desktop computing environments. These designs are expected to provide performance between that of the high-performance and the low-power memory systems, while being much lower cost and standard compatible, similar to the commodity parts that dominate the personal computing landscape today.

Since such systems are expected to be extremely cost sensitive, a majority of innovations for these designs are expected to be centered around software support to efficiently manage the memory system. Besides the techniques described in this dissertation, the software support has to be expanded to include new challenges similar to those seen in low-power, highly-integrated systems. Since desktop systems are also emerging with more and more functionality being integrated on to the CPU (exemplified by integration of subsystems like graphics, Wi-Fi, etc.), maintaining quality-of-service for each application is a crucial problem to solve here. Demand-based partitioning of the memory address space based on intelligent allocation can help here. It is interesting to note here that this demand based partitioning has to occur on the fly since users can move from executing a graphics intensive application to using a spreadsheet tool, each of which have different memory access characteristics (bandwidth sensitive vs. latency sensitive).

REFERENCES

[1] HP ProLiant DL580 G7 Server Technology. http://goo.gl/aOQ3L.

[2] Load-Reduced DIMMs. http://www.micron.com/products/modules/lrdimm/index.

[3] STREAM - Sustainable Memory Bandwidth in High Performance Computers. http://www.cs.virginia.edu/stream/.

[4] Wind River Simics Full System Simulator. http://www.windriver.com/products/simics/.

[5] Intel 845G/845GL/845GV Chipset Datasheet: Intel 82845G/82845GL/82845GV Graphics and Memory Controller Hub (GMCH), 2002.

[6] Java Server Benchmark, 2005.
Available at http://www.spec.org/jbb2005/.

[7] Micron DDR3 SDRAM Part MT41J512M4. http://download.micron.com/pdf/datasheets/dram/ddr3/2Gb_DDR3_SDRAM.pdf, 2006.

[8] Hybrid Memory Cube Consortium, 2011. http://www.hybridmemorycube.org/.

[9] A. Agarwal. Limits on Interconnection Network Performance. *IEEE Transactions on Parallel and Distributed Systems*, 1991.

[10] J. Ahn, J. Leverich, R. S. Schreiber, and N. Jouppi. Multicore DIMM: an Energy Efficient Memory Module with Independently Controlled DRAMs. *IEEE Computer Architecture Letters*, 7(1), 2008.

[11] S. Akioka, F. Li, M. Kandemir, and M. J. Irwin. Ring Prediction for Non-Uniform Cache Architectures (Poster). In *Proceedings of PACT*, 2007.

[12] K. Albayraktaroglu, A. Jaleel, X. Wu, M. Franklin, B. Jacob, C.-W. Tseng, and D. Yeung. BioBench: A Benchmark Suite of Bioinformatics Applications. In *Proceedings of ISPASS*, 2005.

[13] ALTERA Inc. PCI Express High Performance Reference Design. www.altera.com/literature/an/an456.pdf.

[14] M. Awasthi, K. Sudan, R. Balasubramonian, and J. Carter. Dynamic Hardware-Assisted Software-Controlled Page Placement to Manage Capacity Allocation and Sharing within Large Caches. In *Proceedings of HPCA*, 2009.

[15] A. Badem and V. Pai. SSDAlloc: Hybrid SSD/RAM Memory Management Made Easy. In *Proceedings of NSDI*, 2011.

[16] D. Bailey et al. The NAS Parallel Benchmarks. *International Journal of Supercomputer Applications*, 5(3):63–73, 1994.

[17] L. Barroso and U. Holzle. *The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines*. Morgan & Claypool, 2009.

[18] B. Beckmann, M. Marty, and D. Wood. ASR: Adaptive Selective Replication for CMP Caches. In *Proceedings of MICRO*, 2006.

[19] B. Beckmann and D. Wood. Managing Wire Delay in Large Chip-Multiprocessor Caches. In *Proceedings of MICRO-37*, December 2004.

[20] D. Berger, J. Chen, F. Ferraiolo, J. Magee, and G. V. Huben. High-speed Source-synchronous Interface for the IBM System z9 Processor. *IBM Journal of Research and Development*, 2007.

[21] B. Bershad, B. Chen, D. Lee, and T. Romer. Avoiding Conflict Misses Dynamically in Large Direct-Mapped Caches. In *Proceedings of ASPLOS*, 1994.

[22] C. Bienia et al. The PARSEC Benchmark Suite: Characterization and Architectural Implications. Technical report, Princeton University, 2008.

[23] P. J. Bohrer, J. L. Peterson, E. N. Elnozahy, R. Rajamony, A. Gheith, R. L. Rockhold, C. Lefurgy, H. Shafi, T. Nakra, R. O. Simpson, E. Speight, K. Sudeep, E. V. Hensbergen, and L. Zhang. Mambo: a full system simulator for the PowerPC architecture. *SIGMETRICS Performance Evaluation Review*, 2004.

[24] J. Carter, W. Hsieh, L. Stroller, 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. In *Proceedings of HPCA*, 1999.

[25] J. B. Carter and K. Rajamani. Designing energy-efficient servers and data centers. In *IEEE Computer*, July 2010.

[26] R. Chandra, S. Devine, B. Verghese, A. Gupta, and M. Rosenblum. Scheduling and Page Migration for Multiprocessor Compute Servers. In *Proceedings of ASPLOS*, 1994.

[27] M. Chaudhuri. PageNUCA: Selected Policies For Page-Grain Locality Management In Large Shared Chip-Multiprocessor Caches. In *Proceedings of HPCA*, 2009.

[28] S. Cho and L. Jin. Managing Distributed, Shared L2 Caches through OS-Level Page Allocation. In *Proceedings of MICRO*, 2006.

[29] E. Cooper-Balis, P. Rosenfeld, and B. Jacob. Buffer On Board Memory Systems. 2012.

[30] J. Corbalan, X. Martorell, and J. Labarta. Page Migration with Dynamic Space-Sharing Scheduling Policies: The case of SGI 02000. *International Journal of Parallel Programming*, 32(4), 2004.

[31] R. Crisp. Direct Rambus Technology: The New Main Memory Standard. In *Proceedings of MICRO*, 1997.

[32] V. Cuppu and B. Jacob. Concurrency, Latency, or System Overhead: Which Has the Largest Impact on Uniprocessor DRAM-System Performance. In *Proceedings of ISCA*, 2001.

[33] V. Cuppu, B. Jacob, B. Davis, and T. Mudge. A Performance Comparison of Contemporary DRAM Architectures. In *Proceedings of ISCA*, 1999.

[34] W. Dally and B. Towles. *Principles and Practices of Interconnection Networks*. Morgan Kaufmann, 1st edition, 2003.

[35] W. J. Dally. Performance Analysis of k-ary n-cube Interconnection Networks. *IEEE Transactions on Computers*, 1988.

[36] V. Delaluz et al. DRAM Energy Management Using Software and Hardware Directed Power Mode Control. In *Proceedings of HPCA*, 2001.

[37] X. Ding, D. S. Nikopoulosi, S. Jiang, and X. Zhang. MESA: Reducing Cache Conflicts by Integrating Static and Run-Time Methods. In *Proceedings of ISPASS*, 2006.

[38] X. Dong, N. Muralimanohar, N. Jouppi, R. Kaufmann, and Y. Xie. Leveraging 3D PCRAM Technologies to Reduce Checkpoint Overhead in Future Exascale Systems. In *Proceedings of SC*, 2009.

[39] X. Dong, X. Wu, G. Sun, Y. Xie, H. Li, and Y. Chen. Circuit and microarchitecture evaluation of 3D stacking magnetic RAM (MRAM) as a universal memory replacement. In *Proceedings of DAC*, 2008.

[40] M. Ekman and P. Stenström. A case for multi-level main memory. In *WMPI*, 2004.

[41] M. Ekman and P. Stenstrom. A Robust Main-Memory Compression Scheme. In *Proceedings of ISCA*, 2005.

[42] Elpida Memory, Inc. News Release: Elpida, PTI, and UMC Partner on 3D IC Integration Development for Advanced Technologies Including 28nm, 2011. http://www.elpida.com/en/news/2011/05-30.html.

[43] X. Fan, W. Weber, and L. Barroso. Power Provisioning for a Warehouse-sized Computer. In *Proceedings of ISCA*, 2007.

[44] X. Fan, H. Zeng, and C. Ellis. Memory Controller Policies for DRAM Power Management. In *Proceedings of ISLPED*, 2001.

[45] Z. Fang, L. Zhang, J. Carter, S. McKee, and W. Hsieh. Online Superpage Promotion Revisited (Poster Session). *SIGMETRICS Perform. Eval. Rev.*, 2000.

[46] B. Ganesh et al. Fully-Buffered DIMM Memory Architectures: Understanding Mechanisms, Overheads, and Scaling. In *Proceedings of HPCA*, 2007.

[47] P. Gratz, B. Grot, and S. W. Keckler. Regional Congestion Awareness for Load Balance in Networks-on-Chip. In *Proceedings of HPCA*, 2008.

[48] D. Gupta, S. Lee, M. Vrable, S. Savage, A. C. Snoeren, G. Varghese, G. M. Voelker, and A. Vahdat. Difference engine: Harnessing memory redundancy in virtual machines. In *OSDI*, 2008.

[49] H. Hanson and K. Rajamani. What Computer Architects Need to Know About Memory Throttling. In *Workshop on Energy Efficient Design, co-located with ISCA*, 2010.

[50] N. Hardavellas, M. Ferdman, B. Falsafi, and A. Ailamaki. Reactive NUCA: Near-Optimal Block Placement And Replication In Distributed Caches. In *Proceedings of ISCA*, 2009.

[51] J. L. Henning. SPEC CPU2006 Benchmark Descriptions. In *Proceedings of ACM SIGARCH Computer Architecture News*, 2005.

[52] J. Howard, S. Dighe, S. Vangal, G. Ruhl, S. J. N. Borkar, V. Erraguntla, M. Konow, M. Riepen, M. Gries, G. Droege, T. Lund-Larsen, S. Steibl, S. Borkar, V. De, and R. V. D. Vijngaart. A 48-Core IA-32 Processor in 45 nm CMOS Using On-Die Message-Passing and DVFS for Performance and Power Scaling. *IEEE JSSC*, January 2011.

[53] J. Howard, S. Dighe1, Y. Hoskote1, S. Vangal, D. Finan, G. Ruhl, D. Jenkins, H. Wilson, N. Borkar, G. Schrom, F. Pailet, S. Jain, T. Jacob, S. Yada, S. Marella, P. Salihundam, V. Erraguntla, M. Konow, M. Riepen, G. Droege, J. Lindemann, M. Gries, T. Apel, K. Henriss, T. Lund-Larsen, S. Steibl, S. Borkar, V. De1, R. V. D. Wijngaart, and T. Mattson. A 48-Core IA-32 Message-Passing Processor with DVFS in 45nm CMOS. In *Proceedings of ISSCC*, 2010.

[54] H. Huang, P. Pillai, and K. G. Shin. Design And Implementation Of Power-Aware Virtual Memory. In *Proceedings Of The Annual Conference On Usenix Annual Technical Conference*, 2003.

[55] H. Huang, K. Shin, C. Lefurgy, and T. Keller. Improving Energy Efficiency by Making DRAM Less Randomly Accessed. In *Proceedings of ISLPED*, 2005.

[56] H. Huang, K. Shin, C. Lefurgy, K. Rajamani, T. Keller, E. Hensbergen, and F. Rawson. Software-Hardware Cooperative Power Management for Main Memory. 2005.

[57] IBM. *IBM Power 795: Technical Overview and Introduction*, 2010.

[58] Intel. Intel 7500/7510/7512 Scalable Memory Buffer (Datasheet). Technical report, 2011.

[59] B. Jacob, S. W. Ng, and D. T. Wang. *Memory Systems - Cache, DRAM, Disk*. Elsevier, 2008.

[60] J. W. Jang, M. Jeon, H. S. Kim, H. Jo, J. S. Kim, and S. Maeng. Energy Reduction in Consolidated Servers Through Memory-aware Virtual Machine Scheduling. *IEEE Transactions on Computers*, 2010.

[61] J. Jeddeloh and B. Keeth. Hybrid Memory Cube – New DRAM Architecture Increases Density and Performance. In *Symposium on VLSI Technology*, 2012.

[62] JEDEC. *JESD79: Double Data Rate (DDR) SDRAM Specification.* JEDEC Solid State Technology Association, Virginia, USA, 2003.

[63] JEDEC. *JESD79-3D: DDR3 SDRAM Standard*, 2009.

[64] Y. Jin, E. J. Kim, and K. H. Yum. A Domain-Specific On-Chip Network Design for Large Scale Cache Systems. In *Proceedings of HPCA*, 2007.

[65] N. Jouppi. Improving Direct-Mapped Cache Performance by the Addition of a Small Fully-Associative Cache and Prefetch Buffers. In *Proceedings of ISCA-17*, pages 364–373, May 1990.

[66] A. Kahng, B. Li, L.-S. Peh, and K. Samadi. ORION 2.0: A Fast and Accurate NoC Power and Area Model for Early-Stage Design Space Exploration. In *Proceedings of DATE*, 2009.

[67] R. E. Kessler and M. D. Hill. Page Placement Algorithms for Large Real-Indexed Caches. *ACM Trans. Comput. Syst.*, 10(4), 1992.

[68] C. Kim, D. Burger, and S. Keckler. An Adaptive, Non-Uniform Cache Structure for Wire-Dominated On-Chip Caches. In *Proceedings of ASPLOS*, 2002.

[69] D. E. Knuth. *The Art of Computer Programming: Fundamental Algorithms*, volume 1. Addison-Wesley, third edition, 1997.

[70] C. Kozyrakis. Memory Management Beyond Free(). In *Proceedings of the International Symposium on Memory Management*, ISMM '11, 2011.

[71] C. Kozyrakis, A. Kansal, S. Sankar, and K. Vaid. Server engineering insights for large-scale online services. *Micro, IEEE*, 30(4):8 –19, july-aug. 2010.

[72] R. LaRowe and C. Ellis. Experimental Comparison of Memory Management Policies for NUMA Multiprocessors. Technical report, 1990.

[73] R. LaRowe and C. Ellis. Page Placement policies for NUMA multiprocessors. *J. Parallel Distrib. Comput.*, 11(2), 1991.

[74] R. LaRowe, J. Wilkes, and C. Ellis. Exploiting Operating System Support for Dynamic Page Placement on a NUMA Shared Memory Multiprocessor. In *Proceedings of PPOPP*, 1991.

[75] A. Lebeck, X. Fan, H. Zeng, and C. Ellis. Power Aware Page Allocation. In *Proceedings of ASPLOS*, 2000.

[76] C. Lefurgy et al. Energy management for commercial servers. *IEEE Computer*, 36(2):39–48, 2003.

[77] C. E. Leiserson. Fat-Trees: Universal Networks for Hardware-Efficient Supercomputing. *IEEE Trans. Comput.*, 34(10), Oct. 1985.

[78] K. Lim, J. Chang, T. Mudge, P. Ranganathan, S. K. Reinhardt, and T. F. Wenisch. Disaggregated Memory for Expansion and Sharing in Blade Servers. In *Proceedings of ISCA*, 2009.

[79] K. Lim, Y. Turner, J. Santos, A. AuYoung, J. Chang, P. Ranganathan, and T. F. Wenisch. System Level Implications of Disaggregated Memory. In *Proceedings of HPCA*, 2012.

[80] W. Lin, S. Reinhardt, and D. Burger. Designing a Modern Memory Hierarchy with Hardware Prefetching. In *Proceedings of IEEE Transactions on Computers*, 2001.

[81] S. Liu, K. Pattabiraman, T. Moscibroda, and B. G. Zorn. "Flicker: Saving Refresh-Power in Mobile Devices through Critical Data Partitioning". In *Proceedings of ASPLOS*, 2011.

[82] C. K. Luk, R. Cohn, R. Muth, H. Patil, A. Klauser, G. Lowney, S. Wallace, V. J. Reddi, and K. Hazelwood. Pin: Building Customized Program Analysis Tools with Dynamic Instrumentation. In *Proceedings of PLDI*, 2005.

[83] P. Magnusson, M. Christensson, J. Eskilson, D. Forsgren, G. Hallberg, J. Hogberg, F. Larsson, A. Moestedt, and B. Werner. Simics: A Full System Simulation Platform. *IEEE Computer*, 35(2):50–58, February 2002.

[84] Micron Technology Inc. *Micron DDR2 SDRAM Part MT47H64M8*, 2004.

[85] Micron Technology Inc. *Micron DDR3 SDRAM Part MT41J64M16*, 2006.

[86] R. Min and Y. Hu. Improving Performance of Large Physically Indexed Caches by Decoupling Memory Addresses from Cache Addresses. *IEEE Trans. Comput.*, 50(11), 2001.

[87] N. Muralimanohar, R. Balasubramonian, and N. Jouppi. Optimizing NUCA Organizations and Wiring Alternatives for Large Caches with CACTI 6.0. In *Proceedings of MICRO*, 2007.

[88] D. G. Murray, S. H, and M. A. Fetterman. Satori: Enlightened page sharing. In *In Proceedings of the USENIX Annual Technical Conference*, 2009.

[89] O. Mutlu and T. Moscibroda. Stall-Time Fair Memory Access Scheduling for Chip Multiprocessors. In *Proceedings of MICRO*, 2007.

[90] B. Mutnury et al. Analysis of Fully Buffered DIMM Interface in High-Speed Server Applications. In *Proceedings of Electronic Components and Technology Conference*, 2006.

[91] J. Navarro, S. Iyer, P. Druschel, and A. Cox. Practical, Transparent Operating System Support For Superpages. *SIGOPS Oper. Syst. Rev.*, 2002.

[92] O. Mutlu and T. Moscibroda. Parallelism-Aware Batch Scheduling - Enhancing Both Performance and Fairness of Shared DRAM Systems. In *Proceedings of ISCA*, 2008.

[93] K. Oh and X. Yuan. *High-Speed Signaling: Jitter Modeling, Analysis, and Budgeting.* Pearson Technology Group, 2011.

[94] J. Ousterhout et al. The Case for RAMClouds: Scalable High-Performance Storage Entirely in DRAM. *SIGOPS Operating Systems Review*, 43(4), 2009.

[95] T. Pawlowski. Hybrid Memory Cube (HMC). In *HotChips*, 2011.

[96] L.-S. Peh and W. Dally. A Delay Model and Speculative Architecture for Pipelined Routers. In *Proceedings of HPCA*, 2001.

[97] N. Rafique, W. Lim, and M. Thottethodi. Architectural Support for Operating System Driven CMP Cache Management. In *Proceedings of PACT*, 2006.

[98] Rambus Inc. Rambus XDR Architecture. http://www.rambus.com.

[99] L. Ramos, E. Gorbatov, and R. Bianchini. Page Placement in Hybrid Memory Systems. In *Proceedings of ICS*, 2011.

[100] S. Rixner, W. Dally, U. Kapasi, P. Mattson, and J. Owens. Memory Access Scheduling. In *Proceedings of ISCA*, 2000.

[101] T. Romer, W. Ohlrich, A. Karlin, and B. Bershad. Reducing TLB and Memory Overhead Using Online Superpage Promotion. In *Proceedings of ISCA-22*, 1995.

[102] G. Sandhu. DRAM Scaling and Bandwidth Challenges. In *NSF Workshop on Emerging Technologies for Interconnects (WETI)*, 2012.

[103] T. Sherwood, B. Calder, and J. Emer. Reducing Cache Misses Using Hardware and Software Page Placement. In *Proceedings of SC*, 1999.

[104] B. Sinharoy et al. IBM Power7 Multicore Server Processor. *IBM Journal of Research and Development*, 55(3), 2011.

[105] A. Snavely, D. Tullsen, and G. Voelker. Symbiotic Jobscheduling with Priorities for a Simultaneous Multithreading Processor. In *Proceedings of SIGMETRICS*, 2002.

[106] D. R. Stauffer, J. T. Mechler, M. Sorna, K. Dramstad, C. R. Ogilvie, A. Mohammad, and J. Rockrohr. *High Speed Serdes Devices and Applications*. Springer Publishing, 2008.

[107] K. Sudan, N. Chatterjee, D. Nellans, M. Awasthi, R. Balasubramonian, and A. Davis. Micro-Pages: Increasing DRAM Efficiency with Locality-Aware Data Placement. In *Proceedings of ASPLOS-XV*, 2010.

[108] K. Sudan, K. Rajamani, W. Huang, and J. B. Carter. Tiered Memory: An Iso-Power Memory Architecture to Address the Memory Power Wall. *IEEE Transactions on Computers*, 2012.

[109] M. Swanson, L. Stoller, and J. Carter. Increasing TLB Reach using Superpages Backed by Shadow Memory. In *Proceedings of ISCA*, 1998.

[110] Synopsys Inc. Synopsys' DesignWare IP for PCI Express with Support for Low-Power Sub-States. `synopsys.mediaroom.com/index.php?s=43&item=1044`.

[111] M. Talluri and M. D. Hill. Surpassing the TLB Performance of Superpages with Less Operating System Support. In *Proceedings of ASPLOS*, 1994.

[112] Tezzaron Semiconductor. 3D Stacked DRAM/Bi-STAR Overview, 2011. `http://www.tezzaron.com/memory/Overview\_3D\_DRAM.htm`.

[113] K. Therdsteerasukdi, G.-S. Byun, J. Ir, G. Reinman, J. Cong, and M. Chang. The DIMM tree architecture: A high bandwidth and scalable memory system. In *Proceedings of ICCD*, 2011.

[114] S. Thoziyoor, N. Muralimanohar, and N. Jouppi. CACTI 5.0. Technical report, HP Laboratories, 2007.

[115] B. Verghese, S. Devine, A. Gupta, and M. Rosenblum. Operating system support for improving data locality on CC-NUMA compute servers. *SIGPLAN Not.*, 31(9), 1996.

[116] C. A. Waldspurger. Memory resource management in vmware esx server. In *OSDI*, 2002.

[117] D. Wallin, H. Zeffer, M. Karlsson, and E. Hagersten. VASA: A Simulator Infrastructure with Adjustable Fidelity. In *Proceedings of IASTED International Conference on Parallel and Distributed Computing and Systems*, 2005.

[118] D. Wang et al. DRAMsim: A Memory-System Simulator. In *SIGARCH Computer Architecture News*, September 2005.

[119] H.-S. Wang, L.-S. Peh, and S. Malik. Power-Driven Design of Router Microarchitectures in On-Chip Networks. In *Proceedings of MICRO*, 2003.

[120] M. Ware, K. Rajamani, M. Floyd, B. Brock, J. C. Rubio, F. Rawson, and J. B. Carter. Architecting for power management: The IBM POWER7 approach. In *The 16th IEEE International Symposium on High-Performance Computer Architecture (HPCA-16)*, January 2010.

[121] D. Watts, D. Furniss, S. Haddow, J. Jervay, E. Kern, and C. Knight. IBM eX5 Portfolio Overview: IBM System x3850 X5, x3950 X5, x3690 X5, and BladeCenter HX5. `www.redbooks.ibm.com/abstracts/redp4650.html`.

[122] D. Wu, B. He, X. Tang, J. Xu, and M. Guo. RAMZzz: Rank-Aware DRAM Power Management with Dynamic Migrations and Demotions. In *Proceedings of Supercomputing*, 2012.

[123] D. Ye, A. Pavuluri, C. A. Waldspurger, B. Tsang, B. Rychlik, and S. Woo. Prototyping a hybrid main memory using a virtual machine monitor. In *ICCD*, 2008.

[124] D. H. Yoon, J. Chang, N. Muralimanohar, and P. Ranganathan. BOOM: Enabling Mobile Memory Based Low-Power Server DIMMs. In *Proceedings of ISCA*, 2012.

[125] W. Zhang and T. Li. Characterizing and Mitigating the Impact of Process Variations on Phase Change based Memory Systems. In *Proceedings of MICRO*, 2009.

[126] X. Zhang, S. Dwarkadas, and K. Shen. Hardware Execution Throttling for Multi-core Resource Management. In *Proceedings of USENIX*, 2009.

[127] Z. Zhang, Z. Zhu, and X. Zhand. A Permutation-Based Page Interleaving Scheme to Reduce Row-Buffer Conflicts and Exploit Data Locality. In *Proceedings of MICRO*, 2000.

[128] H. Zheng et al. Mini-Rank: Adaptive DRAM Architecture For Improving Memory Power Efficiency. In *Proceedings of MICRO*, 2008.

[129] H. Zheng, J. Lin, Z. Zhang, and Z. Zhu. Decoupled DIMM: Building High-Bandwidth Memory System from Low-Speed DRAM Devices. In *Proceedings of ISCA*, 2009.

[130] Y. Zhou, J. Philbin, and K. Li. The Multi-Queue Replacement Algorithm for Second Level Buffer Caches. In *Proceedings of the General Track: 2002 USENIX Annual Technical Conference*, 2001.

[131] Y. Zhou, J. Philbin, and K. Li. The Multi-Queue Replacement Algorithm for Second Level Buffer Caches. In *Proceedings of USENIX*, 2001.

[132] Z. Zhu and Z. Zhang. A Performance Comparison of DRAM Memory System Optimizations for SMT Processors. In *Proceedings of HPCA*, 2005.