ABSTRACT

Title of dissertation: Multi-Level Main Memory Systems: Technology Choices, Design Considerations, and Trade-off Analysis

Paul Kenton Tschirhart, Doctor of Philosophy, 2015

Dissertation directed by: Professor Bruce Jacob
Department of
Electrical & Computer Engineering
University of Maryland, College Park

Multi-level main memory systems provide a way to leverage the advantages of different memory technologies to build a main memory that overcomes the limitations of the current flat DRAM-based architecture. The slowdown of DRAM scaling has resulted in the development of new memory technologies that potentially enable the continued improvement of the main memory system in terms of performance, capacity, and energy efficiency. However, all of these novel technologies have weaknesses that necessitate the utilization of a multi-level main memory hierarchy in order to build a main memory system with acceptable characteristics. This dissertation investigates the implications of these new multi-level main memory architectures and provides key insights into the trade-offs associated with the technology and organization choices that are integral to their design.

The design space of multi-level main memory systems is much larger than the traditional main memory system's because it also includes additional cache design

and technology choices. This dissertation divides the analysis of that space into three more manageable components. First, we begin by exploring the ways in which high level design choices affect this new type of system differently than current state of the art systems. Second, we focus on the details of the DRAM cache and propose a novel design that efficiently enables associativity. Finally, we turn our attention to the backing store and evaluate the performance effects of different organizations and optimizations for that system.

From these studies we are able to identify the critical aspects of the system that contribute significantly to its overall performance. In particular, we note that in most potential systems the ratio of hit latency to miss latency is the dominant factor that determines performance. This motivated the development of our novel associative DRAM cache design in order to minimize the miss rate and reduce the impact of the miss latency while maintaining an acceptable hit latency. In addition, we also observe that selecting the page size, organization, and prefetching degree that best suits each particular backing store technology can help to reduce the miss penalty thereby improving the performance of the overall system.

Multi-Level Main Memory Systems: Technology Choices, Design Considerations, and Trade-off Analysis

by

Paul Kenton Tschirhart

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2015

Advisory Committee:
Professor Bruce Jacob, Chair/Advisor
Professor Manoj Franklin
Professor Ankur Srivastava
Professor Alan Sussman
Professor Donald Yeung

For my parents, Mark and Deborah;

my sisters, Mary and Megan;

and

my wife, Tanya

Acknowledgments

First and foremost, I would like to thank my wife, Tanya, whose patient support and advice got me through more than one minor grad school crisis.

I am also deeply grateful for my wonderful family who have continually supported me through a long and difficult grad school journey. If it were not for their constant love and encouragement, I am not sure I would have made it to this point.

I owe thanks to my fellow labmates: Ishwar Bhati, Mu-Tien Chang, Elliot Cooper-Balis, Paul Rosenfeld, and Jim Stevens. The many discussions that I have had with them over the years have helped me to become a more knowledgeable and skilled researcher. Special thanks goes to Jim Stevens for all of his programming guidance. My coding habits have been forever improved as a result of his incessant badgering during my early days with the lab.

I would also like to thank my advisor, Bruce Jacob, for his vision and for providing the opportunities that have shaped my career as a computer architecture researcher.

Finally, I would especially like to thank my mentors and friends at Intel Labs: Shih-Lien Lu, Zeshan Chishti, Chris Wilkerson, and Jim Greensky. Shih-Lien's guidance has played a critical role in my development as a researcher. In addition, Chris also made a significant impact on my career. In fact, two chapters of this dissertation began as a lunch conversation with him. Most importantly though, I am grateful to Zeshan for his truly assiduous support and thoughtful advice, without which finishing my PhD would have undoubtedly been much more difficult.

Table of Contents

List of Tables

List of Figures

Chapter 1: Introduction and Motivation

For more than three decades the main memory system has been built using only Dynamic Random Access Memory (DRAM) [23]. However, the scaling of DRAM has approached its limits and the density of DRAM devices has stagnated as a result. Meanwhile, data sets have continued to grow at a rate of 50% annually, quickly out pacing the growth of the main memory system [24]. In response, new technologies and architectures have been proposed as a means of continuing the development of larger, faster, more power efficient main memory systems that can meet the growing needs of high performance computing applications. All of the candidate alternative technologies, though, have serious drawbacks that limit their usefulness in a monolithic main memory organization like the current architecture. As a result, many of the newly proposed architectures which utilize these technologies make use of a hierarchical, heterogeneous organization that uses a faster, less dense memory technology to cache a more dense, slower one [25–27].

This dissertation investigates these new multi-level main memory architectures in order to improve the understanding of the complex dynamics that determine their performance. In particular, we perform a series of studies to determine which aspects of the system form bottlenecks and to gauge its performance relative to

existing architectures. We also analyze the cache and backing store sub-systems that make up the hierarchy in order to ascertain their unique design requirements. Together these studies reveal how the different components of the system interact and the overall effects of these interactions, enabling the design of future multi-level main memory systems which take into account the impact of the diverse technology, organization, and optimization choices available.

1.1 Main Memory Trends

The work in this dissertation was inspired by five recent trends that have been observed both in academic research and in industry developments. They include the observed slowdown in DRAM scaling, the emergence of alternative memory technologies, the development of alternative architectures to support the novel memory technologies, the introduction of high performance SSDs, and the appearance of in-package DRAM caches. Together these trends have led to and shaped the heterogeneous, hierarchical main memory architecture that is the focus of this work.

1.1.1 DRAM Scaling

For many years, DRAM was able to achieve regular increases in density which, in turn, enabled the continued capacity growth of DIMMs and the main memory system. However, with each successive scaling generation it has become more difficult to fabricate smalller DRAM cells. This can be seen in the trend in Figure 1.1 where the distance between each capacity generation has grown over time. Re-

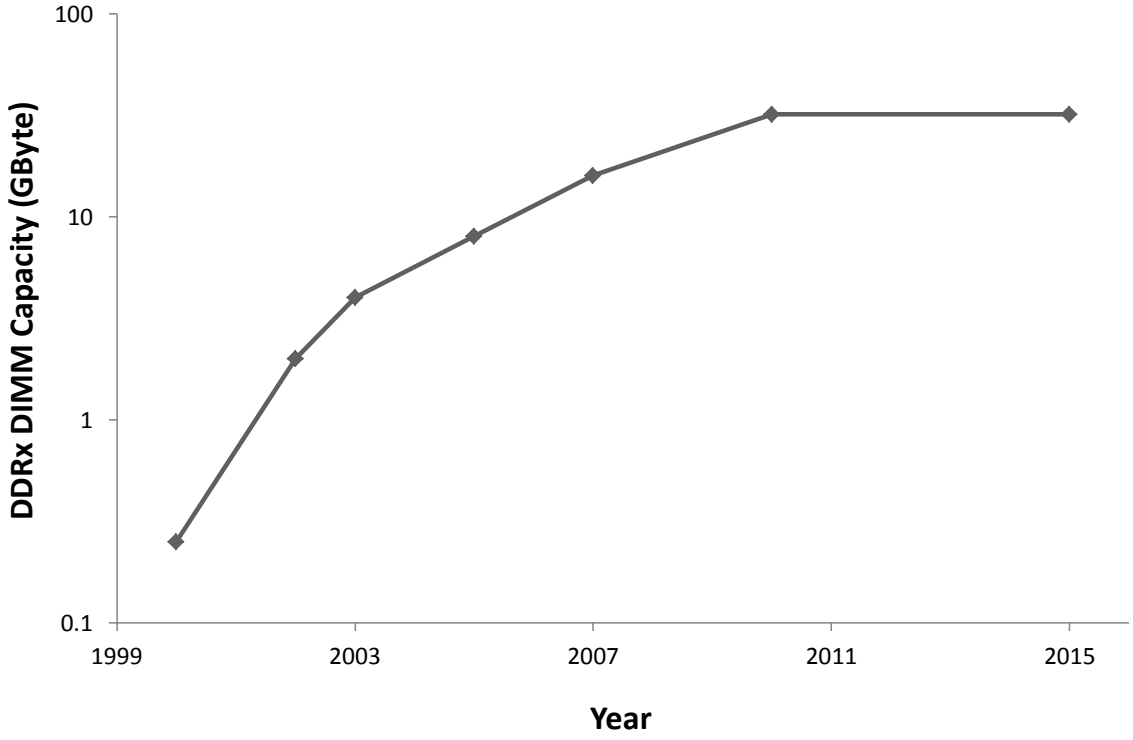

Figure 1.1: The slowdown in standard DDRx DRAM DIMM capacity improvement over the past 15 years. [3–8]

cently, scaling of DRAM DIMM capacity has virtually ceased. While specialized DIMMs are available that enable higher capacities by utilizing additional hardware, the standard commodity DDRx DRAM DIMM has had a maximum capacity of 32GB for the past 4 years. At the same time, the working sets of programs have continued to grow and are beginning to exceed the available memory capacity in standard systems.

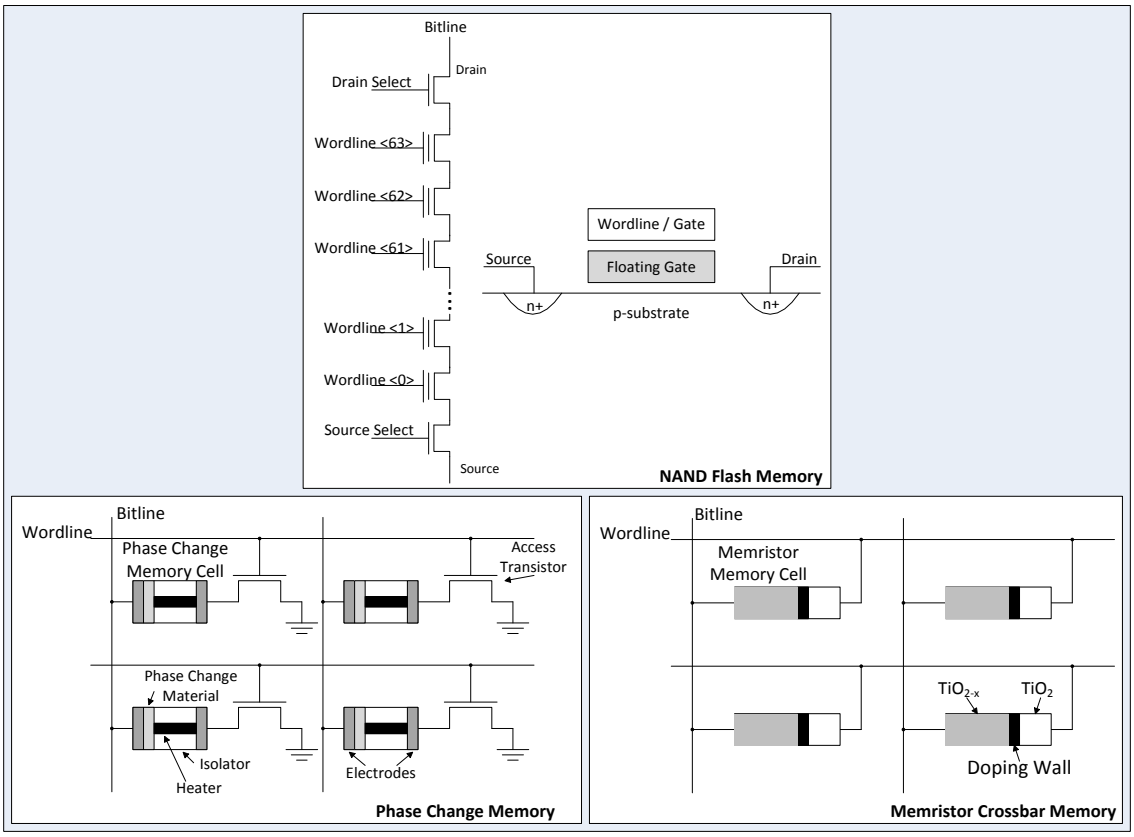

Figure 1.2: Some examples of recently proposed alternative non-volatile memory technologies that could be used to build multi-level memory systems.

1.1.2 Alternative Technologies

In response to the slowdown in DRAM scaling, increased efforts have been made to identify and develop a new technology that would either take its place or supplement it in order to continue the growth of main memory capacity. There are many potential non-volatile technologies that could one day fill this role including Phase Change Memory (PCM) [25, 26, 28, 29], Memristors [30], NAND Flash [31] and others [32]. However, of these candidate memory technologies, only NAND

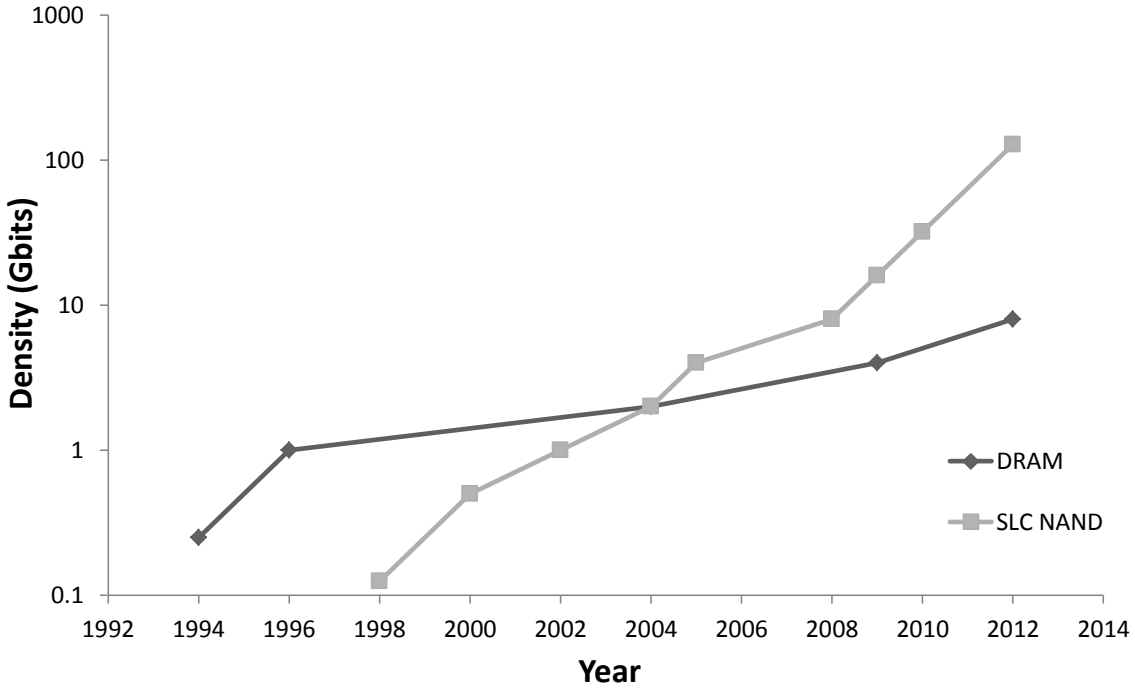

Figure 1.3: Scaling of NAND SLC Flash versus DRAM die density over time [9–20].

Flash is currently available in large quantities and has well established operating parameters. NAND flash lacks some of the possible performance advantages of the other technologies but it provides a considerable improvement to DRAM capacity scaling as can be seen in Figure 1.3.

1.1.3 Alternative Architectures

However, all of the potential memory technologies that are currently being considered have significant weaknesses that limit their ability to simply replace DRAM. As a result, significant efforts have been made in academic research to address this problem architecturally by introducing a multi-level memory system that utilizes a DRAM cache to counteract some of the drawbacks of the potential technolo-

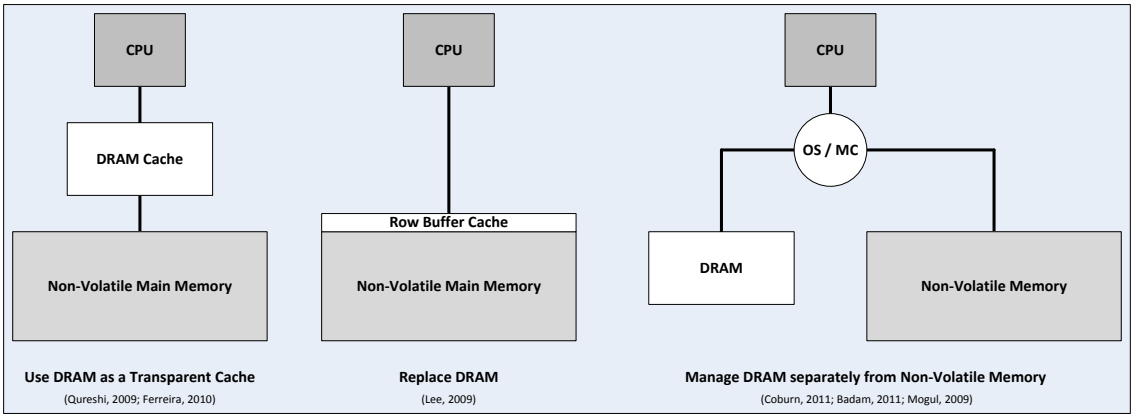

Figure 1.4: Some examples of proposed multi-level memory system architectures.

gies [25, 26, 28, 31]. Some of these approaches re-purpose a smaller version of the DRAM main memory system to serve a cache for the slower, more dense backing store technologies [25, 26, 31]. While other approaches attempt to completely replace the DRAM with the new technology all together by adding a more complex and deeper row buffer to the memory device [28]. This row buffer can be thought of as a private cache architecture where each memory device gets its own separate cache. Still other approaches prefer a software-driven solution that modifies that operating system or provides some additional software support so that the system can utilize these new memory technologies more efficiently [33–35]. These solutions differ from the hardware managed approach in that they tend to maintain the DRAM and non-volatile memory as separate address spaces. However, these different address spaces can still be thought of as a sort of hierarchy with the smaller, faster DRAM storing the more critical pages while the non-volatile memory stores everything else.

1.1.4 High Performance SSDs

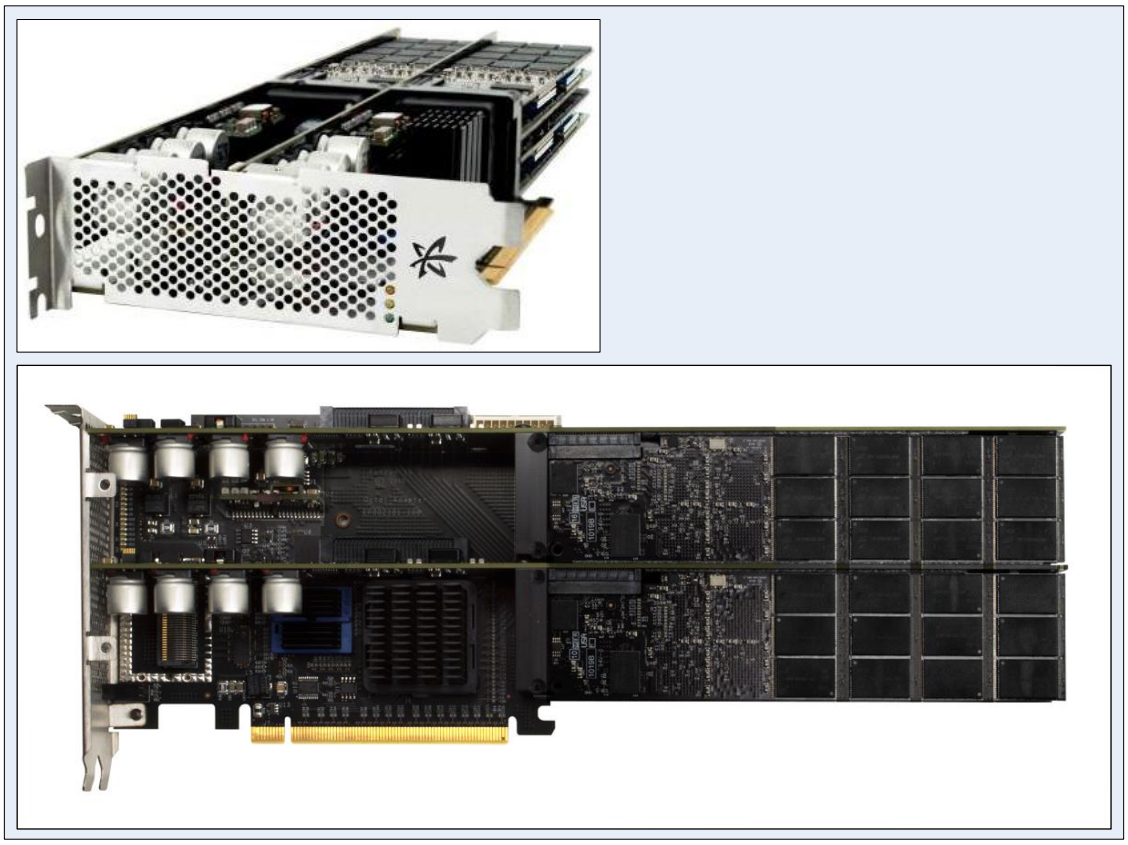

Figure 1.5: An example of a high performance PCIe SSD. [21, 22]

In the meantime, industry has moved forward with a short term solution in the form of PCIe SSDs. These devices help to reduce the impact of page faults by providing an additional layer between the DRAM and the disk that is composed of low cost NAND flash. This approach leverages the existing disk and OS infrastructure to enable the relatively easy integration of large quantities of NAND flash into the memory hierarchy. By treating the high performance NAND flash array like a disk or a peripheral device, these products are able to use the virtual memory system

or drivers to interact with them. This eliminates the need for hardware support by the CPU or memory controller and has allowed for the rapid adoption of these devices. The widespread utilization of these devices in data centers is a testament to the current need for additional memory capacity.

1.1.5 In-Package DRAM Caches

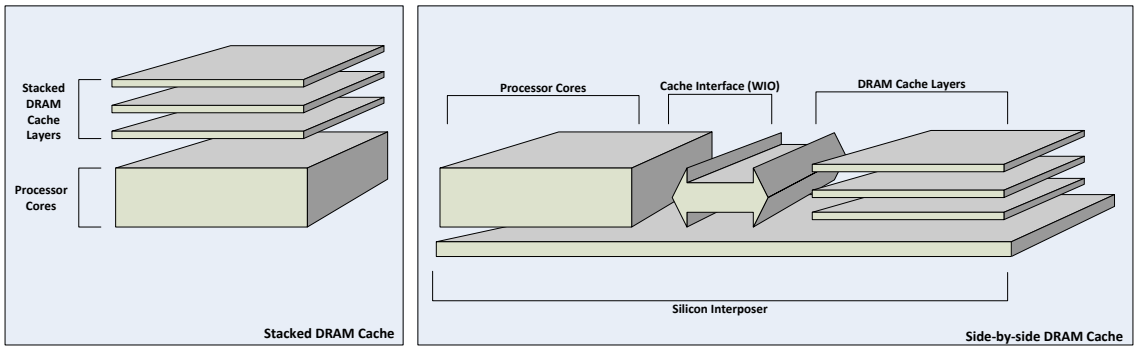

Figure 1.6: Some examples of recently proposed in package DRAM caches.

Finally, while some parts of industry have been focusing on improving the storage system to help counteract the effect of the DRAM scaling slowdown, other parts of industry have concentrated on extending the cache hierarchy instead. These solutions have taken several forms including 3-D stacking DRAM dies on top of the logic dies and the packaging of DRAM dies side by die with logic dies. The HMC architecture takes the 3-D stacking approach and could potentially place a large DRAM cache (refered to as the near memory) in package with the processor [36]. These HMC devices have remarkable potential but are not yet available on the market. The side by side approach, on the other hand, is more limited but a version

of it is currently available in some high end Intel server chips. However, in either case, these architectures provide an efficient way to implement the DRAM caches that are integral to the creation of multi-level main memory systems and their development has fueled additional research in that direction.

1.2 Problem Description

The development of multi-level main memory systems has opened a vast new design space for the development of future memory architectures that could be constructed to satisfy a wide variety of goals. However, in general, this new class of system is not particularly well understood. It features all of the design considerations that must be taken into account when creating a new main memory but also includes additional cache and memory technology concerns as well. As a result, it is often difficult to anticipate all of the effects that design adjustments will have on the performance of the entire system due to the complex relationships that exist between its different components. In addition, some of the problems related to the cache design, like meta-data storage, are unique to DRAM caches and therefore require novel solutions. Furthermore, other difficulties can arise from the unique characteristics of many of the potential memory technologies that can be utilized in these architectures. This dissertation helps to resolve this situation by performing a detailed study of the different levels of these systems in order to better understand their interactions.

In particular this dissertation is mainly concerned with the following five ques-

tions. What are the primary bottlenecks in multi-level main memory architectures? How do these architectures compare to current architectures? What aspects of DRAM cache design most significantly affect its performance? Can associativity be implemented in DRAM caches in an efficient way? And finally, how can backing stores that utilize potential non-volatile technologies be engineered to reduce the miss penalty?

1.3 Contributions and Significance

The primary contributions of this dissertation are as follows:

1. We develop a new suite of simulators that enable the investigation into multi-level memory systems that utilize novel memory technologies that currently lack established parameters and access protocols. This simulation infrastructure consists of new simulators for the cache controller (HybridSim) and the novel memory system (OMS) and also incorporates existing DRAM and processor simulators to allow for studies involving all levels of the computer system. In addition to facilitating the studies presented in this dissertation, these simulators also allow for the rapid prototyping of other novel memory technologies due to the simplified access protocol model used by OMS.

2. We analyze the impacts of different design decisions and technology choices on the performance of the overall system. Most importantly, we observe that the performance of most multi-level main memory system organizations is primarily determined by the ratio of their cache access latency to their backing store

access latency. As a result, reducing the impact of the miss penalty is critical to ensuring sufficient performance for most systems. Additionally, these studies also enable us to identify the maximum backing store access latency at which a hardware managed cache achieves an acceptable performance advantage over the current state of the art system that utilizes a high performance PCIe SSD.

3. We investigate the benefits and drawbacks of hardware and software DRAM cache management for different system designs. In particular, we characterize the overhead of the OS and file system on page faults compared to a hardware managed page replacement system. We show that the software delays can account for roughly 50% of the total access time in software managed systems. We also demonstrate that by utilizing hardware management and avoiding the software overheads, it is possible to achieve up to a 7x boost in performance for random read workloads.

4. We determine the influences of workload characteristics on system performance, in particular the degree of random access, the memory footprint, the number of threads, and the L3 Misses Per Kilo Instructions (MPKI). Our experiments show that these properties of workloads tend to determine both how much stress is placed on the cache and how cacheable the working set will be. As a result, by taking into account these aspects of a workload it is possible to understand which components of the system it is most likely to emphasize. For instance, the frequency of random accesses in a workload determines how

much of the working set will be cacheable and therefore can highlight the miss latency of an architecture. This information can also be used to identify which architectures are best suited for a particular class of applications.

5. We quantify the impact of cache size, miss rate, and miss latency on the performance of different DRAM cache designs. The results from these studies show that associativity can provide a considerable performance speedup in situations where the backing store latency is significantly longer than the cache access latency. However, we note that there is a delicate balance between the benefits of associativity and hit latency such that providing associativity at the expense of too much hit latency can ultimately hinder performance. Importantly, these results also show the limited utility of implementing DRAM caches with greater than 4-way associativity.

6. We highlight the importance of taking the DRAM memory system structure into account when implementing caches with DRAM. Our results show that the failure to take into account the concurrent structures within the DRAM address space can result in system slow downs of more than 2x.

7. We propose a new associative DRAM cache design that efficiently provides tag storage and lookup that we refer to as Combo-Tag. There are three key innovations that enable this design: coalesced tag accesses for multiple sets, a small 4KB SRAM tag buffer, and a novel tag replacement algorithm for the tag buffer. Combo-Tag successfully balances hit latency, miss latency, and memory usage to provide an associative cache that achieves a 14% average reduction

in access latency (up to a 30-60% reduction in many cases). Furthermore, it achieves this while utilizing 10-100x less on chip tag storage and 50% fewer bytes in DRAM to store tags when compared to the current state of the art DRAM cache designs.

8. We explore the impact of page size and concurrency on backing store performance for several different potential access latencies. The results from these studies indicate that as backing store latency increases, larger page sizes and additional concurrency are required in order to provide acceptable performance.

9. We analyze the effects of prefetching with different page sizes and prefetching degrees on the overall performance of the system. From these experiments we see that prefetching can provide roughly a 1.2x increase in overall performance by reducing the number of misses in the cache. However, our results also reveal that it is possible to over prefetch and seriously diminish system performance as well.

10. We evaluate the effects of channel organization on backing store performance and determine whether a relatively narrow host channel can be used to reduce the pin out of potential backing store architectures. In this study we show that the performance impact of introducing a buffered host channel is less than 10% of the ideal unbuffered case. We also demonstrate that the performance of different host channel organizations varies with the access latency of the backing store.

Chapter 2: Main Memory Background

Before we can discuss multi-level main memory systems and the wide variety of design decisions that are introduced by them, we must first establish some baseline knowledge regarding the existing memory system. The overall structure of the main memory system as it exists today has been largely unchanged for several decades. There are layers of on-chip SRAM cache that contain the most accessed and, therefore, the most important portions of the address space. These caches are then backed up by the main memory system which utilizes slower DRAM to provide greater capacity and lower cost at the expense of performance. In turn, the main memory system acts as a another sort of cache for the disk, storing data that is not important enough to reside in the SRAM but important enough to necessitate avoiding frequent long latency disk operations to access it. This hierarchy of SRAM caches, DRAM main memories and Disks has worked relatively well up until recently. However, as we saw in Chapter 1, workload demands and technology limitations are forcing the middle layer of the traditional hierarchy, the DRAM main memory, to expand into its own separate hierarchy adding new layers and complexity to the overall memory system. In this chapter we will focus on the structure of the DRAM main memory portion of the overall memory hierarchy in an effort to better

acquaint the reader with the existing architecture of the system that the rest of this dissertation will be working to reinvent.

2.1 Main Memory Organzation

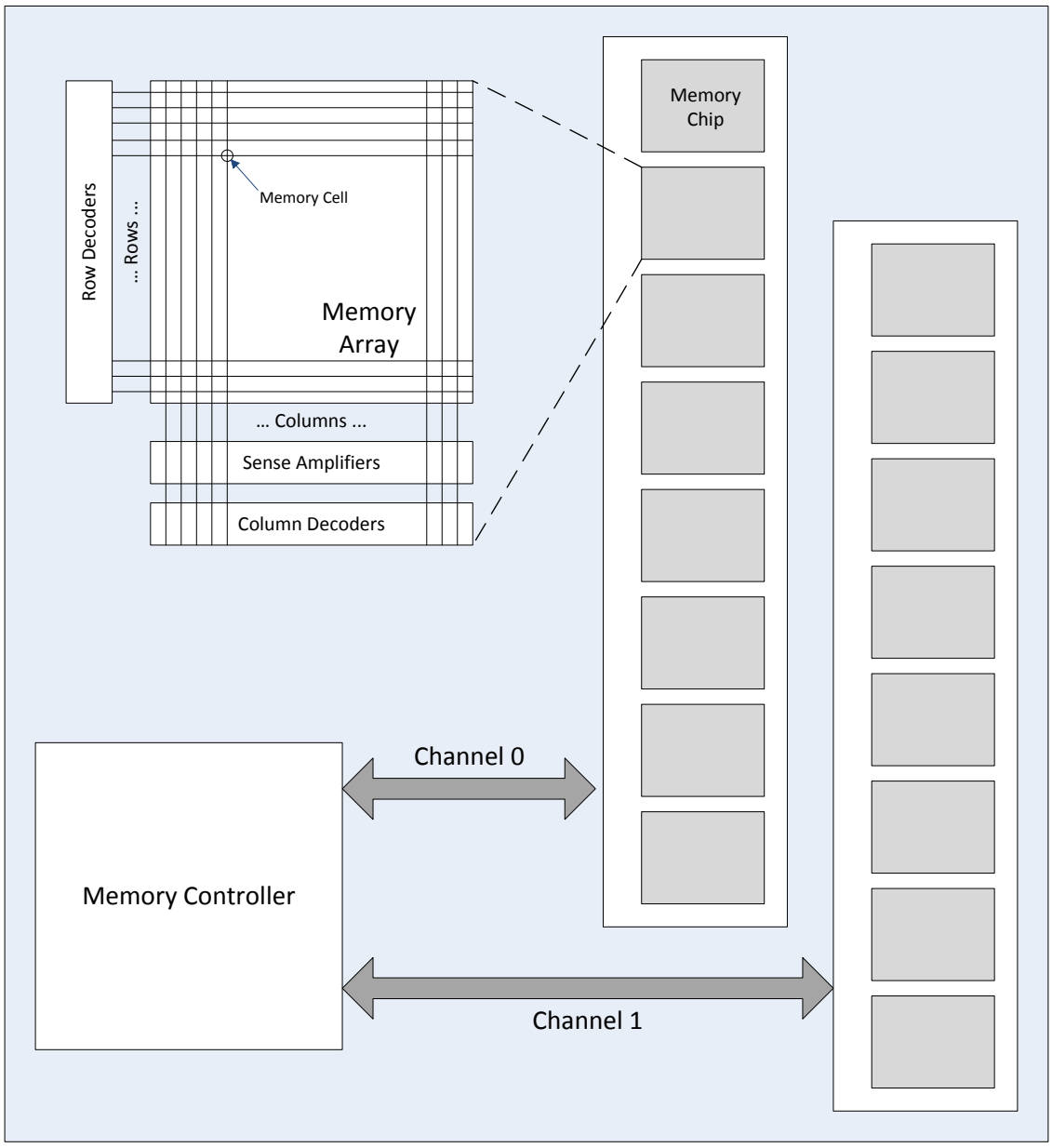

Figure 2.1: The structure of a typical main memory system.

Regardless of the underlying cell technologies all random access main memories take on the form presented in Figure 2.1. They consist of an array of memory cells that are organized in columns and rows with a memory cell at the intersection of each column and row. This organization allows all of the bits to be accessed independently and with essentially the same latency. In contrast, sequential access memories like hard disk drives experience different latencies for different bits. In order to access the bits decoders are needed to determine where a particular cell is located in the array. These take inputs in the form of row and column addresses and translate them by selecting the appropriate row and column lines. Once a cell or group of cells has been selected, the sense amplifier circuitry is used to drive the data from the data buffers into the selected cell or read the data present in those cells out to the data buffers. Of these five components - the cell array, the row and column decoders, the sense amps and the data buffers - the only two which change significantly from technology to technology are the cell array and the sense amplifiers. Different structures are needed for these components for some of the technologies due to differences in their operating principles. However, throughout the remainder of this dissertation the same five components will be present in each memory technology discussed.

In addition to the five components that make up the internal structure of the memory devices, there are also four components that make up the overall memory system. These components are the memory controller, the channels, the memory ranks, and the memory banks. The memory banks are concurrent portions of the memory device which can each perform a memory operation largely independent

from the other portions of the device. In some technologies, such as flash, different names are given to the separate concurrent units which make up the device. However, most kinds of memory have some form of internal concurrency. In addition, the devices can then be grouped into ranks that act together to contribute a portion of each memory access to that rank. In this way, devices with relatively narrow interfaces can be used to make wide channels. These channels then connect the memory controller, which typically resides on chip in today's microprocessors, to the ranks of devices which are usually packed in a Dual In-line Memory Module (DIMM). Unlike the five internal components of the memory devices, two of these four external system components will vary significantly from architecture to architecture independent of technology: the ranks and the channels. The rest of this chapter will focus on the different ways to implement these two integral system components.

2.2 Channel Organization

One or more of the channel configurations pictured in Figure 2.2 can be found in nearly every computer system. The most common channel organization for main memory systems is the one pictured in the lower left of Figure 2.2. This is the JEDEC (Joint Electron Devices Engineering Council) standard memory channel organization and it consists of a memory controller talking to memory modules which share several multi-drop buses. These buses provide the commands, addresses and data to the memory modules from the memory controller. Because the modules

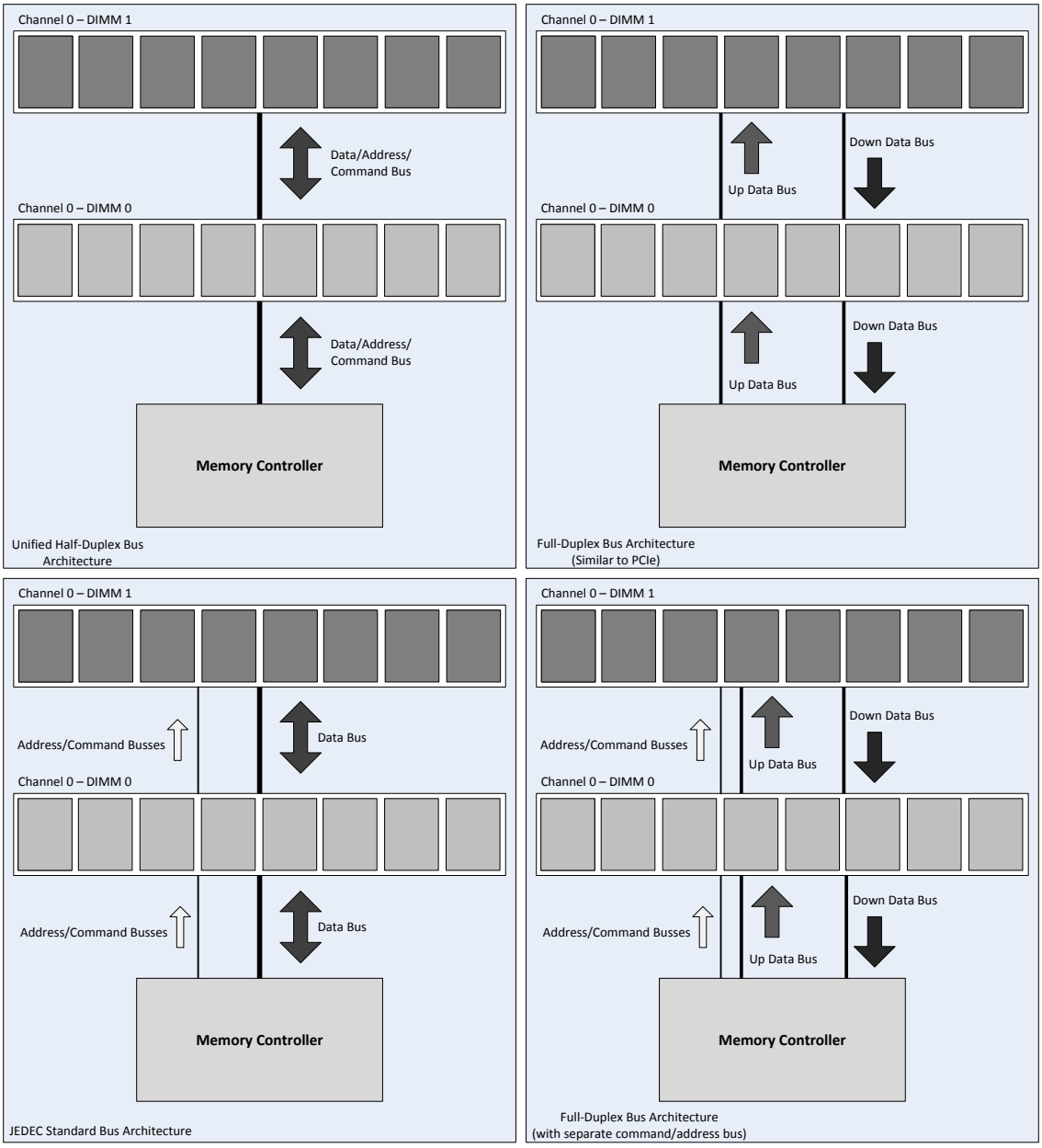

Figure 2.2: Conceptual examples of several different styles of channel organization.

share the address and data buses, chip select signals (not shown) are needed to indicate to a module that the commands and data are intended for it. Also, because there is only one data bus, it must allow data to travel both to and from the memory arrays using the same wires. This is referred to as a half duplex bus

because communication can only go one way at any moment in time. JEDEC is the body that has determined the industry-wide open DRAM standards since the advent of SDRAM. Therefore, because nearly all computers use the JEDEC standard memories, nearly all computers also use the corresponding bus organization. This organization makes sense for homogeneous main memory systems but it remains to be seen if it will be appropriate for heterogeneous systems. It is the current standard though and therefore serves as a good baseline comparison for any investigation into channel organizations for possible multi-level main memory organizations.

Another common channel organization is the full-duplex separate bus organization shown in the upper right of Figure 2.2. This is the channel architecture that is used by PCI Express (PCIe) devices such as graphics cards and high performance SSDs. Instead of utilizing separate buses for commands and data, this architectures features separate buses that go to and from the devices. Because these separate buses allow communication in both directions at the same time, the channel is referred to as full-duplex. The PCIe implementation of this channel architecture is point-to-point unlike the multi-drop architecture used by the JEDEC standard bus architecture, meaning that each channel can only talk to one thing. However, other separate full-duplex bus architectures such as the FBDIMM architecture allowed for multiple DIMMs to talk to the memory controller though the same channel by creating a multi-hop store and forward network using the DIMMs on the channel [37]. The channel architecture for the recently proposed Hybrid Memory Cube (HMC) also resembles this channel organization [36]. It should also be noted that the up and down buses do not have to be symmetric, meaning that it is possible to make

the up bus wider than the down bus or vice versa. Furthermore, determining the optimal ratio of up to down bus bandwidth is a complex problem that has been the subject of debate and has been investigated in the literature [38].

Aside from JEDEC and full-duplex bus channel architectures that can currently be found in most modern computers there are also two other potential channel organizations that we will consider in this work. The first is a very simple half-duplex bus that handles all of the system traffic, including commands, addresses, and data. This type of channel architecture resembles that chip interface used on many NAND flash memory chips. We would use this sort of architecture if we wanted to connect our main memory system channel directly to a simplistic chip interface like the one used by Flash chips [39]. The second alternative architecture is a purely hypothetical one that attempts to combine the features of the JEDEC standard architecture with the full-duplex bus architecture by providing both up and down buses as well as a separate command/address bus. These two channel architectures can also be found in Figure 2.2.

There are advantages and disadvantages to each of these different channel organizations in terms of performance and pin usage. For instance, the unified half-duplex architecture has the advantage of utilizing all of the available pin bandwidth to satisfy each operation. This means that the latency of individual operations may be considerably lower than an architecture like the full-duplex bus organization which only utilizes a portion of its available pin bandwidth for an operation. However, under heavy loads, the full-duplex channel's ability to receive data while sending commands or data simultaneously could provide better performance than

the raw bandwidth of the unified architecture. Also, half-duplex channels have a turn around time associated with them which can hamper performance in heavy traffic situations. Similarly, separating the command and address bus from the data bus can enable more efficient utilization of concurrency since commands can be sent while data is being received. However, this situation might not occur that often and in that case the extra pins being used to enable a separate command/address channel are essentially being wasted. So, it is clear that the design decisions regarding the channel architecture can greatly affect the efficiency and performance of the overall memory system as well as the entire computer system as a whole.

2.3 Ranking

Memory devices often feature relatively narrow interfaces since the number of pins that can be realistically implemented on a chip is limited by space and cost. For instance, many DRAM parts come in x8 or x16 configurations, meaning that they have 8 or 16 data pins. These narrow DRAM devices are then typically grouped together in ranks of 8 or 4 to form a 64 bit wide channel. This organization is shown in Figure 2.3. By grouping the devices together, this architecture is able to significantly reduce the transfer latency for an access. This is because each device will only have to send an 8 byte portion of the total 64 byte access. As a result, each device will only need to send 8 bursts instead of 64, which reduces the transfer time by a factor of 8. The downside to this arrangement is that it reduces the number of concurrent units in the system in order to achieve the reduction in latency. This

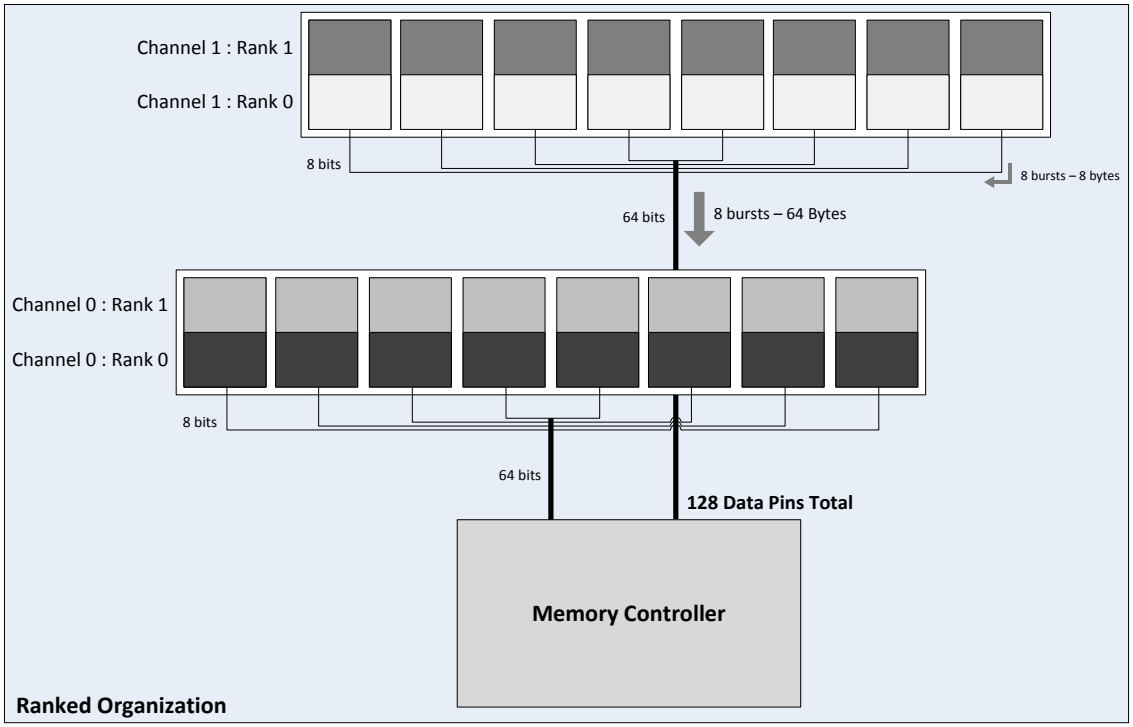

Figure 2.3: An example of a memory system with two channels and two 64 bit wide ranks per channel.

works well as long as the traffic going to the memory system is not too severe and does not exceed the amount of concurrency in the system per unit time. In other words, if there are 4 concurrent units in the system and they each take 50 ns to perform a memory operation, then as long as the request stream doesn't issue more then 4 requests every 50ns then everything should typically work well. However, if the traffic increases such that there are more than 4 requests every 50ns then some requests will wind up having to wait on others thereby increasing the average access latency. In that case, having some additional concurrency can wind up resulting in a lower average latency value even though the latency of a single memory operation is slightly longer due to the reduction in rank bandwidth.

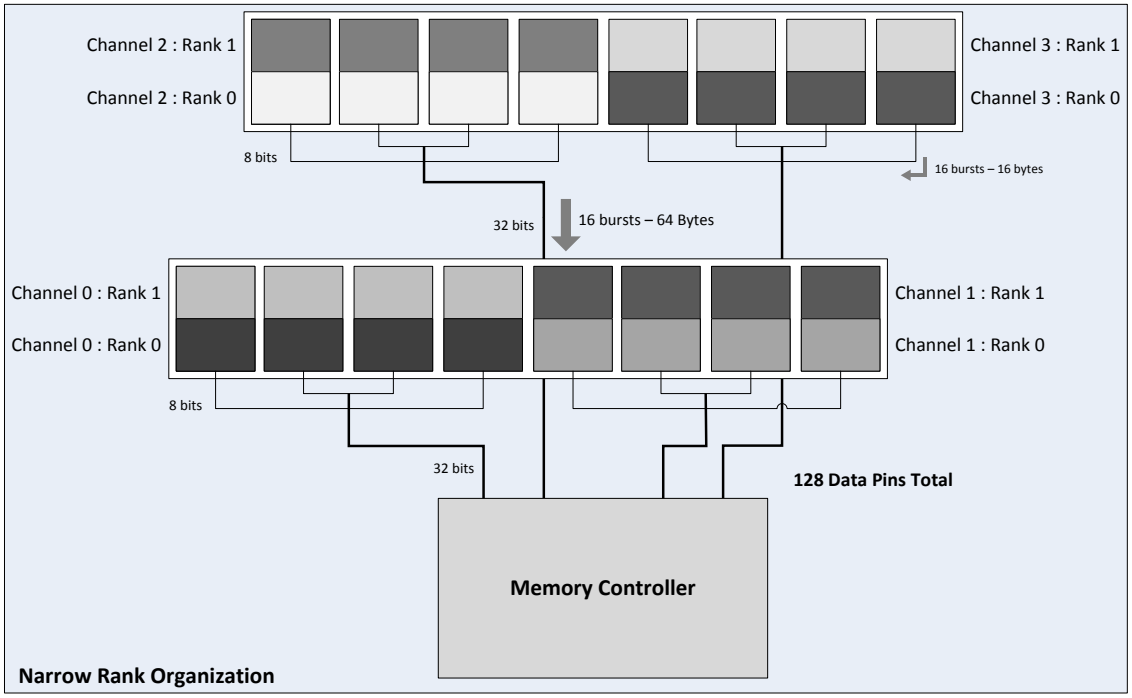

Figure 2.4: An example of a memory system with 4 narrow 32 bit wide channels and 2 ranks per channel.

In today's DRAM based memory systems, it is often the case that the request stream does not overload the available concurrency in the system. So, DRAM systems typically have a structure that closely resembles the one in Figure 2.3. However, as the latency of the memory technology being used increases, the likelihood that the request stream will overload the available concurrency per unit time also increases. Also, as the latency of memory operations increases, the percentage of the overall access latency that is due to transfer time decreases. For instance, in a DRAM system it might take 50 ns to perform a read and then another 4-5 ns to transfer the data back to the memory controller over the bus. Therefore, the 5 ns transfer time is 9% of the total access latency. But if the memory array takes

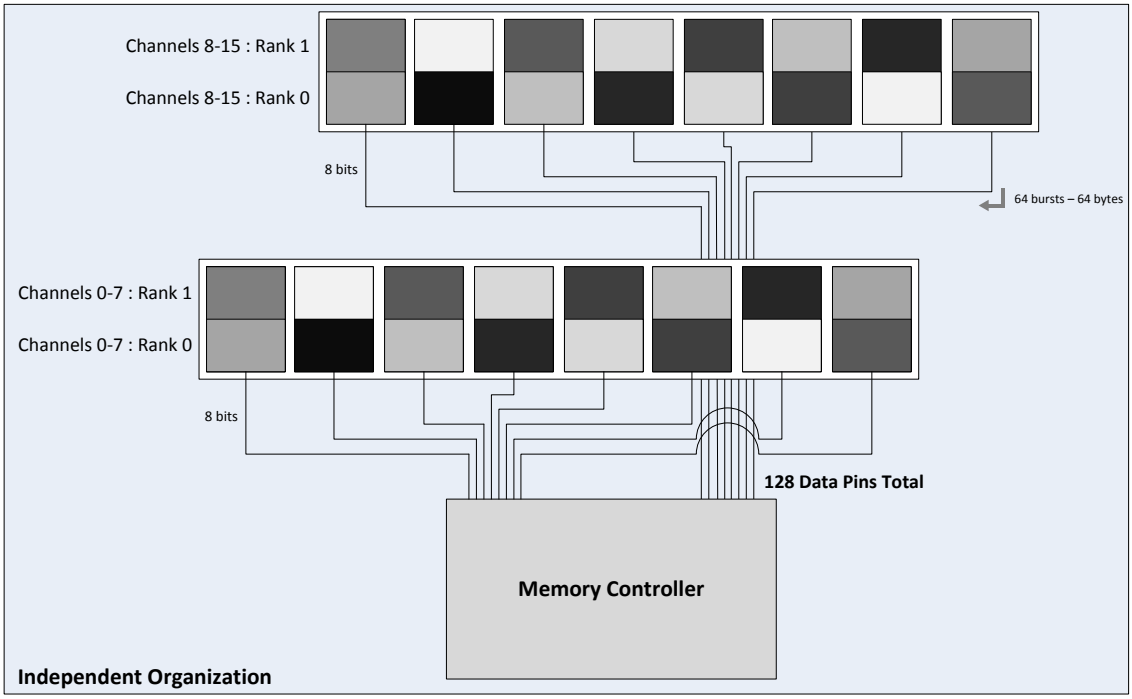

Figure 2.5: An example of a memory system with many very narrow 8-bit channels each of which has two ranks.

150 ns to perform a read, then the 5 ns transfer would only account for 3% of the total access latency. As a result, concurrency becomes more critical to system performance than transfer latency and adding more ranks to the system by using less chips becomes more attractive as access latency increases. An organization that is designed to accomplish this is shown in Figure 2.4. In this organization, fewer chips are used to create the ranks, which increases the latency of individual accesses but provides twice as much concurrency to the system.

It is also possible to have a channel organization which maximizes the amount of concurrency in the system by eliminating ranks altogether and having each chip operate independently. An example of this type of channel architecture is pictured

in Figure 2.5. This organization further extends the concept that was used by the narrow rank organization by maximizing the available concurrency in the system at the cost of even longer transfer latencies. In systems that use slow technologies such as flash, this sort of trade-off is often worth while because the long access latencies of the memory technology make the ability to perform operations in parallel extremely important.

Finally, it is important to point out that though the three designs presented in this section feature very different organizations they all utilize the same number of data pins. In many systems, the pins needed to create the buses are some of the most expensive parts of the entire design. As a result, keeping the number of pins to a bare minimum is often a critical design goal. Therefore, in order for different channel architectures to be considered equivalent it is necessary that they utilize that same number of pins.

2.4 Buffering

Another way to balance the transfer latency with the concurrency in the system is to introduce a buffer that can act as a mediator between a very fast host channel and slower device channels. Figure 2.6 shows an example of a buffered architecture. By sharing a very fast connection to the memory controller between many slower devices or ranks of devices it is possible to take full advantage of the bandwidth offered by such a fast channel. Ideally the individual chips would all be able to operate at the same speeds as the host channel. However, it is frequently the case

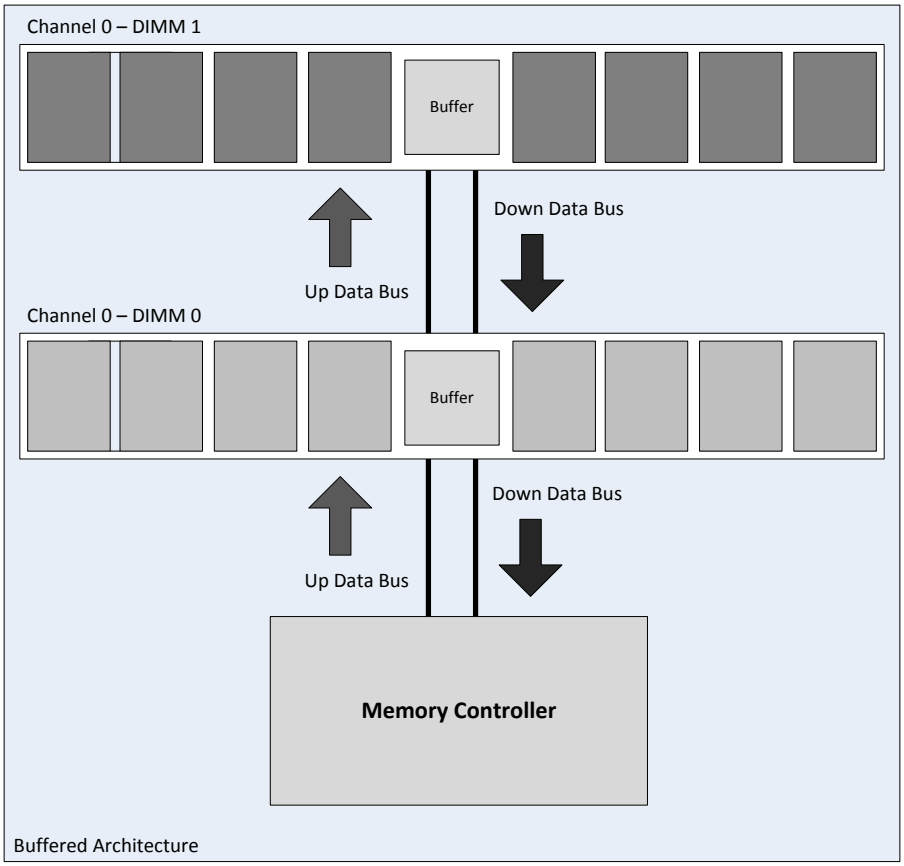

Figure 2.6: An example of a memory system that utilizes a buffer chip to act as an intermediary between a very fast split bus master channel and many slower full duplex chips with narrow device channels.

that it is simply not feasible to run the device interfaces at high speeds for power and cost reasons. Also, adding a buffer to the DIMM can enable the use of more memory devices without needing to increase the pin count on the memory controller for additional channels. There have been several proposed architectures that have looked into building memory system with a buffer chip either on the motherboard or on the memory modules used in the system [37, 38, 40–42]. However, most of these proposals have not seen widespread acceptance with the exception of LR-DIMM.

This may change as additional levels are added to the main memory system and as different memory technologies are utilized.

Chapter 3: Memory Technologies

For the past several decades, the main memory system has been built exclusively using standard DRAM parts [23]. This provided an acceptable balance between performance, power, price, and capacity for the main memory system until relatively recently. As DRAM scaling has slowed though, we have seen the introduction and development of new memory technologies intended to fill the gaps in performance, power, or capacity that have opened up. One of the most conspicuous of these new technologies is Flash, which has seen widespread adoption in the form of high performance PCIe SSDs. These SSDs help to bridge the performance gap between the DRAM and the disk layers of the system by providing a high capacity backing store for the DRAM. This helps to reduce the page fault penalty in systems that have working sets which exceed the reasonable capacity of a traditional DRAM main memory. Additionally, alternative DRAM designs such as LPDDR and RL-DRAM have also been developed and are commonly used in mobile devices. These devices use the same underlying DRAM technology but are designed to maximize performance or power at the cost of other metrics. Finally, other novel technologies which utilize entirely different mechanisms to store data are also being explored as a way to enable the continued improvement of the main memory system.

One of the advantages of a multi-level main memory architecture is that it allows the designer to utilize multiple memory technologies to build the main memory system. In this way the designer is able to leverage the strengths of the different technologies and construct a memory system with attributes that are not possible in a single layer homogeneous memory system. However, there are advantages and disadvantages to each of the potential technologies. Some of these characteristics are inherent to the technology itself, others are the result of design decisions that are made at the device level. Therefore, in order to correctly utilize these technologies, it is important to both understand how they work and to be familiar with the design decisions that shape the devices that use them. This chapter will focus on a broad selection of the potential main memory technologies that are available in hopes of providing some understanding of the trade-offs that are possible when building multi-level main memory systems.

3.1 DRAM

The JEDEC standard DRAM memory consists of a capacitor, which is used to store the electric charge that represents a 1 or a 0, and a transistor, which is used to control access to the capacitor. This simple two element cell pictured in Figure 3.1 is what is located at the intersection of each row and column in a DRAM array. When data is written to the cell, a voltage is applied to the gate of the transistor via the word line which causes it to turn on, connecting the bit line to the cell capacitor. The desired charge can then be driven onto the bit line and subsequently into the

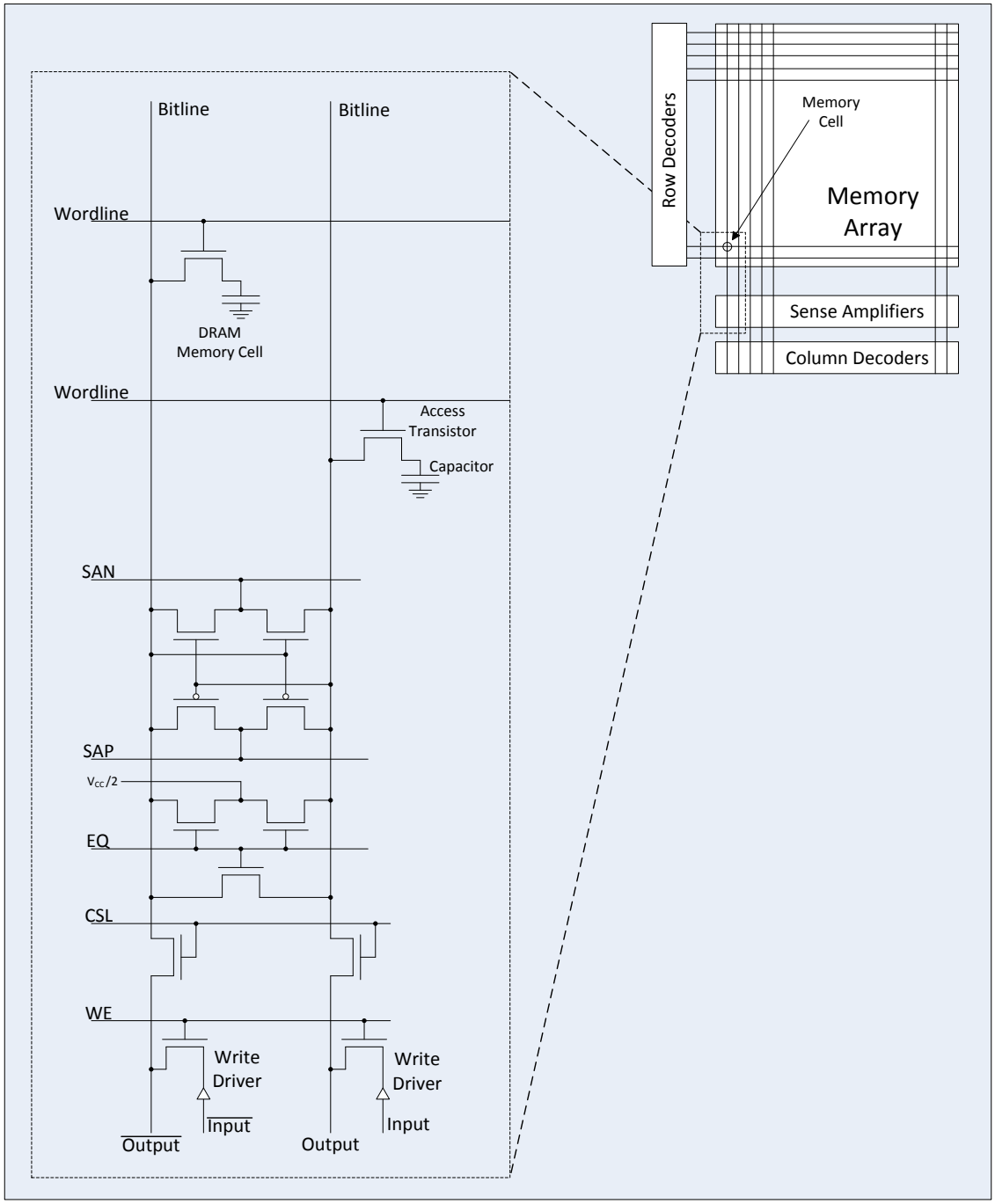

Figure 3.1: The structure of a typical DRAM-based memory array.

cell capacitor thereby storing a 1 or a 0. The cell is read by first precharging the bit lines so that they are halfway between the high and low voltage values. Then the

proper voltage is applied to the pass transistor thereby connecting the bit line to the capacitor. When this happens the charge on the capacitor causes the voltage of the bit line to either increase or decrease slightly. Delicate sense amplifier circuits are then used to detect the slight changes in the bit line's voltage and determine whether a particular cell had a charge on it or not.

3.1.1 Access Process

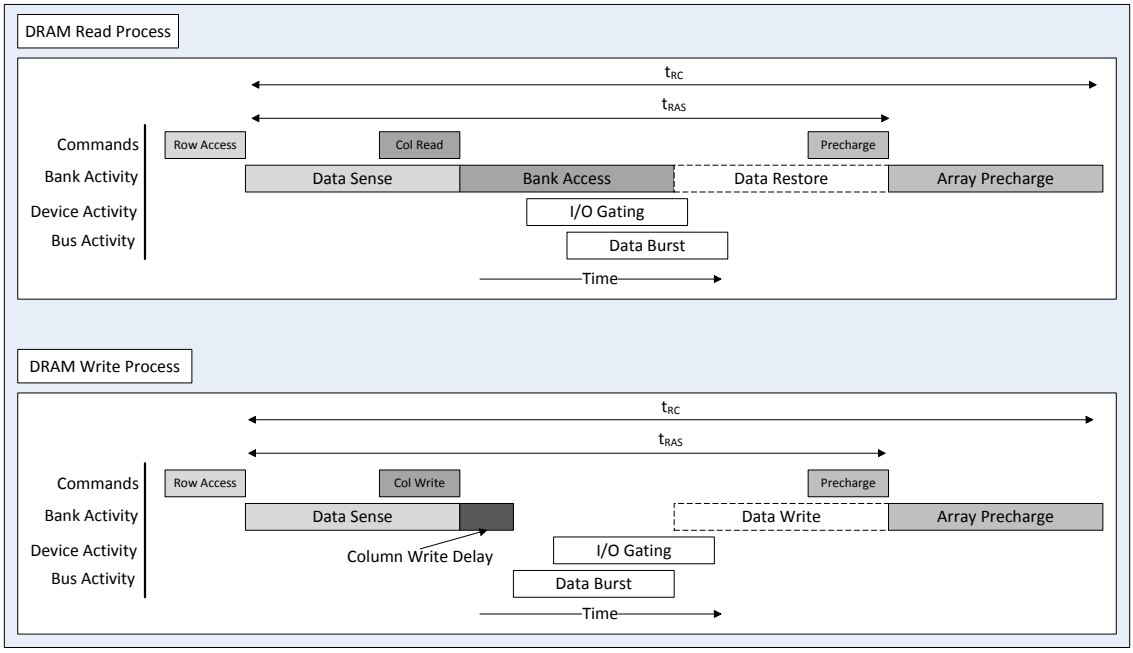

Figure 3.2: The complex method of writing to and reading from DRAM.

In the DRAM system many sense amplifiers are used at the same time to read out an entire row. The requested columns from that row are then sent back to the memory controller. From the perspective of the memory controller, the memory accesses look like the operations in Figure 3.2 where the timings and operation of

the sense amplifiers are represented by a single delay (the row access). DRAM reads and writes are very similar because they are comprised of the same four parts: a row access, a column access, a data restore and a precharge. During a DRAM read the charge on the capacitor is allowed onto the bit line. Consequently the data is lost from the storage cell and a write is required to restore the data. Therefore, a data restore operation can be seen in both operations in Figure 3.2, in the read it restores that original data and in the write it stores the new data. The primary difference between the read and the write besides the direction of data flow are the column access delays: t_{CL} for the read and t_{CWD} for the write. Despite this minor difference the operations are almost identical. As a result, the start to finish time of the two operations, t_{RC}, is the same. [43–45]

However, despite this similarity between reads and writes, the access operations of DRAM are considerably more complex than other technologies. This is because the DRAM access involves more steps than just a simple read or write command. For instance, most other technologies do not separate the column and row accesses nor do they feature a separate precharge command. In some ways this more complex access process is advantageous for DRAM as it allows for more complex and clever command scheduling algorithms that make very efficient use of the available hardware. However, this complexity also increases the bandwidth that is used for each access because multiple commands are required.

The circuit pictured in Figure 3.3 is the basic sense amplifier used in DRAM systems. It senses the difference in the voltages on the bit line that is connected to the cell being read and another reference bit line that is just precharged with the

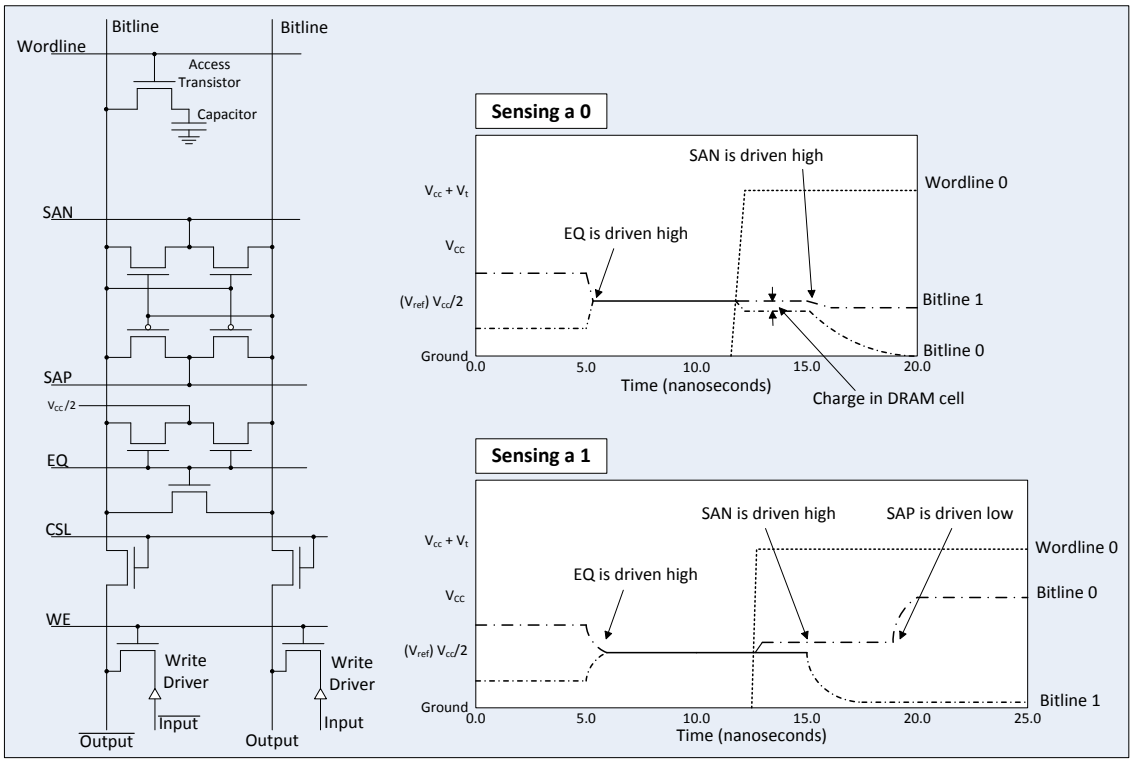

Figure 3.3: The operation of the sense amplifiers used to access the information in the DRAM array.

voltage halfway between the high and low voltages. The operation of this circuit can be seen in Figure 3.2. First, the voltage equalizing portion of the circuit is used to initially precharge both lines with the same voltage that is half of the high voltage. Then after enough time has passed to ensure that both lines have the same voltage the equalizing circuit is switched off. Next, the appropriate word line is driven high connecting the cell to the bit line. If the capacitor in the cell that is being read has a charge on it, this charge will cause a slight increase in the voltage of the bit line. Similarly, if the capacitor is not charged it will absorb some of the charge from the bit line causing the bit line's voltage to decrease slightly. Again some time

is taken to ensure that the lines have settled on their new values. After this time the SAN signal is driven high which connects the upper transistors in the sensing circuit in Figure 3.3 to ground via the SAN wire. If the voltage of the left bit line is slightly less than the other bit line then the upper right hand transistor will be more conductive and will pull the right bit line down to ground. This behavior can be seen in the "Sensing a 0" graph in Figure 3.3. In this way, the sense amplifier is able to detect that the cell being read did not have a charge on it. Conversely, if the cell does have a charge on it, it will cause the bit line to have a slightly higher voltage. Therefore, the reference bit line will be driven to ground when the SAN signal connects the sense amp to ground. The voltage of the bit line being read will still be just slightly more than the precharge voltage though. So, to push the value of that bit line up to the full high voltage the SAP signal is used to connect the lower transistors in the sensing circuit to VDD via the SAP wire. Since the left bit line is now equal to ground the bottom right transistor is fully on and conducts the full VDD onto the right bit line driving it high. This behavior can be seen in the "Sensing a 1" graph in Figure 3.3. The operation of this circuit is significant because it is the primary factor that determines the read latency of the DRAM system. The faster this circuit can sense the charges on the cell capacitors, the faster those values can be communicated back to whoever needs them. [1, 43]

As was mentioned earlier, DRAM is a volatile memory technology because it requires power in order to maintain the data already written to the array. Great care is taken to ensure that the pass transistors in DRAM cells can fully disconnect the capacitor from the bit line and prevent any charge from escaping the capacitor

before it is read. However, despite this effort the transistors are not perfect and so a very small amount of charge is still able to leak off of the cell capacitor. Over time this gradual loss of charge will result in the loss of data as the voltage level of the charged cells decreases below the value that is detectable by the sense amps. To counteract this, the data in each cell must be periodically read out and rewritten in order to preserve the proper charges on the capacitors. This process is referred to as refresh and it must be performed every 64ms in order to preserve the system's data integrity. [43]

The pass transistor leakage current that necessitates DRAM refresh is also one of the major barriers to DRAM scaling. As the pass transistors become smaller they are less able to hold back the charge on the capacitor. As a result, alterations to the pass transistor are often needed to enable continued scaling. In addition to the difficulty of scaling the pass transistor, it is also critical that the cell capacitor's capacitance is large enough to store enough charge regardless of the capacitor's physical size. If the capacitance isn't large enough then the sense amplifier won't be able to sense its influence on the bit lines during read operations. In fact, regardless of technology generation the DRAM cell capacitor must have more than 25fF of capacitance in order to serve as a storage element. These two requirements - the pass transistor that can prevent too much charge leakage and the capacitor that can store enough charge - are often barriers that must be overcome in order to enable scaling to the next generation. As the industry attempts to push scaling ever further, the solutions needed to satisfy those requirements are becoming more difficult to develop. Furthermore, it is conceivable that at some point it will simply

become physically impossible to satisfy those requirements at some scale. [46, 47]

3.1.2 Organization

DRAM is typically organized into channels, ranks, banks, rows and columns. The channels, ranks, and banks are the concurrent elements of the system, which are capable of performing memory operations independently of one another with some limitations. The channels are the physical connections between the memory modules and the memory controller. They are shared by the other concurrent units in the memory system and so form the most important layer of parallelism available in the memory system. The ranks are formed by ganging together memory devices that each contribute a part of the overall memory access. There can be multiple ranks on a memory module and they generally operate independently of one another. They do however contend with one another for shared resources such as the channel that connects the memory module to the memory controller. Within the ranks there is a final layer of concurrency formed by the banks. Banks are subsections of the memory device that can also act independently of one another with some limitations. The last two layers of the typical DRAM organization are the rows and columns which point to a specific location within the memory array. The row is a section of the array that is read into the sense amplifiers on a read command. The column refers to a 8 or 16 byte access within the larger row that is sent back to the memory controller in multiple one or two byte bursts.

3.1.3 Addressing

In order to access a particular piece of data within the DRAM memory system, the memory controller needs some way to know where to look within the very large address space of the memory. To accomplish this the numerical memory address is broken down into components that then index a specific channel, rank, bank, row, and column. Using a different set of bits from the address ensures that each numerical address maps to just one unique location in the memory system. So, if there are 4 channels then two bits are needed to index which channel a particular memory address belongs to. These bits can be taken from anywhere within the memory address but it is often optimal to use the lower order bits for this. This is because the lower order bits change more frequently than the upper bits so using them for the channel portion of the location ensures that memory accesses will be spread out across as many channels as possible thereby making maximum use of the available channel concurrency. Similarly, other segments of bits are chosen from the address to index different layers of the memory system in such a way that maximizes concurrency. However, this does not always mean that its best to assign bits to memory components in order of their perceived importance to system concurrency. In other words, it is not always optimal to assign the lowest bits to channel, the next lowest to the rank and so on.

One situation where it helps to use a slightly different address mapping is in systems that use an open page configuration. An open page memory system chooses to leave a row open in the sense amplifiers after accessing data. This means that

later accesses will have a lower latency if they map to the same row because the row access phase of the DRAM transaction can be skipped. So, in an open page system it is sometimes better to set up the address mapping so that sequential addresses will map to the same row. That way the likelihood of hitting an open row will be increased.

Therefore, selecting the correct address mapping scheme for a particular system is critical to its overall performance. If the wrong scheme is used then the available concurrency or open rows in the system will not be fully utilized. However, determining the correct scheme to use is not always a straight forward process because the address streams of different workloads can have very different patterns in terms of which bits change most often.

3.1.4 LPDDR

In response to the growing demand for low power memory devices, especially in mobile systems, the LPDDR type of DRAM was developed. Initially LPDDR was just a modified version of standard DDR SDRAM that had several features that reduced the overall power consumption of the device. In particular, it was able to operate with a supply voltage of just 1.8 V instead of the typical 2.5 V. It also featured temperature compensated refresh which could reduce the refresh frequency. DRAM cells need to refreshed less often when they are physically cooler, so the refresh frequency can be adjusted to take advantage of that when the device is cold. Finally, LPDDR also included the ability to completely turn off the device to save

power, at the cost of the data stored on the device. The development of LPDDR has continued over the years with the introduction of LPDDR2, LPDDR3, and LPDDR4. These later versions have improved upon the original idea by increasing data rates, providing more bandwidth, and further improving power efficiency.

These devices are typically used in cell phones and tablets where power efficiency is an important design consideration that must be balanced directly against performance and capacity. However, as the size of main memories in servers have increased, so has their power consumption. As a result, the memory system now accounts for a significant amount of the overall system's power consumption. This trend has led some researchers to investigate incorporating LPDDR as the main memory technology of servers [48].

3.1.5 RLDRAM

Another relatively new alternative version of DRAM is RLDRAM which, unlike LPDDR, optimizes for latency over power efficiency. RLDRAM devices make several design trade-offs in order to drastically shorten their random access latency. They are generally more power hungry and provide lower densities than standard DDR parts and also only offer closed page support. However, they can be as much as 5x faster than their current generation standard DDR counterparts. These devices were designed with networking, high-end commercial graphics, and L3 cache applications in mind because these applications tend to require low latency random accesses. With the development of in-package DRAM caches, they could also po-

tentially be a candidate for the last layer cache technology in those architectures as well, though their limited density is a concern. Some researchers have already begun exploring the possibilities of utilizing RLDRAM as a cache technology in heterogeneous memory systems [27].

3.2 NAND Flash

In many ways NAND flash memory is the polar opposite of DRAM in the world of random access memories. It is capable of extremely high densities but suffers from long read and write latencies. To amortize the cost of those long latency accesses flash performs operations at a page granularity (4-16 KB) as opposed to DRAM's much finer word granularity (64 B). In addition, unlike DRAM, NAND flash is non-volatile and does not require constant power to maintain the integrity of the data in its array.

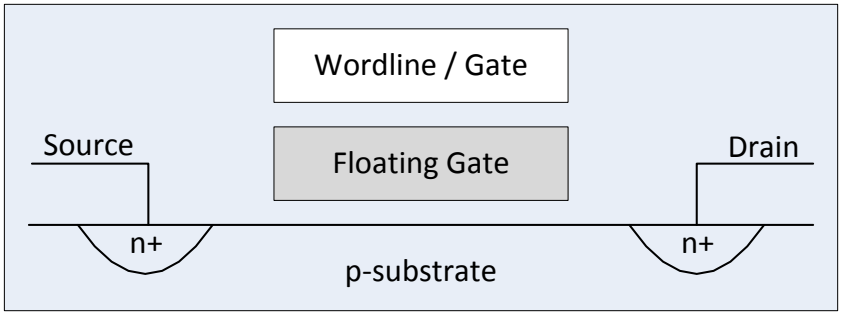

Figure 3.4: A typical flash cell with a floating gate between the substrate and the normal transistor gate.

Where DRAM utilizes capacitors to hold charges that represent 1s and 0s, flash uses floating gate transistors like the one pictured in Figure 3.4 which store

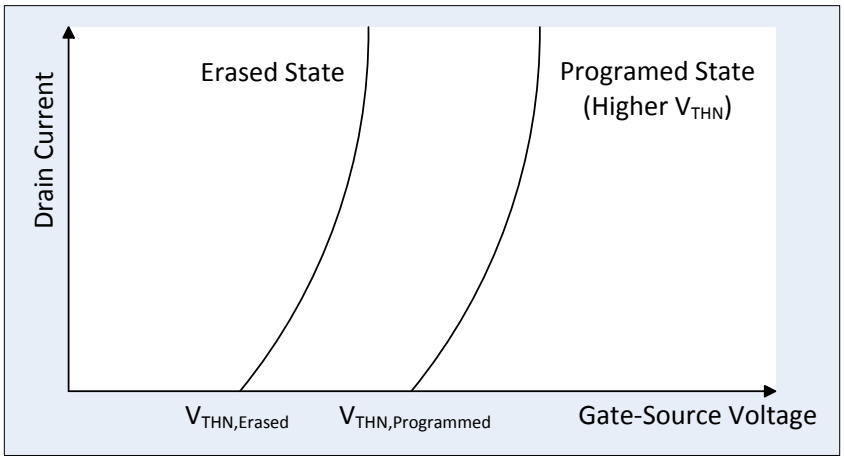

Figure 3.5: The shift in threshold voltage that occurs when charge is trapped on the floating gate of a flash cell.

the 1s and 0s as charges trapped in a floating gate between the channel and the normal control gate. This additional charge on the floating gate shifts the threshold voltage of the transistor so that it requires more voltage on the gate to turn on. This behavior is shown in Figure 3.5. Therefore, whether a gate contains a 1 or a 0 can be determined by detecting whether the transistor turns on for a particular voltage.

3.2.1 Organization

NAND flash is capable of higher densities than other styles of random access memory like DRAM because instead of storing a single bit in a cell, NAND flash cells are composed of strings of floating gates which each store a bit. This structure, which is depicted in Figure 3.6, enables higher density by increasing the ratio of storage components to access transistors. Each DRAM capacitor needs its own pass

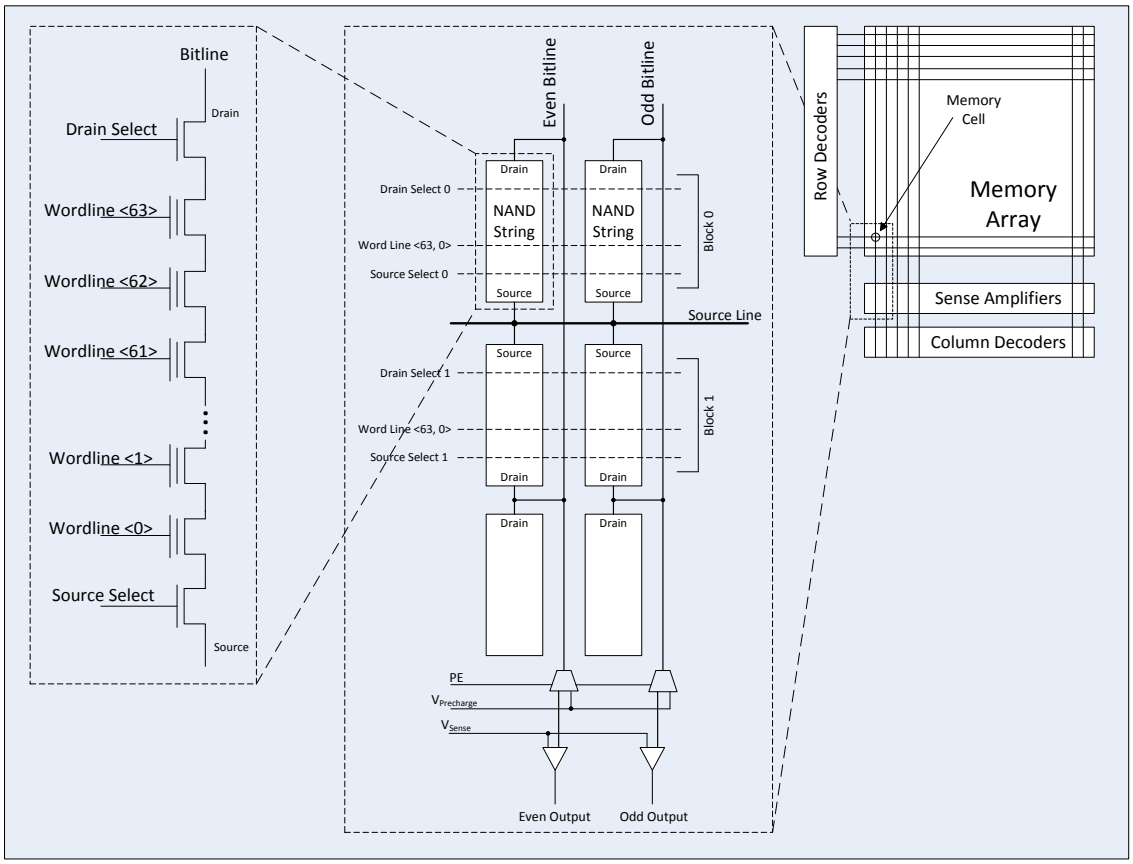

Figure 3.6: The structure of a typical NAND Flash-based memory array.

transistor in order to operate correctly however the structure of NAND flash only needs two transistors for a string of many floating gates. Furthermore, the length of that string is limited only by performance concerns since it affects the operation of the sense amplifier typically used with NAND flash. Even with these concerns typically there are 64 floating gates in each cell. So, there are 64 storage cells for every two pass transistors, a much better ratio than the 1:1 found in DRAM. [1, 49, 50]

At the memory system level, the NAND flash memory is typically organized into channels, dies, planes, blocks, and pages. This organization is somewhat sim-

ilar to the DRAM organization in that the first three levels represent the different kinds of concurrency that are available in the system. Like DRAM the highest and most effective form of concurrency available in the flash system is the channels that connect the chips to the memory controller. Within a flash device there is then also the possibility of having multiple dies, each of which can perform operations independently of one another. However, the dies share the same channel so like ranks and banks in DRAM, operations can only be interleaved across multiple dies. This provides concurrency but not the true parallelism that the channel level of concurrency provides. The plane level of the flash array organization is slightly more complex than the other kinds of concurrency that we have discussed thus far. Planes are capable of performing operations in parallel but they have to perform the same type of operation [39]. The last two layers of the flash array, the blocks and pages, do not enable any concurrent operations and are just organization units like the rows and columns in DRAM. The blocks of a flash system are the units that can be erased whereas pages can be written to and read from. There are typically 64 pages in each block though some flash devices can feature more and older flash devices often featured 32 pages per block.

3.2.2 Access Process

The more complicated structure of the NAND flash cell makes its operation less intuitive than that of simpler cells such as the DRAM cell. The table in Figure 3.1 provides an overview of the voltages needed to perform reads, writes and erases. To

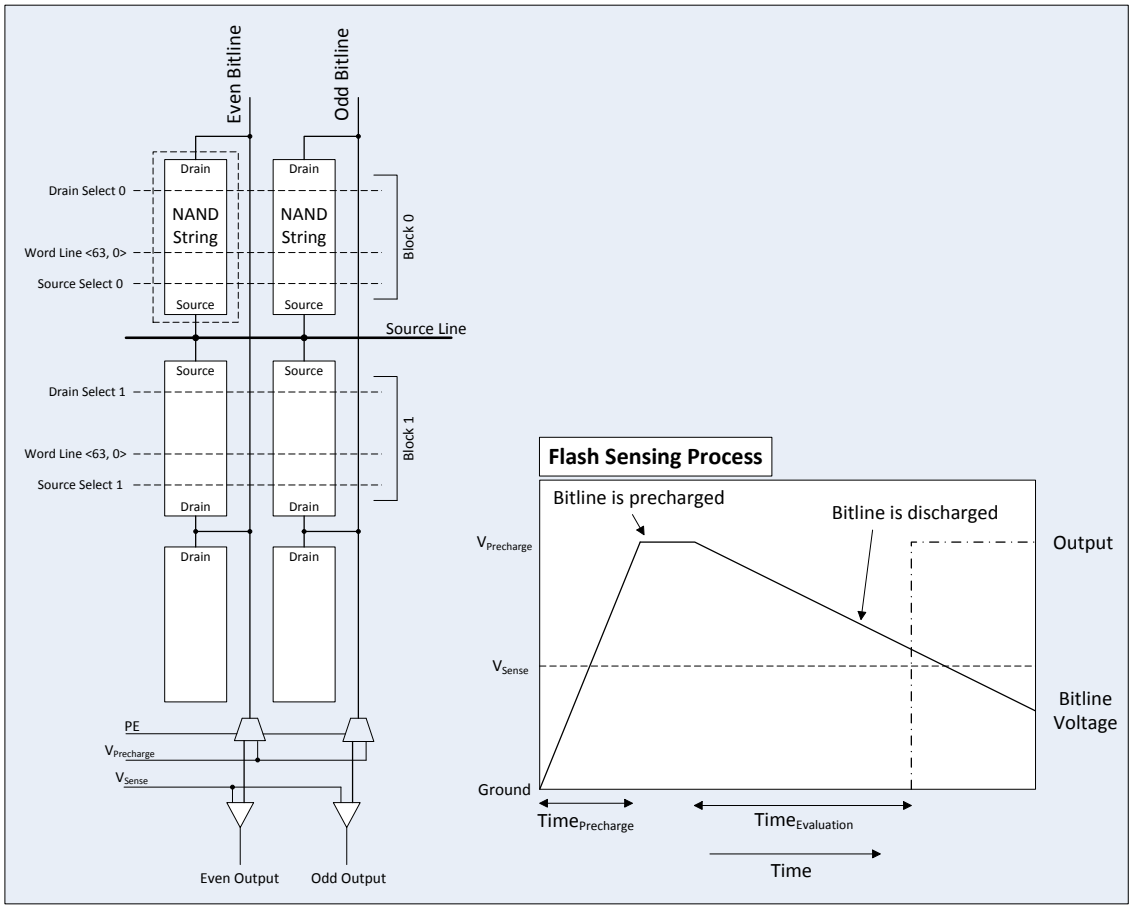

Figure 3.7: The charge run down method of reading a NAND flash cell.

erase the cell, the p substrate is strongly biased inducing Fowler-Nordheim tunneling which causes any charges on the floating gate to migrate to the substrate. After some time, most of the charges will be gone and the floating gate will be erased. Because the substrate is shared by many floating gates, the erase cannot erase a single floating gate at a time. Instead, the NAND flash die is divided up so that there are many areas that can be erased independently. These areas are referred to as blocks and generally contain many NAND strings which are all erased at the same time. To program a particular cell, its control gate is biased to a high voltage while

Inputs	Erase	Program	Read
Bit line	Floating	0 V	High or low
DSL	0 V	20 V	5 V
WL 63	0 V	20 V	0 V
WL 62	0 V	5 V	5 V
WL 2	0 V	5 V	5 V
WL 1	0 V	5 V	5 V
WL 0	0 V	5 V	5 V
SSL	0 V	0 V	5 V
Source Line	Floating	0 V	0 V
P well tie-down	20 V	0 V	0 V
Comment	Erases entire block	Programming cell 63	Reading cell 63

Table 3.1: Example inputs for each NAND flash cell operation [1].

the other control gates in the string are biased to a lower voltage. The high voltage on the control gate of the desired cell again induces Fowler-Nordheim tunneling which causes charges to be drawn from the substrate onto the floating gate. Finally, to read the cell its control gate is grounded while the other control gates in the string are biased to specific voltage. This voltage ensures that those transistors are turned on so that a current can flow through the string. If the floating gate that is being read was in the erased state it will allow current to flow while a programmed floating gate will not. The sensing circuits for the flash array then detect whether current was able to flow and use this information to determine the state of the floating gate. Setting the control gate to ground on the floating gate that is being read still allows

it to conduct because the erase process causes positive charges to tunnel onto the floating gate while the negative charges tunnel off. In this way an erased floating gate transistor actually has a threshold voltage lower than zero so grounding the control gate turns it on if it's erased. [1, 50]

Because it is possible to add different amounts of charge to the floating gate it is possible to adjust the threshold of the gate to different values. In this way, multiple bits can be stored using a single floating gate with each combination of bits corresponding to a different threshold level. For instance, for a two bit cell, the cell could contain the value 01 when its threshold is 1V and 10 when its threshold is 2V. This multi-level cell (MLC) capability is another factor which allows NAND flash to achieve much higher densities than other random access memories. However, the sense amplifiers must be able to detect the additional threshold levels. To accomplish this either more hardware is needed to test for each level simultaneously or more time is needed so that the same hardware can test for each level sequentially. Because NAND flash is used today primarily as a storage memory the same sensing hardware is often used despite the performance cost because it maximizes the density gains. In addition, MLC NAND flash has a shorter lifetime than single level cells (SLC). Over the lifetime of a flash cell its thresholds will shift due to various wear out mechanisms. Because MLC divides the threshold of an SLC into sub-thresholds which are much narrower it is less able to tolerate shifts in the threshold. However, this shorter lifetime is generally accepted because MLC vastly increases density without increasing price. [49]

The typical sensing circuits used with NAND flash determine whether a float-

ing gate was programmed by using a charged capacitor to generate current through the NAND string for a specific amount of time. If the floating gate being read is erased then the capacitor will have been completely discharged by the end of that time period. Otherwise there will still be some charge on the capacitor. A simple voltage comparator is then used to determine if there is any charge left on the capacitor. Because the amount of charge trapped on a programmed floating gate can vary, different amounts of current will flow for different floating gates that have all been programmed. Therefore, the voltage that is compared to the voltage of the NAND string is not zero but instead a threshold value. If the charge left on the NAND string is below this threshold, it is reasonable to conclude that the floating gate was conducting enough to be considered erased. This process and the related circuit is shown in Figure 3.7. [49]

Typically, NAND flash arrays use the capacitance of the bit line as the capacitor in order to reduce the hardware overheads introduced by this method of sensing. This is why the capacitor in Figure 3.7 is shown in dotted lines. However, though this method reduces hardware costs by eliminating a capacitor from the sensing circuit, it restricts the possible size of the discharging capacitor to the capacitance of the bit line. As can be seen in Figure 3.7, the time it takes the capacitor to discharge is the largest component of the overall time it takes to perform the sensing operation. Therefore, because this sensing hardware uses the relatively large bit line capacitance as its capacitor it is much slower than it would be if it used a very small capacitor that could discharge quickly. Furthermore, the adjacent bit lines can interact with the discharging of the bit line that is being read.

Therefore, they need to be forced to a fixed voltage in order to electrically shield the bit line being read from that interaction. This means that only half of the bit lines can be read at any one time. While the even bit lines are being read, the odd bit lines are forced to the fixed voltage to shield them and vice versa. So, a read operation in NAND flash actually involves two sense operations but requires half the sensing circuits that would normally be required thereby further reducing the hardware overhead. Other potentially faster sensing circuits are possible but would require more hardware. However, these circuits are not used because NAND flash is primarily a storage memory. Therefore, the performance cost of using the bit line capacitance is considered acceptable since it greatly reduces the peripheral hardware, reducing cost and improving density. [49]

3.2.3 Addressing and Garbage Collection

The programming operation of NAND flash memory can only add charge to the floating gate and so an erase is needed before data can be rewritten into a NAND cell. However, the erase process takes a very long time and so performing an erase on a cell before rewriting data back into it would greatly reduce the performance of NAND flash. Therefore, subsequent writes to the same host address are actually sent to different cells in the NAND flash array. An address map is maintained which keeps track of which NAND cells contain the data for a particular host address. When a host address is rewritten, the data is placed in a new location in the array, then the address map entry is updated to reflect this and the old location is marked

as dirty. So, unlike DRAM, the address translation scheme which breaks down the numerical address into channel, rank, bank, row, and column bits isn't critical to properly utilizing the available concurrency in the system. Instead, the address mapping scheme is responsible for assigning pages in a way that efficiently utilizes concurrency.

When it is possible to perform an erase without hindering performance too much, a process referred to as garbage collection cleans the dirty portions of the array by erasing them. This process is not trivial because the erase is performed for a whole block of the NAND array. Often, some valid data still exists in the block and must be copied to a new location before the erase can proceed. This leads to an effect called "write amplification" because a page of valid data may be moved around multiple times inside the NAND array to avoid several different erases. This results in multiple writes for a single piece of data even though the user only ever wrote the data once. To compound this problem, the floating gates can only be programmed so many times before they begin to break down and no longer work. So, not only do the additional writes result in longer erase times and unnecessary power use, they also cause the device to wear out more quickly.

Wear leveling algorithms are used to try to combat the wear out problem while minimizing write amplification. These algorithms come in two varieties that deal with the two different kinds of data that can be present in the flash array: dynamic wear leveling and static wear leveling. Dynamic data is data that is frequently accessed and changed whereas static data tends to remain in the array unused for long periods of time. These two kinds of data have very different effects on the

overall wear-out of the flash array and therefore have to be dealt with in different ways. [51]

Dynamic wear leveling algorithms address the basic problem of wear out due to frequent accesses by trying to evenly spread out the incoming writes evenly so that individual gates all have roughly the same amount of wear [51]. This is generally accomplished by mapping the incoming writes to locations in the NAND flash array using an algorithm that attempts the balance the traffic going to each section of the array. The most basic version of this dynamic wear leveling is a clock-like algorithm that distributes the writes across the array in a round robin fashion. However, once every page has been written, the subsequent writes will need to be placed in blocks that have been erased. This means that this algorithm depends heavily on the garbage collection process selecting blocks to erase in an even fashion so that some blocks are not erased and written to more often than others. More complex algorithms can be devised which avoid this problem by tracking the erase count of the various blocks in the array and adjusting the write placement accordingly.

However, a large amount of the data that is typically added to a flash array is static data. As a result, it can become very difficult to find blocks to erase that either do not contain lots of clean, valid pages or have not been erased recently. The static data in each block means that additional writes will have to be done to get the valid data off of the block before it can be erased. All of those writes drastically increase the write amplification so typically garbage collection algorithms try to avoid blocks with lots of valid data. However, this can lead to a situation where some blocks in the array start to be erased more than others simply because they

contain the smallest amount of static data. The resulting imbalance in wear-out across the different blocks in the array can then lead to faster device failure. To address this problem, static wear leveling algorithms periodically move valid static data to a different location in the flash array opening up the less worn, previously static block to new data. These static wear leveling algorithms can increase the write amplification because they do involve moving data around but they can ultimately greatly extend the life of the device.

3.2.4 Cache Register Operations

Another method of improving the performance of NAND flash memory devices involves including a second buffer register in the die that allows for pipelining read or write commands to the same die. Initially flash devices were equipped only with a single buffer register that would hold data while it was being sent back to the memory controller or written into the actual memory array. This was necessary because flash reads and writes are done at a 4KB granularity but the flash device itself has only an 8bit wide interface. So, the data needed to be stored somewhere while the operations were taking place. Adding a second register allowed for an additional operation to take place while the transfer was occurring. These two registers are called the cache register and the data register. During a read operation the data is transfered from the NAND flash array to the data register. After the process of getting the data off of the array is complete, the page can then be quickly transfered from the data register to the cache register provided that the cache register isn't

already holding something. Then another read process can take place on the array and that page can be stored in the data register while the other page is being sent back to the memory controller from the cache register. In this way it is possible to completely hide the transfer latency of some operations thereby reducing the overall average access latency. Initially this pipelining of operations using the cache and data registers was only possible if the addresses being accessed were sequential. [52] However, it was later expanded to support random addresses as well [39].

3.3 PCM

3.3.1 Organization

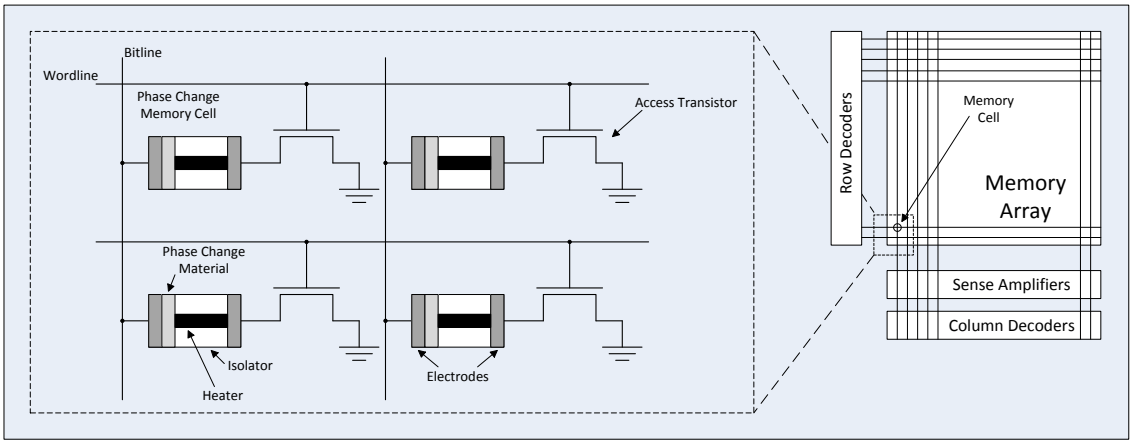

Figure 3.8: The proposed structure of a PCM based memory array.

Phase change memory (PCM) is an emerging non-volatile memory technology that boasts better latencies and lifetime than NAND flash. Figure 3.8 shows a typical PCM cell which is composed of a small amount of chalcogenide glass which

is connected to the bit line, a heating element, and an access transistor, which is connected to the word line. Like DRAM, PCM has a separate access transistor for each storage unit rather than strings of storage units like NAND. It works by changing the state of the chalcogenide glass between two stable phases: amorphous and crystalline. These two different phases of the material have very different resistances which can be used to represent a 1 (crystalline) or a 0 (amorphous).

Because these devices have not been widely produce or used to build memory systems, their system level organization is still somewhat uncertain. However, most architecture studies that focus on incorporating PCM into the main memory system assume a DRAM-like organization for the PCM portion of the memory system [25, 28].

3.3.2 Access Process

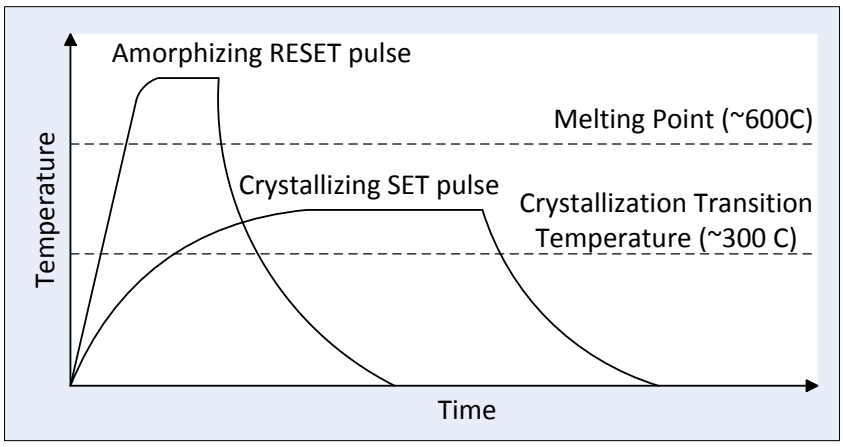

Figure 3.9: The different processes involved with writing a 1 or a 0 using PCM.

To place the material in its amorphous state it is heated to a high temperature

which melts the material and then it is quenched quickly. Alternatively, to place the material in its crystalline state it is heated to a temperature between the crystallizing temperature and the melting temperature for a longer period of time. This behavior can be seen in Figure 3.9. Therefore, it takes much longer to write a 0 into a PCM cell than it does to write a 1. To create the heat needed for the phase changes voltage is applied to the heater, which is generally just a resistor, via the bit line. A shorter, higher amplitude voltage pulse is used to achieve the melting temperature and rapid quenching needed for the amorphous state. Similarly, a longer, lower amplitude voltage pulse is used to create the longer lasting crystallizing temperature needed for the crystalline state. Applying a voltage on the appropriate word line opens the desired cell's pass transistor causing the current resulting from the voltage pulse to flow only into that cell. In this way, individual cells can be programmed. Because the crystallization process is a gradual one PCM, like flash, is potentially capable of MLC by varying the degree of crystallization of the chalcogenide material. This creates different levels of resistance which can be used to represent different multi-bit combinations. So, PCM is still capable of potentially high densities despite its need for more pass transistors than NAND. However, as was the case with flash, such MLC cells require either more complicated sense amplifiers or longer sensing times. [25, 28, 29]

The state of a PCM cell is determined by a charge rundown process that is very similar to the one used to read NAND flash. First a capacitor is charged to a specific voltage and then it is allowed to discharge through a bit line. The voltage on the word line of the desired cell is driven high which turns on the pass transistor

and connects the cell to the bit line allowing the current to flow through it. At the end of a specified period of time, the charge on the capacitor is compared to some reference voltage. If the PCM cell was in the higher resistance amorphous state then less current would have flowed and more charge would remain on the capacitor giving it a higher voltage. Conversely, if the cell was in the lower resistance crystalline state then much more current would have flowed and little charge would remain on the capacitor. Therefore, if the capacitor's voltage is above the reference voltage than the cell was in the amorphous state and in the crystalline state otherwise. Because the method of sensing is essentially the same as NAND flash its seems strange that PCM has much better read latencies. This is because PCM uses dedicated sensing capacitors which have a much smaller capacitance than a NAND flash bit line and therefore discharge much faster. In addition, because the current must only flow through a single cell the total resistance in the bit line is much lower during a read than is the case with NAND flash and so more current can flow [28,53]. This further increases the speed at which the capacitor can discharge and reduces the overall latency of the read operation. In addition, some PCM systems use current comparing sensing circuits rather than the voltage comparing sensing circuits described thus far [28]. These are even larger but are capable of even faster sensing operations. It should be noted that those sensing circuits could also be used to determine the current through a NAND flash cell. [28, 54]

When the PCM cell is heated the chalcogenide material expands and subsequently contracts as it cools. The repeated expansion and contraction of the material as a result of writing data into the cell degrades the contact of the electrode with the

glass. Eventually this contact is degraded to the point that it can no longer reliably conduct the programming currents into the cell. This means that, like flash, PCM cells are only capable of performing a finite number of reads before they cease to work. While many sources state that this write endurance is orders of magnitude greater than flash, wear leveling is still employed in order to maximize the lifetime of PCM, especially when PCM is used as a main memory [25, 28, 29, 55]. Though it is not as necessary as it is with NAND flash, wear-leveling still ensures that the PCM chip's lifetime will be measured in years even if it is written to much more frequently than is anticipated. [55]

3.3.3 Difficulties

There are always many hurdles that have to be overcome when developing a new memory technology. Two of the most prominent challenges facing that development of PCM have to do with its power consumption and resistance drift. The write process of PCM requires high current density over a long time period. Therefore, in order to keep the power consumption of the PCM device at reasonable levels, the number of simultaneous writes that can be performed has to be strictly limited. This negatively affects the potential write performance of PCM devices. Furthermore, the high power consumption of PCM devices can potentially increase the temperature of the device to the point that logical errors or incomplete phase transitions occur. So, keeping the power consumption of the PCM device in check is critically important to its proper operation and its success as a memory technology. [56]

Another difficulty currently facing PCM development has to do with the resistance properties of the chalcogenide material. Ideally, PCM would be able to store multiple bits per cell in the similar way to MLC NAND flash by assigning different levels of resistance to different combinations of the bits. However, the resistance of the chalcogenide material tends to drift for a while after it has been written to. This results in states that had resistances that were initially quite different eventually developing resistances that are less distinct. Over time it would then become more difficult to tell the different states apart. This property of the chalcogenide material is somewhat understood though and researchers have some solutions to potentially solve the problem. [57]

3.4 Memristor

3.4.1 Organization

The existence of the memristor was first suggested by Leon Chua in the 1970s [58]. It is characterized as a the fourth fundamental 2-port circuit element where the other three are the resistor, the capacitor and the inductor. The resistance of the memristor changes as a result of the magnitude of voltage that is applied across it. Therefore, the current that flows through a memristor as a result of a voltage being applied across it depends not only on the present voltage but also on the voltage that was applied during the previous programming cycle. This results in the hysteresis curve in Figure 3.11 where a different resistance can be seen for the same voltage depending on whether the previous voltage passed the write threshold

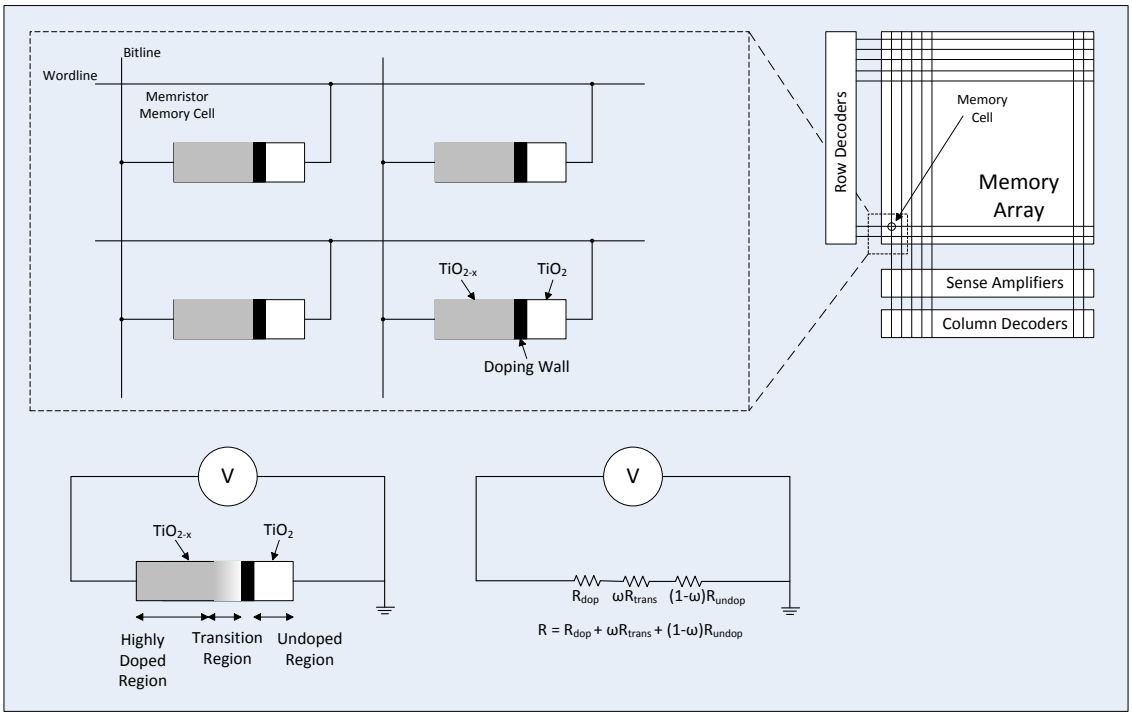

Figure 3.10: The proposed structure of a Memristor based memory array. Note the lack of access transistors.

(Vth1) or the erase threshold (Vth2). Therefore, the memristor can be used to store a 1 or a 0 as high or low resistances which can be changed by applying different voltages across the memristor. [59]

There are many ways of fabricating memristors however the TiO2 version pictured in Figure 3.10 is one of the most common. This cell primarily consists of two sections of TiO2: the upper, conductive section has been doped with oxygen vacancies while the lower, resistive section is normal TiO2. The cell works by shifting the concentration of oxygen ions in the TiO2 material. When a negative voltage is applied to undoped side of the memristor it attracts the oxygen ions in the doped TiO2 thereby expanding the lower resistance region of the material. As a result,

the resistance of the entire cell decreases. Similarly, if a positive voltage is applied to the undoped side of the memristor, the oxygen vacancies will be repelled thereby increasing the more resistive region of the material. Also, because the oxygen vacancies are able to move relatively quickly through the TiO2 material the write latency for memristors has the potential to be very small. In addition, the memristor can take on a range of resistance values depending on the voltages that are applied to it. Therefore, the memristor, like flash and PCM, is also potentially capable of MLC. [60, 61]

Like PCM, the Memristor technology is still relatively new and has not yet been implemented in any widely available devices. As a result, it is difficult to say what its eventual system level organization will look like. However, like PCM, most researchers appear to assume that it will be organized similarly to DRAM.

3.4.2 Access Process

The memristor memory cell is composed of only a single two port circuit element. As a result, it can easily be incorporated into a very high density memory array structure called the crossbar array. This simple structure, which is pictured in Figure 3.10, is capable of much higher densities than other memory arrays because it does not use pass transistors. This effectively doubles the possible density of the array because half the hardware is needed for each cell compared to a typical PCM or DRAM cell. Instead, the proper voltages are simply applied on the desired row and column wires of the crossbar. For instance, to put the memristor element

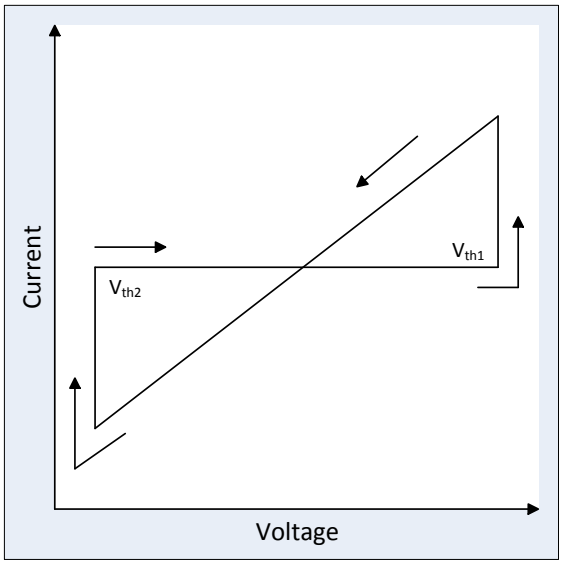

Figure 3.11: The cyclical relationship between voltage and resistance in a memristor.

into its lower resistance mode a positive voltage is applied to the row wire (which corresponds to the doped side of the memristor in this example) while the column wire is grounded. This causes current to flow through the memristor that resides where the row and column wires intersect thereby changing its state. To prevent changing the state of the other memristors, the other column wires are also set to the positive voltage which counteracts the effect of the row wire's voltage. To read a memristor, current is injected into the desired row and measured at the desired column. Because there are alternate paths for the current to take, the operation is performed three times. The first time the initial state of the cell is read, then it is explicitly programmed to a 1 and the current is remeasured, then it is explicitly programmed to a 0 and the current is read a third time. By comparing the three currents it is possible to determine the state of the cell being read. Since this

memory also uses current to determine whether a cell is in a high or low resistance state it is assumed that its sensing circuits will be very similar to ones used with PCM. [59, 62]

3.4.3 Difficulties

One of the most significant challenges currently facing the memristor's development is actually not directly related to the memory technology itself but instead to its crossbar array architecture. The crossbar array attempts to determine the resistance of the memristor at the intersection of a row and column line by biasing the other wires in the array. This should isolate that resistance so that all of the current flowing through the array will go through that element. However, in practice what often happens is that sneak paths form within the array that allow some current to travel through other memristors, sometimes avoiding the desired memristor entirely. When this happens, the resulting resistance measurement can be significantly off and the resulting data corrupt. [63] Therefore, in order to reliably utilize a crossbar array, the problem of sneak paths must first be dealt with. Researchers are currently working to address this both to enable the continued development of memristor memories and also to allow other potential technologies to utilize crossbar array architectures.

	SRAM		DDR2 DRAM		DDR3 DRAM	
Cell Elements	6T		1T1C		1T1C	
Density	-		2Gb	[2]	8Gb	[2]
Read time (ns)	1		55	[?, 64]	50	[44, 45, 64]
Write/Erase time (ns)	1		55	[?, 64]	50	[44, 45, 64]
Read op. voltage (V)	-*		1.8	[?, 64]	1.35-1.5	[44, 45, 64]
Write op. voltage (V)	-*		1.8	[?, 64]	1.35-1.5	[44, 45, 64]
Write Endurance	10^{16}		10^{16}	[46]	10^{16}	[46]
	LPDDR2		RLDRAM			
Cell Elements	1T1C		1T1C			
Density	4Gb	[2]	576Mb	[2]		
Read time (ns)	60	[2]	10	[2]		
Write/Erase time (ns)	60	[2]	10	[2]		
Read op. voltage (V)	1.14 - 1.95	[2]	1.35	[2]		
Write op. voltage (V)	1.14 - 1.95	[2]	1.35	[2]		
Write Endurance	10^{16}	[46]	10^{16}	[46]		
	NOR Flash		PCM		Memristor	
Cell Elements	1T		1T1R		1R	
Density	2Gb	[2]	???		???	
Read time (ns)	100	[2, 65, 66]	$\sim$ 60 - 200	[28, 46, 67, 68]	$\sim$ 50	[69, 70]
Write/Erase time (ns)	1-2us(word)/10-900ms(block)	[2, 65, 66]	$\sim$ 60 / 300	[28, 46, 67, 71]	$\sim$ 50-250	[30, 70]
Read op. voltage (V)	1.7-2	[2, 65, 66]	$\sim$ 1-1.8-3	[28, 46, 67]	$\sim$ 1-2.5-3	[60, 62]
Write op. voltage (V)	1.7-2	[2, 65, 66]	$\sim$ 1.6-3	[28, 46, 71]	$\sim$ 3-3.44	[60, 62]
Write Endurance	10^5	[2, 65, 66]	$\sim 10^4$ - 10^9	[28, 46, 67, 71, 72]	???	
	SLC NAND Flash		MLC NAND Flash		TLC NAND Flash	
Cell Elements	1T		1T		1T	
Density	512Gb	[2]	2Tb [2]	2.5Tb		
Read time (ns)	25us	[73, 74]	50us	[?, 75]	100us	[76]
Write/Erase time (ns)	200us (page) / 2ms (block)	[73, 74]	900us (page) / 2ms (block)	[?, 75]	2.5ms (page) / $\sim$ 2ms (block)	[76]
Read op. voltage (V)	2.7-3.6	[46, 73, 74]	2.7-3.6	[2, 75]	3.3	[2]
Write op. voltage (V)	2.7-3.6	[46, 73, 74]	2.7-3.6	[2, 75]	3.3	[2]
Write Endurance	10^5	[73, 74]	10^4	[2, 75]	2,500	

Table 3.2: Comparison of Memory Technologies.

3.5 Technology Comparison

Table 3.2 presents a summary of the different main memory technologies discussed in this section and their characteristics with regard to speed, density and power usage. From these values it is quickly clear that the non-volatile technologies

are generally not equal to DRAM in terms of latency performance. The excitement surrounding the newer technologies is understandable considering their stark improvement over NAND flash's latencies. However, the low latency numbers for those technologies are the result of carefully fabricated prototype devices. In the case of PCM, recent mass production attempts of the technology have yielded less impressive performance figures. This can be seen in the wide range of values for the PCM latencies, particularly the erase latency. It is prudent then to assume that the longer latency figures for these technologies are a more accurate representation of their eventual performance. With this consideration in mind it becomes clear that if a memory system which incorporates non-volatile technologies is to perform as well as DRAM only systems, some additional engineering is required to hide the longer latencies. On the other hand, the motivation behind attempting to use these technologies despite their less than optimal latencies is evident in their higher density characteristics. Such characteristics may enable the construction of much larger main memory systems than would be possible with only DRAM thereby providing for the greater memory needs of future applications.

In addition, all of the non-volatile technologies that have been discussed in this section share many similarities. For instance, they all work on the same basic principle: their storage cells resist current flow by varying amounts depending on their state. Therefore, all of the non-volatile technologies can also use the same sensing circuits. Similarly, organizational decisions have similar affects on the different technologies. The more sensing circuits that are included on a chip, the more bit lines that can be read simultaneously and the longer the burst of data from the

device. Also, the devices all have asymmetric read and write operations. The significantly longer latency costs of a write compared to a read mean that scheduling is much more important for all of these technologies. Writing at an inopportune time can result in much more significant delays than result from incorrectly scheduled writes in a DRAM system. Finally, all of these technologies also have finite lifetimes and wear out from over use. Therefore, they all require some form of wear leveling scheme. These similarities are the motivation behind the notion that all non-volatile technologies can be represented by a generic non-volatile memory in memory system studies. The similarities between the technologies mean that many of the modifications needed to incorporate the memories into the main memory system will be appropriate for all of the non-volatile memory types regardless of their underlying cell technology.

Chapter 4: Multi-Level Main Memory Systems

The capacity, power, lifetime, and performance limitations of the technologies that could potentially fill the main memory roll has led designers and researchers to propose the division of the main memory sub-system into a new hierarchy within the broader memory and storage system hierarchy. These architectures typically divide the main memory system into a smaller, faster, more expensive cache portion and a bigger, slower, less expensive backing store. In this way different technologies and organizations can be utilized at the different levels in order to construct a main memory system with characteristics that are not possible with a single memory technology. This chapter will discuss the different varieties of these new memory systems and will also compare them to the current state of the art system architecture.

4.1 Multi-Level Main Memory Organization

The extension of the memory system to enable the incorporation of new, slower main memory technologies has resulted in a new cache layer. There are potentially two ways to think about this new cache layer. In one sense it represents an extension of the cache hierarchy as it is also functionally a cache. In another sense, it is possible to consider it a new level of the main memory because it is off-die and is

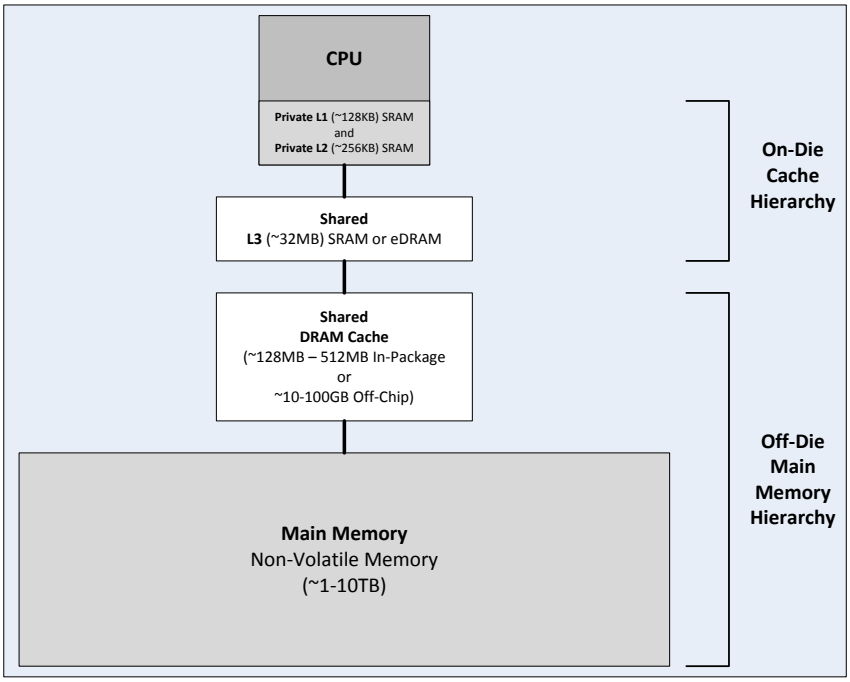

Figure 4.1: The typical multi-level main memory organization that will be discussed in this work

often managed by the main memory controller. For the purposes of this dissertation, we tend to view the new cache layer in the latter way, as a new layer of the main memory system. Figure 4.1 illustrates our view of the new system architecture with a multi-level main memory system.

In addition, multi-level main memory systems can be constructed so that the layers of the main memory are utilized in different ways or composed of different memory technologies. The resulting architectures can be hierarchical or independent, heterogeneous or homogeneous, and software managed or hardware managed. A hierarchical system uses some layers of the main memory as cache to store the most frequently accessed subset of the data that is contained in the lower levels

of the system. Alternatively, it is also possible to distribute the data across the different levels of the system so that each level is assigned a different portion of the address space rather than having one act as a cache. In this dissertation we will focus on the multi-level systems that form a clear hierarchy with their layers.

4.1.1 Heterogeneous Main Memory Systems

Heterogeneous main memory systems are another type of main memory system that utilize two or more different memory technologies to build the main memory system. We consider these architectures to be another distinct subset of multi-level main memories because they do not have to form a hierarchy and can instead use the two memories in a side by side address space configuration. In this dissertation we will refer to memory systems that are both heterogeneous and hierarchical as hybrid memory systems. This differentiates those systems from hierarchical systems which use the same memory technology at both levels (varying the organization instead) and from heterogeneous systems which do not use one of the memories as a cache for the others.

A great deal of research into heterogeneous memory systems has focused on different ways of incorporating non-volatile memory into the main memory system. The first of these systems to be proposed was eNVy in 1994, which utilized NOR Flash as its backing store technology and DRAM as its cache [77]. Later, several different architectures were proposed that utilized PCM as the backing store [25, 28].

More recently, Chatterjee et al. proposed constructing heterogeneous memory

systems utilizing different types of specialized DRAM [27]. This work utilized RL-DRAM as the cache technology and LPDDR as the backing store technology similar to one of the architectures presented in Chapter 8 .

4.2 Software versus Hardware Management

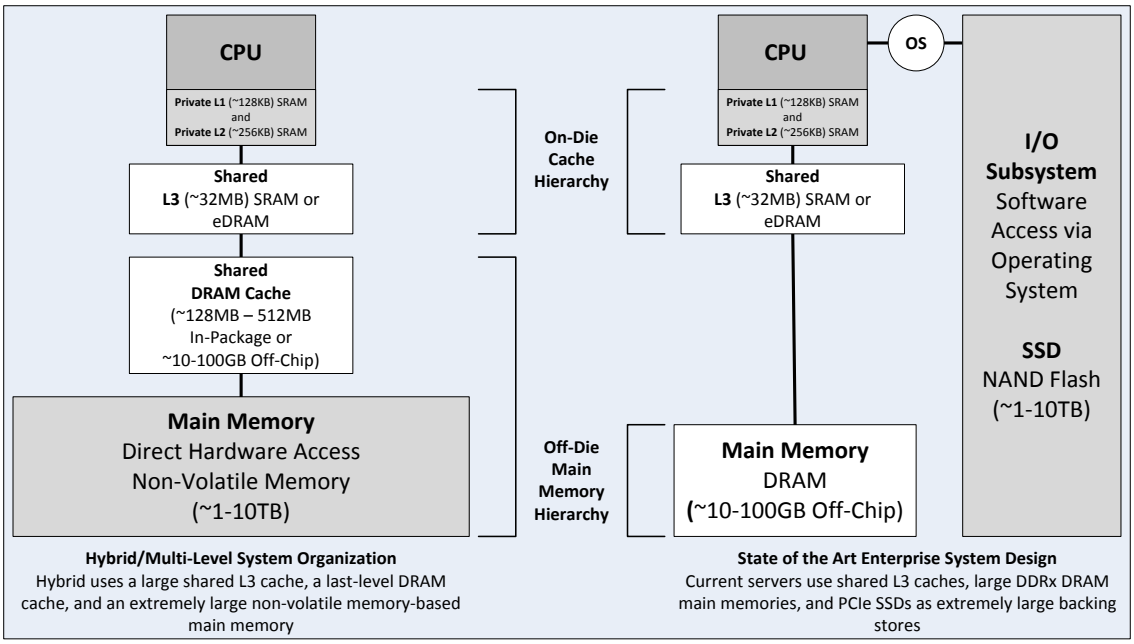

Figure 4.2: Hybrid organization versus a typical enterprise-class SSD organizataion

In addition to heterogeneous, hybrid, and hierarchical multi-level main memories, there are also architectures that utilize either software or hardware to manage the cache layer of the main memory. To understand the differences between hardware and software managed multi-level main memory architectures, we compare a hardware managed Hybrid architecture to a software managed architecture. The software managed architecture used in this comparison is representative of a modern

Table 4.1: Hybrid Memory vs. SSD Comparison

	Hybrid	**SSD**
Page Placement	CPU memory controller	OS - virtual memory/buffer cache
Garbage Collection (where necessary)	NV controller chip	NV controller chip
Virtual to Physical Address Translation	OS - virtual memory	OS - virtual memory
Physical to Backing Store Address Translation	CPU memory controller	OS - block layer
Backing Store access scheduling	backing store controller chip	OS - I/O scheduler
Host Interface	direct connection to CPU memory controller	PCIe root complex
File System	tmpfs ramdisk	ext3

enterprise-class system which utilizes a high performance PCIe SSD. The Hybrid architecture used in this comparison resembles the Flash Hybrid architecture proposed in [31] and the PCM Hybrid architecture proposed in [25]. This architecture extends the functionality of the memory controller to manage the DRAM main memory as a cache for a large backing store. In both systems, the backing store hardware is largely identical to the hardware of an SSD. The key difference is that the memory controller, rather than the operating system, is responsible for generating requests and managing whether a page is stored in the DRAM cache or the backing store. Figure 4.2 illustrates the differences between the SSD and Hybrid approaches.

This rest of this section will describe the key design differences of the Hybrid and SSD systems by comparing the Hybrid design to the design of a current state-of-the-art enterprise PCIe SSD. These differences are summarized in Table 4.1.

4.2.1 Stalling versus Interrupting

The storage system was designed for long latency spinning disks so one of the key concerns was hiding disk latency. This was accomplished on time-shared systems

by having the OS scheduler switch to another task and using an I/O interrupt to indicate completion of the access. The process waiting on the I/O operation would later be awoken by the OS scheduler to resume execution. Modern systems have inherited this feature and it still makes sense for most types of I/O operations. However, as with high speed networks, when the hardware is capable of low enough latency or high enough throughput, interrupting can introduce unnecessary overhead in the system. This overhead can come from several possible sources including the time it takes to service an interrupt, the time it takes to perform a context switch to another task, the time it takes the OS scheduler to resume execution of the waiting task, and indirect costs such as cache pollution. At the time of this work, the typical design for a modern SSD still relies on the interrupt and task switch approach.

Since backing store accesses are hidden from the OS in the Hybrid memory design, the interrupt and task switch process is replaced with a simpler stalling process. This is similar to how the CPU cores just wait on the memory controller to perform DRAM accesses. While this may seem non-intuitive due to the significantly longer latencies of some backing store technologies such as flash, if the overhead of the interrupt and task switch is sufficiently large, then stalling can offer better performance. For instance, the work in [78] indicated that the delay incurred by a context switch can be as much as 1.5 ms, which is considerably longer than most flash read latencies. Therefore, in some situations a system could potentially benefit greatly from stalling on a read access to the backing store rather than task switching and waiting for an interrupt.

4.2.2 Page Placement

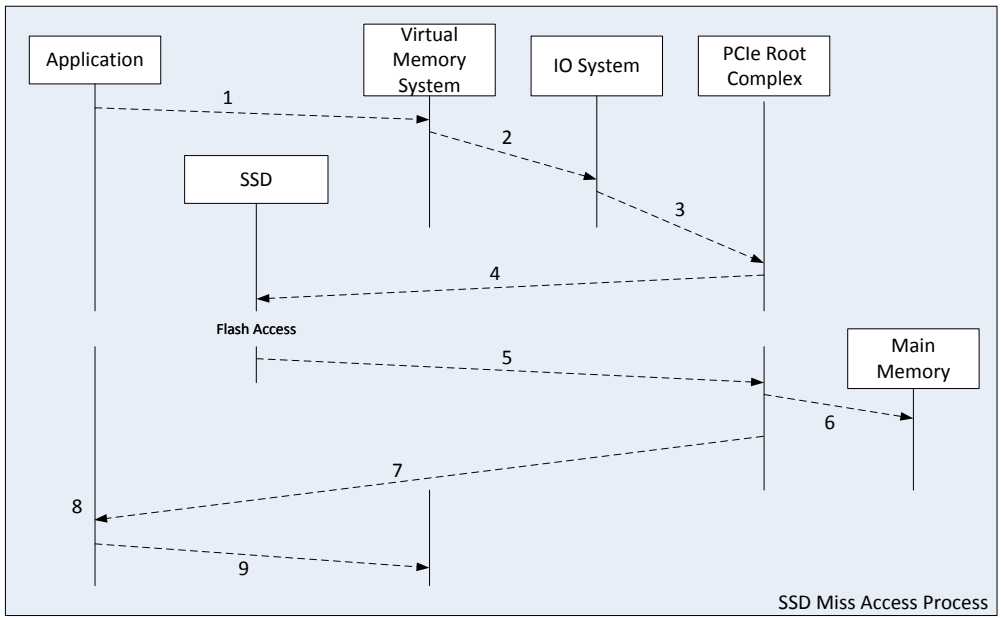

Figure 4.3: The steps involved in servicing a miss of the DRAM for the SSD organization.

Currently, the OS virtual memory system determines which virtual pages are kept in the main memory page frames and which virtual pages are stored on the SSD backing store (either the original file for file-backed page or the swap for anonymous virtual pages). This process is described in Figure 4.3. Step 1 of this diagram starts with the application generating a request to the virtual memory system. Step 2 occurs on a page miss; here the virtual memory system selects and evicts a virtual page from the main memory. The virtual memory system also passes the requested virtual page to the I/O system. During step 3 the I/O system generates a request for the SSD. This request is then sent to the PCIe root complex which directs it

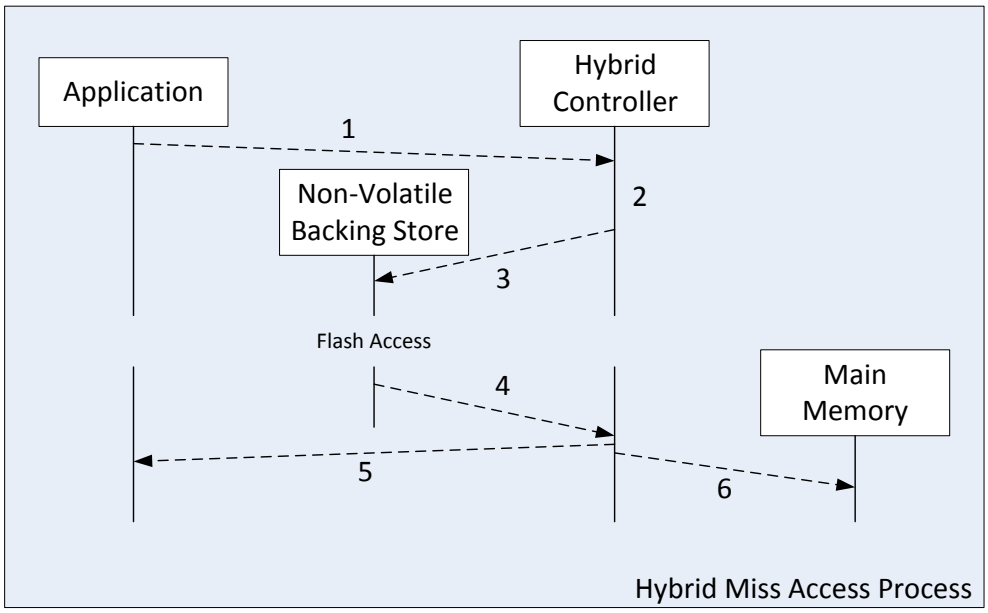

Figure 4.4: The steps involved in servicing a miss of the DRAM for the Hybrid organization.

to the SSD in step 4. The PCIe root complex is the host bridge that connects the CPU and memory to the PCIe system. More information on the PCIe system and its architecture can be found at [79] and in the official PCIe specifications at [80]. To specify which virtual page to bring in from the SSD, the OS sends the SSD controller a logical block address. The SSD then uses that logical block address to determine the actual physical location of the virtual page associated with that address and issues a request to the device which contains that virtual page. For the virtual page that is evicted from the main memory, the SSD allocates a new physical page for that virtual page and issues a write to the appropriate device. This occurs between steps 4 and 5. After the SSD handles the request, it sends the data back to the CPU via the PCIe root complex, step 5. The PCIe root complex the passes

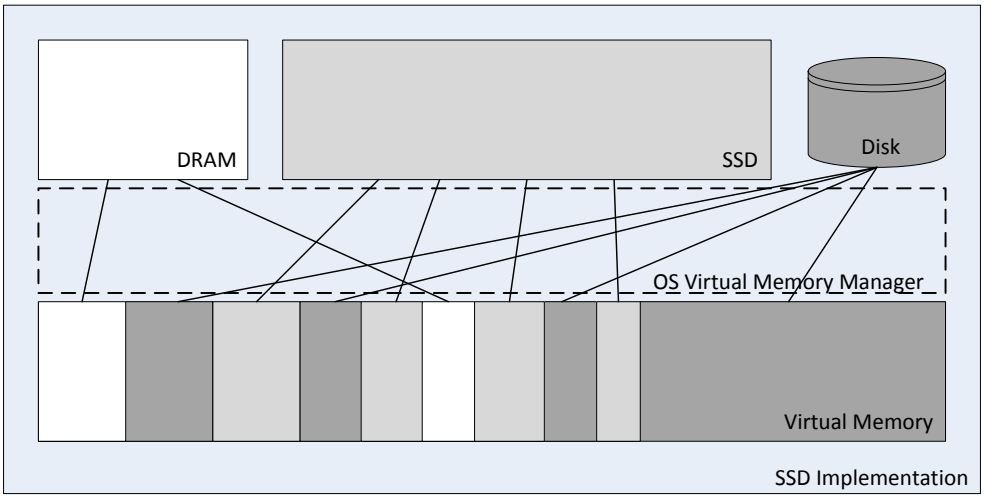

Figure 4.5: The division of the various address spaces involved in the SSD organization.

the data to the main memory system where it is written, step 6. Once the write is complete the PCIe root complex raises an interrupt alerting the OS scheduler that an application's request is complete. This is step 7. Finally, during step 8, the application resumes, reissues its request to the virtual memory system and generates a page hit for the data. The division of the virtual memory address space between the different physical address spaces in SSD-based systems is illustrated in Figure 4.5.

In the Hybrid system, the backing store is presented to the OS virtual memory manager as the entire physical memory address space. This address space organization is presented in Figure 4.6. It appears to the OS that the computer's main memory is the size of the backing store. The actual DRAM main memory address space is hidden from the OS and is managed by the memory controller as a cache.

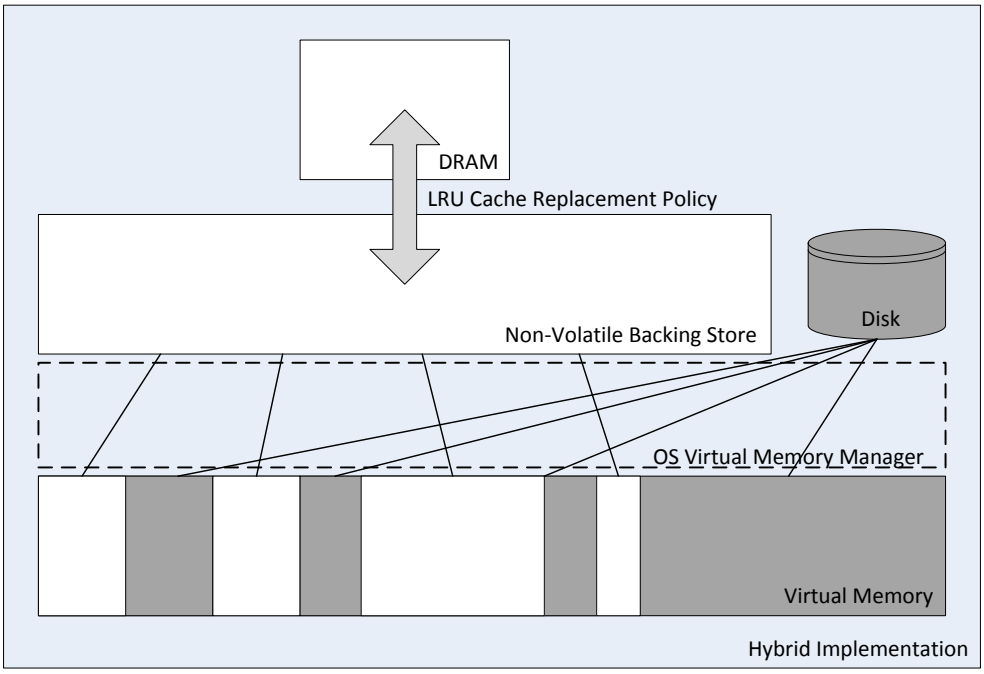

Figure 4.6: The division of the various address spaces involved in the Hybrid organization.

Together the backing store and DRAM cache form the Hybrid memory. Accesses to this Hybrid memory have a granularity of 64 bytes, just like DRAM. The cache lines in the DRAM cache have a granularity of 4KB because that is the typical size of an OS virtual memory page. Keeping the fill granularity the same in both the SSD and Hybrid systems removes a possible source of confusion from the results presented later in this dissertation. Figure 4.4. shows the access process for the Hybrid architecture which is considerably simpler than the SSD process. Step 1 begins with the application generating a request. Step 2, varies somewhat depending on the degree of associativity implemented in the Hybrid system. If the system is using a direct mapped cache like the one proposed in [81] then the appropriate location

in the DRAM is determined from the address and accessed. The tag is retrieved with the data in a single accesses. In this system a DRAM cache access begins with the Hybrid memory controller checking its tag database to determine if a particular cache line is present in the DRAM cache. If the cache line is present in the DRAM cache, then the access is serviced by the DRAM as a normal main memory access (not shown in Figure 4.4.). When an access misses the DRAM cache, the Hybrid controller selects a page in the DRAM to evict and schedules a write-back if the page is dirty. In the current implementation of our Hybrid memory controller, a least recently used (LRU) algorithm is used to determine which page to evict. The missed page is then read in from the backing store and placed in the DRAM. This is step 3. During this step, the Hybrid memory controller can also prefetch additional pages into the DRAM or write back cold dirty pages preemptively, similar to how the virtual memory memory works, to further improve read performance. Currently, the Hybrid system implements sequential prefetching. More complex prefetching schemes such as stream buffers, stride prefetching, and application directed prefetching are also compatible with this design. Step 4 is the backing store handling the request. Once the data has been received from the backing store the Hybrid controller passes the data to the application, step 5. Finally, during step 6 the data is written into DRAM from the Hybrid memory controller.

4.2.3 Associativity

The page table utilized by the OS is functionally fully associative and so, in order to provide a more fair comparison, the cache in our Hybrid system is 16 way set-associative. We also analyze the effects of different levels of associativity on Hybrid performance later in the study to determine the role that associativity plays in the performance of both systems.

In order to implement associativity efficiently a Hybrid main memory design must store the tags for its large cache in an efficient manner. The tag size is computed as b - c + a + s, where b is the number of bits in the backing store address space, c is the number of bits in cache address space, a is the number of bits needed to represent the cache associativity, and t is the number of bits needed for state including the valid bit, the dirty bit, replacement policy state bits, and other related data. For sufficiently large Hybrid memories, the tag store can become too large in terms of transistor budget to be implemented directly on the CPU. There have been several solutions that have been proposed to solve this problem of tag overhead in DRAM caches. These range from adding a tag cache to temporarily store tags [82], storing tags alongside data in DRAM [83] and implementing the DRAM cache as direct mapped [81]. To keep matters as straightforward as possible in the experiments presented in Chapter 6, we assume that the tags of the associative Hybrid implementations are stored in SRAM and can be accessed efficiently. Such a system could be realized with either a dedicated tag store or a tag cache in a real world implementation.

4.2.4 Prefetching

The operating system prefetches data from the backing store on a page fill in a effort to reduce future misses and to amortize the cost of the backing store access. To keep the comparison as fair as possible we have implemented a sequential prefetcher as part of the Hybrid design. On a cache fill this prefetcher grabs the next 16 pages after the missed address in addition to the missed page itself. We arrived at the 16 page prefetching degree as a result of experimentation that is included in Chapter 6 of this dissertation. More advanced prefetching schemes are another area of potential future work that the researchers would like to explore.

4.3 Software Managed Approaches

4.3.1 Polling SSDs

There have been a number of projects that have modified the software interface for solid state drives by polling the disk controller rather than utilizing an IO interrupt to indicate when a request completes [84] [85] [86]. This is similar to the Hybrid architecture presented in this chapter in that the memory controller and the application poll when a request is outstanding to the backing store. The key differences between the two approaches are that the software based designs still utilize the same PCIe interface and basic operating system structures as current PCIe SSD designs. The performance advantages or limitations that IO polling would have compared to the Hybrid architecture approach are an open area of study and

hopefully a subject for future work.

4.3.2 Persistent Object Stores

Another way to redesign the OS to work more efficiently with SSDs is to build persistent object stores. These designs require the programmer to determine which objects should be persistent and to modify the code to utilize special allocation functions. A portion of the DRAM main memory is then allocated as a cache for the persistent objects. These designs require careful management at the user and/or system level to prevent potential problems such as dangling pointers and to deal with allocation, garbage collection, and other management issues. A performance improvement for certain types of workloads is possible with these designs by creating a customized caching algorithm for persistent objects to be more efficient than the generic operating system paging mechanism. An example of this type of system is SSDAlloc [34], which builds persistent objects for boosting the performance of flash-based SSDs, particularly the high end PCIe Fusion-IO drives. NV-Heaps [33] is another similar system designed to work with upcoming byte-addressable non-volatile memories such as PCM.

4.3.3 Specialized File Systems

Other work focuses on file system approaches for managing non-volatile memory. One example is a file system that has been developed for managing Hybrid main memories [35]. This work is based on the assumption that the OS rather than the

memory controller handles tasks such as page placement, garbage collection, and wear leveling. Another proposed file system uses a technique called short-circuit shadow paging to provide functionality that is optimized for byte-addressable and low latency non-volatile memories (e.g. PCM) [87].

4.4 Hardware Managed Approaches

4.4.1 PCM Based Systems

Over the past few years, a significant amount of work has also been put into designing architectures that can effectively use PCM to replace or reduce that amount of DRAM needed by systems [25] [28] [26]. This body of work anticipates a slow down in the scaling of DRAM and proposes PCM based systems as a way to continue increasing the capacity of main memory to meet demand. Some of the architectures that have been suggested for use with PCM inspired the Hybrid architecture studied in this dissertation in that they also utilize the DRAM as a cache that is managed by the memory controller [25].

4.4.2 Flash Based Systems

However, these PCM designs were not the first to utilize a Hybrid architecture. In 1994, eNVy was proposed as a way to increase the size of the main memory by pairing a NOR flash backing store with a DRAM cache [77]. This design is actually very similar to both the Hybrid architecture studied in this paper and the Hybrid PCM architectures except that it utilizes NOR flash as its non-volatile backing store

technology. In addition, a very similar architecture was also proposed by FlashCache which utilized a small DRAM caching a larger NAND flash system [88]. However, it is engineered to focus on low power consumption and to act as a file system buffer cache for web servers, which means the performance requirements are significantly different than the more general purpose merged storage and memory system. In 2009, a follow-up paper to FlashCache proposed essentially the same design with the same goals using PCM [89].

4.4.3 Solutions from Industry

There have also been several industry solutions that attempt to improve the performance of the storage system [90] [91] [92] [93] [94]. These solutions tend to fall in one of three categories: software acceleration for SSDs, PCIe SSDs, and Non-Volatile DIMMs. Recently, several companies including Oracle have released software to improve the access times to SSDs by treating the SSD differently than a traditional hard disk [90]. Similarly, Samsung recently released a file system for use with its SSDs that takes into account factors such as garbage collection which can affect access latency and performance.

Also, for several years, companies such as Fusion IO, OCZ and Intel have been producing SSDs that utilize the PCIe bus for communication rather than the traditional SATA bus. The additional channel bandwidth provided by PCIe allows for much better overall system performance by alleviating one of the traditional storage system bottlenecks. The SSD design utilized in this study was based on

these products.

Finally, in 2008 Spansion proposed EcoRAM which was a flash based DRAM replacement [93] [95]. Like the Hybrid architecture, EcoRAM allowed the flash to interface directly with a special memory controller over the fast channel. However, EcoRAM utilized non-standard proprietary flash parts to construct its DIMMs and it was meant to replace DRAM. The Hybrid system presented in this work, on the other hand, can use standard NAND flash chips and utilizes the DRAM main memory as a cache.

4.5 Potential Use Cases

In this chapter we have discussed some of the potential differences between software and hardware management of the DRAM cache. These differences could result in the software or hardware approach being better suited to certain use cases. For instance, one potential benefit of software management that we have not discussed extensively until now is the flexibility of software. It is much easier to change the replacement or prefetching policy of a software managed cache than a hardware managed one. As a result, software managed DRAM caches might be better suited to situations where the workload characteristics change frequently. Also, the additional complexity of the prefetching policy in software managed caches could allow for much more accurate prefetches. However, the storage and management of the state required to implement such complex policies would probably be prohibitive in most cases. So, for systems that run a wide variety of workloads or workloads the

have extremely different phases of operation that change frequently, the benefits of this flexibility could outweigh the performance improvement provided by the low overhead of the hardware approach.

Furthermore, the ability to carefully tailor a workload or operating system to a particular memory system, as is sometimes the case in super computing environments, could also result in significant performance improvements for a software managed DRAM cache. However, this requires a considerable amount of effort on the part of the software developers who must have a detailed understanding of how to optimize the workload or operating system effectively. This is generally only the case in a very small number of specialized high performance computer applications where such optimizations are possible and worth while.

For all other systems, though, the low overhead hardware managed approach is probably the most beneficial implementation. The hardware managed approach provides near optimal performance for many workloads and does not require potentially complex and difficult modifications to either the user software or the operating system. Additionally, the hardware managed approach will almost certainly be preferable for systems with backing stores that are significantly faster than SLC flash, such as PCM. In those systems, even the slightest software overhead would represent a significant source of delay in handling misses to the DRAM cache. Therefore, developing and incorporating hardware managed DRAM caches into the memory hierarchy now is also beneficial because it will help to facilitate a more straightforward adoption of alternative backing store memory technologies in the future.

Chapter 5: Simulation Framework

By now it is hopefully clear to the reader that the potential design space for multi-level main memories is extremely large. In order to explore this design space we needed to develop a simulation infrastructure that would be open ended enough to enable the study of vastly different organizations comprised of technologies with indeterminate timings and uncertain access protocols. Furthermore, we needed an infrastructure that would provide detailed simulation of both the actual memory arrays and the cache controller that would manage page placement between the various levels of the hierarchy. We also needed to be able to accurately simulate the impact of software on the designs. This would allow us to compare our hardware based approaches to existing software based approaches. Finally, we also needed to accurately gauge the performance impact of the potential designs. To this end we developed an extensive suite of simulators to model the entire memory hierarchy, including a memory array simulator, a specialized cache controller simulator, and a PCIe root complex simulator. Furthermore, we incorporated our simulators into an existing full system simulator (MARSSx86) in order to capture the effects of software and to provide a complete feedback loop between the software, processor, and memory.

5.1 Open Memory Simulator

One of the primary contributions of this work was the development of the Open Memory Simulator (OMS). OMS is written in C++ and was originally developed as a NAND flash DIMM simulator. It is organized so that each logical component of the system is represented by a software object and its associated parameters in order to give the designer control over all aspects of the system. By setting these parameters in the simulator's ini file, the designer can determine almost all aspects of the systems overall structure and general timing characteristics. For example, the designer is able to specify the read/write/erase times, the number of channels in the system, the number of dies per channel, etc. This utilization of software objects also makes OMS modular and easy to customize as it is relatively easy to locate the code associated with different aspects of the system. The simulator includes a detailed model of all of the components of a typical NAND flash memory system including the Flash Translation Layer (FTL), the controller, and the memory devices themselves. Features such as dynamic wear leveling, garbage collection, and cache register operations are all accurately represented in the simulator. Furthermore, the simulator also includes a detailed channel model that captures the complex interactions that take place when multiple concurrent units (dies, ranks, etc) contend for the same bus. Often these interactions are prevented by memory access timings that ensure that no two units attempt to use the same channel at the same time. However, the development of tight memory access protocols is difficult, time consuming, and must often be repeated when other timing parameters are changed.

As a result, modeling and resolving bus contention in software rather than in the memory access protocol allows for a faster experimentation and prototyping process when investigating novel memory technologies that do not already have established protocols. Over time, the feature set of OMS was extended to enable it to simulate other non-volatile technologies such as PCM. This included adding a DRAM-like addressing mode as an alternative to the flash-style address translation and adding the option to disable garbage collection. All of these features combined allow OMS to easily simulate any non-volatile technology, even purely theoretical technologies with timings that did not resemble any existing devices.

However, as the work into multi-level memory systems progressed, it became apparent that alternative versions of DRAM, such as LPDDR and RLDRAM, would also be of interest. In order to be able to do that, though, several major changes needed to be made to OMS in order to allow for the accurate simulation of DRAM devices. One of the more complex changes that needed to be made had to do with the difference in access protocols used by DRAM and flash systems. DRAM systems, as we discussed in Chapter 3, use a fine grained access protocol that takes several commands to perform a single memory access operation. Flash systems, on the other hand, use a relatively simple protocol which only requires a single command to be issued for a memory access operation. The differences in these two access protocols can be seen in Figure 5.1. In general, the differences in access process do not affect much of the overall system usage; commands are issued, devices and channels are busy for some amount of time and then data returns. However, the complex DRAM access protocol allows for data to be returned at different times in the overall access

process. As can be seen in Figure 5.1, the data is sent back towards the end of an open page DRAM access while it comes back towards the middle of a closed page access. And both of these situations are different than a flash accesses where the data always comes back after the read access time is complete. Therefore, in order to develop a simulator that was capable of modeling all three kinds of memory access processes some method was needed to generalize the timings involved.

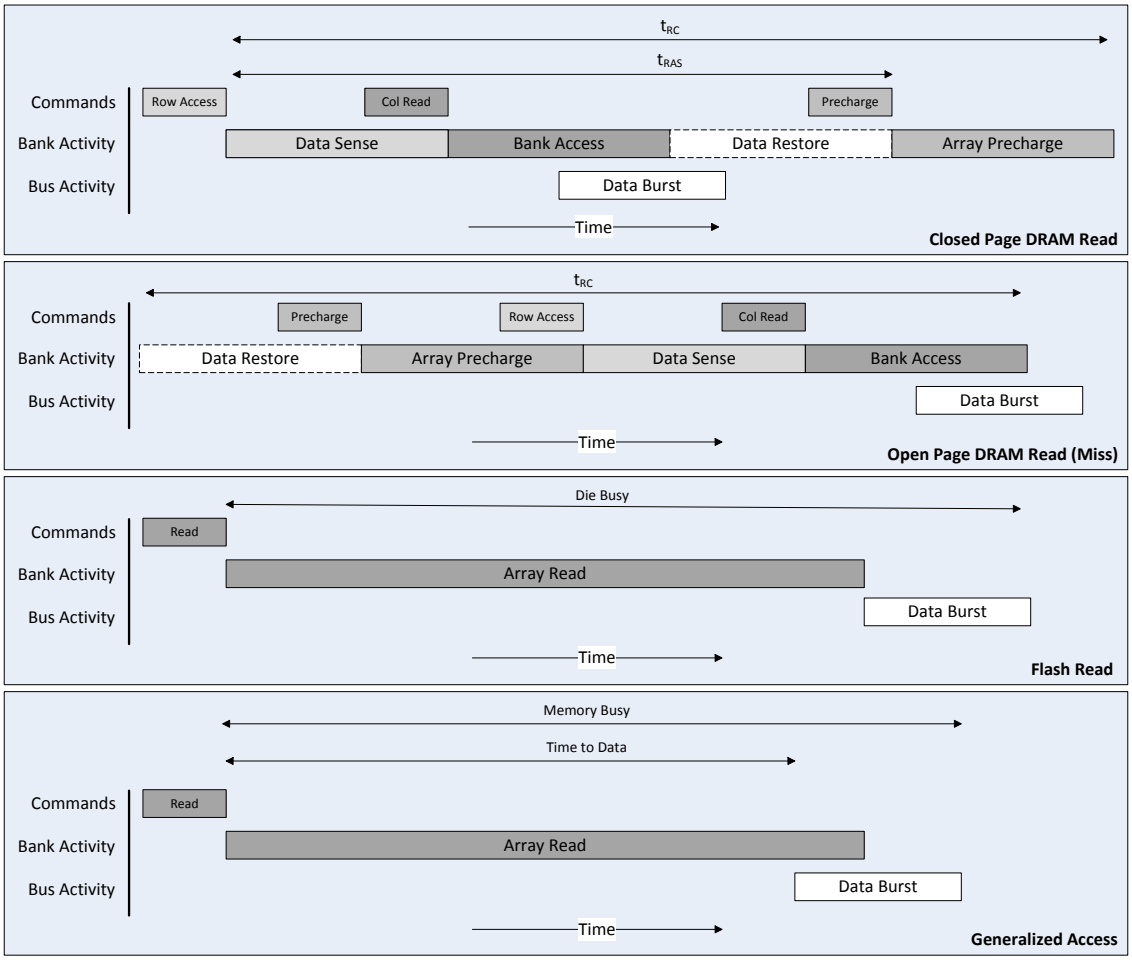

Figure 5.1: A comparison of different memory access timings).

The timing model that is used by OMS is pictured in Figure 5.1. This model

simplifies the access process of all memory technologies into two components: the time that the memory is busy and the time until data is returned. By specifying these two times, it is possible to simulate all three of the access processes that have been discussed thus far as well as any other processes that a designer might wish to simulate. By combining this simplified access model with its detailed channel simulation, OMS is able to generalize any memory technology by specifying significantly fewer parameters than are normally required to describe the timings of a typical DRAM system. As a result, OMS loses a small amount of accuracy in terms of command scheduling but still provides an accurate performance estimate overall because it models the resource contention aspects of the memory system that seem to dominate its behavior. We quantify this difference in accuraccy by comparing OMS to DRAMSim2.

Some additional modifications needed to be made to OMS to enable the simulation of DRAM-based technologies including refresh support, more complex device interfaces, and open page support. One of the biggest differences between DRAM-based devices and the non-volatile devices that OMS was developed to simulate is the need for refresh. Refresh support was added in the form of a new refresh command and logic in the controller to issue the refreshes to the appropriate units at the appropriate times. In addition, DRAM devices generally feature a separate command and address bus that is distinct from the data bus while flash devices utilize a single bus for all data, command and address data. Support for this more complex device interface was also added to the controller and device objects in OMS after it became clear that it could also have a notable impact on performance. Finally,

NAND flash generally does not include a row buffer in the same way that DRAM does and so initially OMS had no concept of an open page system that would enable faster accesses to the same row. In DRAM systems, though, sometimes open page or closed page approaches can have a significant impact on the performance of the system so we added some additional timing and row tracking code to the controller to model open page systems. With these changes OMS was able to simulate DRAM-like systems with an average absolute value difference of 8% compared to DRAMSim2, which has been hardware verified. Details regarding the verification of OMS against DRAMSim2 can be found in Appendix B.

5.2 HybridSim

Another critical portion of the system which required custom simulation was the cache controller that schedules operations between the DRAM cache and the backing store. Due to the novel nature of multi-level memory systems this controller simulation had to be specially developed. Like OMS, the controller simulator, Hybridsim, was written in C++ and features software objects and parameters for every significant logical aspect of the system. This allows the designer a great deal of freedom when investigating different multi-level system designs. For instance, it is possible in HybridSim to build a system which features a 15 way set associative cache with an access granularity of 64 bytes and a backing store with an access granularity of 1024 bytes. HybridSim also features some prefetching techniques such as sequential prefetching and stream buffers.

The simulator was originally developed by Jim Stevens, another member of the memory research lab at UMD. However, in the course of this work several significant modifications needed to be made to the simulator. These modifications generally related to the metadata storage of the DRAM cache and to the way in which HybridSim interfaced with the other simulators in the suite. HybridSim was originally designed with the assumption that the tags and other cache meta data would be stored in a small separate memory that could be quickly accessed. Additionally, methods of storing and accessing the tags for the DRAM cache had to be added to the simulator to enable the work that is presented in the DRAM cache portion of this dissertation. The goal of that was to investigate different ways of collocating tags and data in the DRAM in an efficient way. To that end, HybridSim was modified to to simulate a DRAM cache that requires a separate access for the tags and data in order to perform a cache access. A direct mapped DRAM cache design similar to Alloy cache [81] was also implemented to allow for comparisons against this latency optimized design. Finally, the Combo-Tag design proposed by this work was also implemented in HybridSim in detail. This required extensive changes to the cache address translation code in HybridSim as well as the addition of the tag buffer and it various replacement policies.

HybridSim was also modified to allow it to utilize different memory simulators in different positions of the main memory hierarchy. For the work in this dissertation the two memory simulators that are used are OMS and DRAMSim2 [96], a cycle accurate, hardware verified DRAM memory system simulator. HybridSim was originally designed to use DRAMSim2 as the cache and OMS as the backing store.

However, other organizations are possible which would require OMS to simulate the cache technology (such as RLDRAM) and DRAMSim2 to simulate the backing store technology (such as LPDDR). To enable investigations into many different multi-level memory system technology choices, several different versions of HybridSim were created for each possible combination of the two memory simulators.

5.3 Full System Simulation

The full system simulation environment used in this work is an extension of the MARSSx86 cycle-accurate full system simulator, which is comprised of the PTLSim CPU simulator and QEMU emulator sub-components. PTLSim models an x86-64 Multicore processor with full details of the pipeline, micro-op front end, trace cache, reorder buffers, and branch predictor. This processor simulation also includes a full cache hierarchy model and implements several cache coherency protocols. In addition, MARSSx86 utilizes QEMU [97] as an emulation environment to support any hardware not explicitly modeled by the full system environment, such as network cards and disks. This full system simulation environment is able to boot a full, unmodified operating system, such as any modern Linux distribution, and run standard benchmarks such as PARSEC or SPEC. The simulator captures both the user-level and kernel-level instructions, unlike other simulations that are user-level only, enabling the study and modification of the operating system.

To model the memory system, DRAMSim2 has been integrated into MARSSx86 to service last level cache misses. Like OMS, DRAMSim2 is highly parameterizable

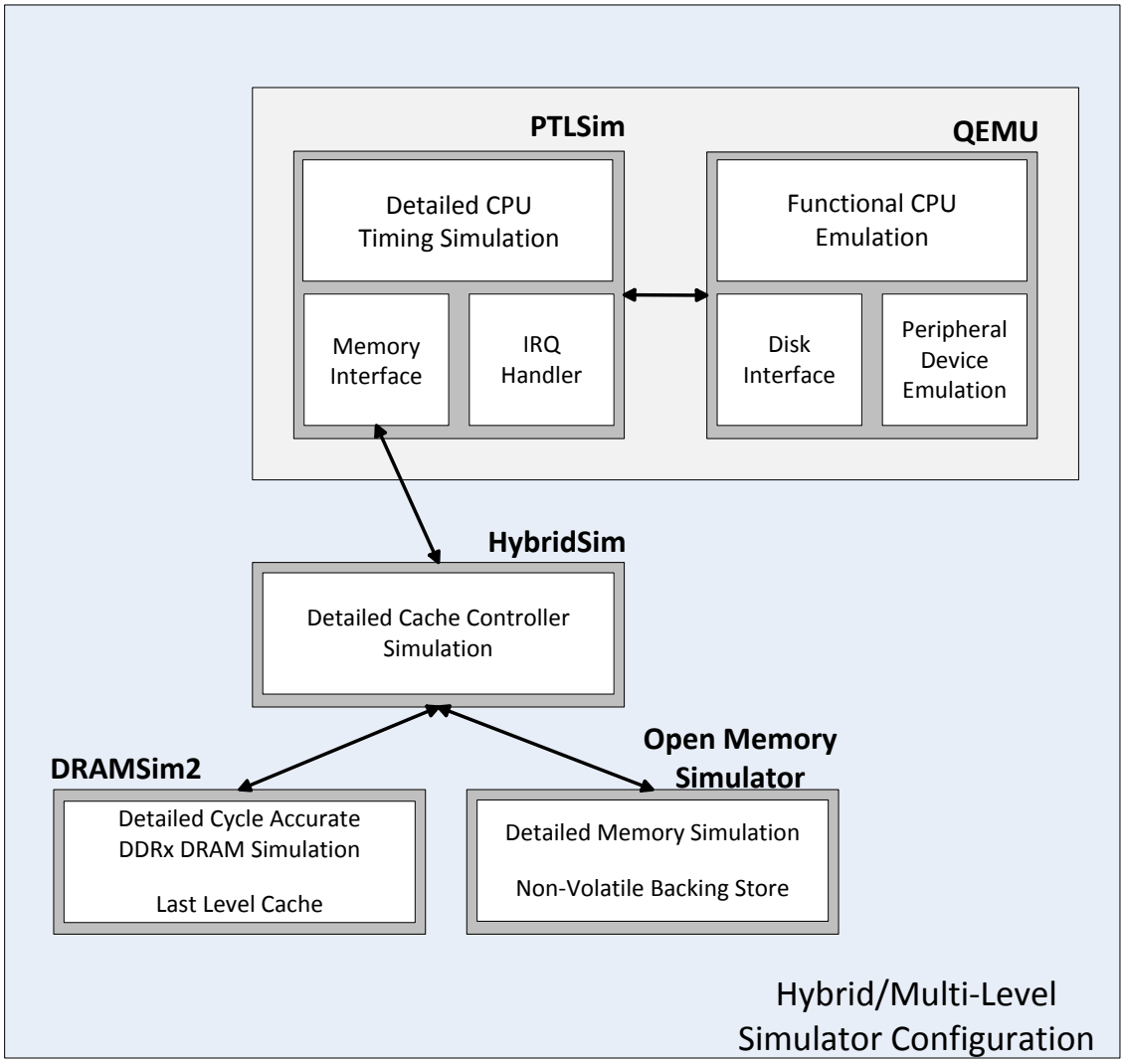

Figure 5.2: Block diagram of a simulation environment for systems with a multi-level organization. The placement of the memory simulators (DRAMSim2 and OMS) in this diagram represent only one of several possible organizations.

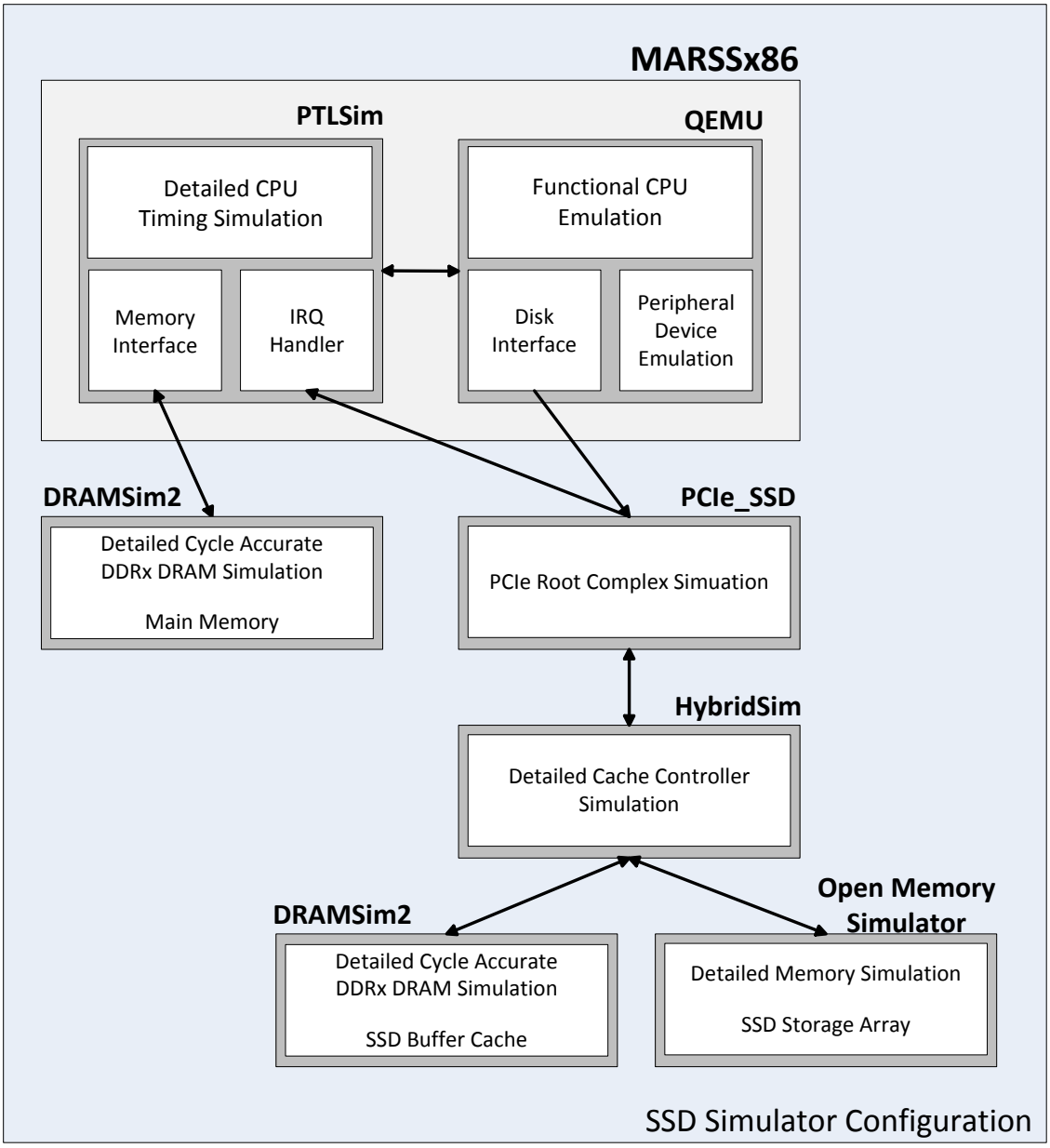

Figure 5.3: Block diagram of the simulation environment for systems with an SSD organization.

and allows for the modification of DRAM memory system organization, including the number of channels, ranks, banks, and so on. The multi-level memory system simulator suite extends the base MARSSx86 + DRAMSim2 system further by adding OMS and HybridSim. OMS provides the detailed simulation of the non-volatile or non-traditional DRAM memory systems that are used in the main memory hierarchy. HybridSim simulates the memory controller for a hybrid main memory system, providing cache management mechanisms to utilize the DRAM and non-volatile memory systems efficiently. A block diagram of the multi-level main memory version of our simulation suite is shown in Figure 5.2. The version of the simulator shown in Figure 5.2 is organized so that DRAMSim2 is providing the DRAM cache simulation while OMS is performing the non-volatile backing store simulation. This was the original way in which the simulation suite was organized. However, extensions to HybridSim have made it possible to utilize both DRAMSim2 and OMS to simulate either level of the main memory hierarchy.

Finally, to understand the performance of the hybrid main memory relative to current systems that utilize solid state drives, we created a full-system SSD simulation and integrated it into MARSSx86. The block diagram of the SSD version of our simulation is shown in the bottom of Figure 5.3. In this configuration, the main memory consists only of a typical DDRx DRAM main memory system simulated using DRAMSim2. Pages that then miss the DRAM main memory are requested from an SSD by the unmodified OS that is running in the full system simulation. The SSD simulator is implemented as a module wrapped around HybridSim called PCI_SSD that simulates the host interface and DMA engine simulation. The SSD

model also explicitly simulates direct memory access (DMA) via a callback to the DRAMSim2 main memory. The addresses for these DMA requests are extracted from QEMU's scatter-gather lists. These lists consist of pairs of pointers and sizes to enable the DMA request to access non-contiguous locations in the DRAM address space. The work developing the PCI_SSD wrapper was primarily performed by Jim Stevens. To the best of our knowledge, this simulation suite provides the first full-system SSD simulation. Prior SSD simulation work [98, 99] exclusively utilized trace-based simulation to study SSD performance.

Chapter 6: Overall System Organization Analysis

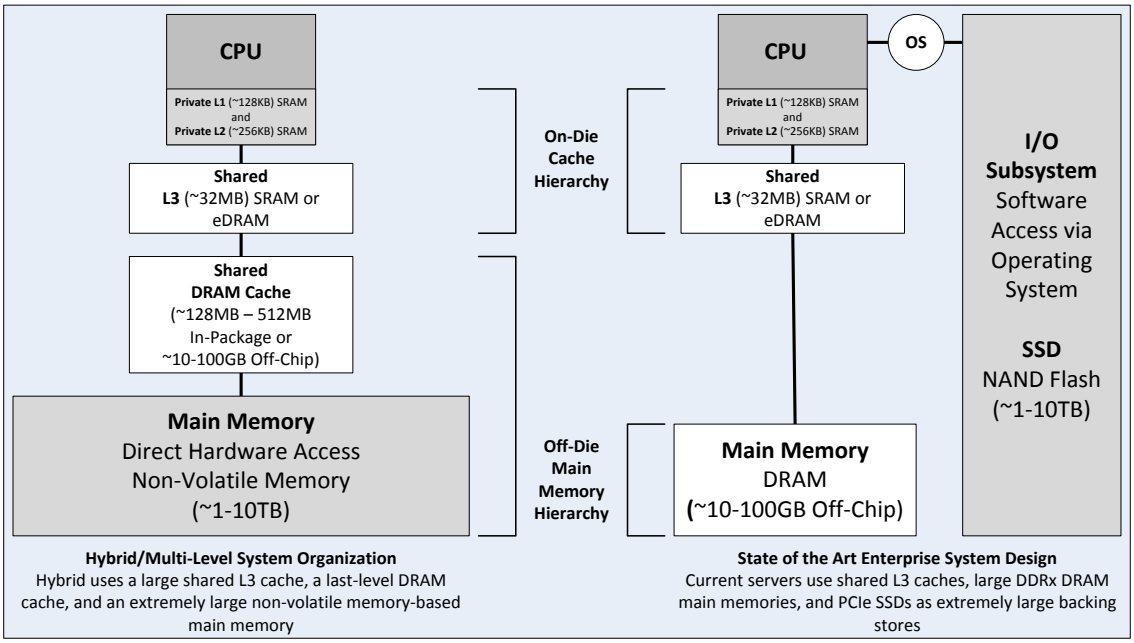

Figure 6.1: Hybrid organization versus a typical enterprise-class SSD organizataion

The first goal of this work is to better understand how certain features of the underlying hardware architecture or workloads affect the performance of systems that utilize either a software or a hardware managed DRAM cache. To achieve that understanding we began by identifying the two principle factors that affected performance in both types of system: the miss rate of the DRAM cache and the average hit/miss latency ratio. The experiments that make up this study therefore

focus on the features that affect those two principle factors such as backing store latency, cache size, associativity, etc. Together these experiments provide a clear picture of which architectural and workload features benefit or harm the performance of both SSD and Hybrid systems and why. For the purposes of this study we do not focus on any particular backing store technology but instead examine points of interest on the range of possible backing store latencies.

6.1 Evaluation Methodology

The system that we use to perform the experiments in this chapter is described in Table 6.1. For these studies we use the MARSSx86 simulator to perform full system simulations. We use MARSSx86 here because it captures both the user-level and kernel-level instructions, unlike other simulations that are user-level only, enabling the study of the operating system. The memory systems in this work are modeled with HybridSim, OMS, and DRAMSim2. For the SSD portions of these experiments we utilize our PCI_SSD module which allows us to use HybridSim, OMS, and DRAMSim2 to model the internals of a typical SSD. The SSD implementation also uses an AHCI driver instead of the default IDE drivers in QEMU. This enables Native Command Queueing and allows the SSD-based system to take advantage of hardware parallelism.

Table 6.1: Baseline Simulator Configuration

Processor	
Number of cores	4-core
Issue Width	4
Frequency	2GHz
On Chip Caches	
L1I (private)	128 KB, 8-way, 64 B block size
L1D (private)	128 KB, 8-way, 64 B block size
L2 (private)	256 KB, 8-way, 64 B block size
L3 (shared)	32 MB, 20-way, 64 B block size
In-Package DRAM Cache	
Organization	16-way, 4 KB page size
DRAM Bus Frequency	DDR3-1333
DRAM Bus Width	64 bits per channel
DRAM Channels	1-16
DRAM Ranks	1 Ranks per channel
DRAM Banks	8 Banks per rank
Row Size	1024 Bytes
tCAS-tRCD-tRP-tRAS	10-10-10-24
Backing Store	
Organization	4 KB page size
Backing Store Bandwidth	PCIe 3.0 x16 equivalent

6.1.1 Benchmarks

To simulate random read access, we utilize the GUPS and MMAP benchmarks. GUPS is based on an implementation of the Giga-Updates Per Second benchmark

by Sandia National Labs [100]. In our experiments, this benchmark allocates a table that is one eighth the size of the simulated backing store in the virtual memory space and then randomly updates locations within it 5000 times. Since the table is larger than main memory, GUPS forces the virtual memory system to swap the tables pages to the backing store.

Our MMAP experiment maps a file that is half the size of the simulated backing store into the virtual address space with the mmap() system call and then performs 10000 random reads to the file. Once again, since the file is larger than main memory, the file system cannot read the entire file into the buffer cache and is forced to read the random addresses from the backing store. For the Hybrid memory version, we overlay a filesystem on top of the memory address space using a tmpfs ramdisk. The purpose of MMAP is to provide a filesystem workload in contrast to the swapping workload of GUPS so that we can understand different aspects of OS overhead when comparing the SSD and Hybrid architectures.

Both MMAP and GUPS also incorporate OpenMP to allow for multiple threads performing random accesses to utilize more parallelism at the hardware level for our study. To understand the effect of varying the degree of random access within our workloads, the gups benchmark also has a random probability, which determines if the next access will be sequential following the current access or if it will be an independent random access. Finally, because both of these benchmarks are run to completion in our tests, execution cycles are used to measure performance.

To test the Hybrid and SSD systems with a large sequential workload, we built the microbenchmarks DD_READ and DD_WRITE based on the dd Linux utility.

Both benchmarks measure the time required to move a 64 MB file between the backing store and DRAM. For DD_READ, the 64 MB file is created and then forced to the backing store with sync and cache flush operations. The dd program is then used to copy that file into memory. In the case of the Hybrid memory, a file system is created using a tmpfs ramdisk and we force the sections of main memory that make up the ramdisk to be flushed to the backing store before the DD_READ starts. DD_WRITE performs a dd run to copy data from /dev/zero and write the data to the disk. For the SSD case, we ensure that the DD_WRITE actually writes to the disk rather than the RAM buffer cache by using the conv\=fdatasync option to dd. For the Hybrid case, we use a special MMIO operation to tell the memory controller to flush dirty pages in the cache to the backing store.

Together we refer to these four workloads as the targeted benchmarks as their intent is to isolate certain behaviors of the systems in question.

To provide context for the results from our targeted workloads, we also utilize several representative benchmarks from the Filebench workload generator [101]. Filebench is a benchmarking tool for the storage system that utilizes scripts written in the Workload Modeling Language (WML) to specify file system interactions for different types of workloads [102]. In WML, users specify a set of processes, which can contain multiple threads, to interact with a number of filesets. The filesets can contain any number of files and any desired directory depth necessary to simulate files used by a real application. Each thread performs a sequence of file operations such as create, delete, read, write, and append, as well as operations to synchronize between threads. Filebench workloads execute with the *run* command that specifies

the number of seconds to execute. Threads repeat their operations in a loop until the time expires. Therefore, unlike the targeted benchmarks, performance for these workloads is measured in IOP/S because they run for a fixed time of 250 ms in the ROI of each workload.

Filebench provides a large number of pre-defined WML files to model common workloads. In this chapter, we chose to utilize the four pre-defined full application workloads: *fileserver*, *varmail*, *webproxy*, and *webserver*. *Fileserver* creates a large number of threads that perform a sequence of create, write, append, read, and stat operations on a large number of files to simulate the I/O operations on a typical file server. *Varmail* simulates a mail server by having a moderate number of threads interact with a large set of small files (which simulate emails) with appends, reads, and deletes. *Webproxy* simulates a proxy server by having a large number of threads read, write, and delete a large number of small files, which simulates proxy caching operations, and an append operation to a log file. *Webserver* has a large number of threads perform a number of file reads and an append to a logfile to simulate a website being uploaded to a large number of simultaneous users.

6.2 The Effect of Prefetching

We begin our investigation by looking at the effects of prefetching on the performance of the Hybrid approach compared to the SSD approach. We noted in some of our earliest observations of the SSD based system that the operating system could, in some circumstances, begin to very aggressively prefetch data from

the SSD. This led us to conclude that some considerable speedup could be achieved by introducing prefetching to the Hybrid system in an attempt to minimize the number of misses that occur in the DRAM cache. To investigate the impact of prefetching we performed an experiment where we gradually increased the size of the prefetching degree in the Hybrid system in order to determine the extent to which prefetching could improve the performance of the hybrid system.

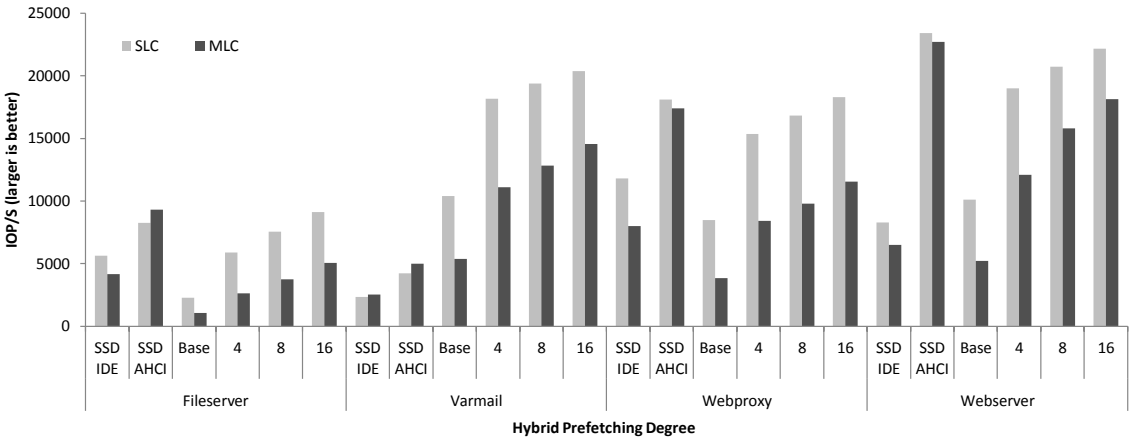

Figure 6.2: The effect of different degrees of prefetching on Hybrid system performance for the representative file system benchmarks. We include SSD performance values for comparison. The Y-Axis is IO operations per second, so larger is better.

Figure 6.2 shows the results of our prefetching experiment compared to the SSD performance for systems with both SLC and MLC flash backing stores. From this graph we can see that prefetching helps to provide the performance needed to bridge the gap between the baseline Hybrid system and the SSD system in 3 of the 4 file system workloads. This makes sense because, as we have mentioned, the SSD system is already aggresively prefetching so adding prefetching to the Hybrid

system simply makes them more equal in terms of features. The results indicate that even an aggressive sequential prefetching degree of 16 pages provides a performance improvement in all cases so for the rest of this chapter we will use a prefetching degree of 16 pages for the Hybrid system. We chose not to extend the prefetching degree beyond the 16 page size because increasing the number of prefetches greatly increases the simulation time and because our initial investigations showed dimishing returns beyond the 16 page size.

6.3 The Effect of Backing Store Latency

Having established the importance of prefetching, we continue our investigation by looking at the effects of the backing store latency on the performance of the SSD and Hybrid approaches. This is a logical next step since the difference between storage and memory technologies has traditionally largely been determined by their access latency. The potential to task switch and perform useful work while a page fill is taking place makes the SSD approach better suited to longer latency technologies. However, it is unclear exactly how long that latency should be in order to see any benefit from the SSD approach. So, we are looking for the crossover point where the backing store is too slow for the Hybrid approach and better suited to the SSD approach. To find this crossover point, we steadily increase the read latency of the backing store by a factor of 10 from 125 to 12500 ns. In addition, we also include some additional latencies at particular points of interest. For instance, 25000 ns is the read latency of SLC NAND Flash and 75000 ns is the read latency

of MLC NAND Flash. Because most non-volatile technologies feature asymmetric read to write latencies, we use a write latency that is 10x the read latency for these experiments. However, with the exception of DD_Write, the benchmarks in this study do not feature write traffic to the backing store as a system bottleneck.

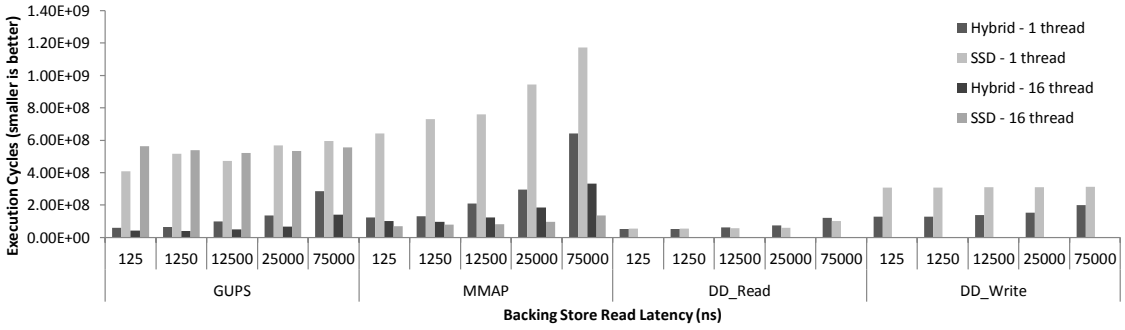

Figure 6.3: The effect of different backing store read latencies on System Performance for the targeted benchmarks. The Y-Axis is the execution time in cycles, so smaller is better. DD_Read and DD_Write are single threaded only.

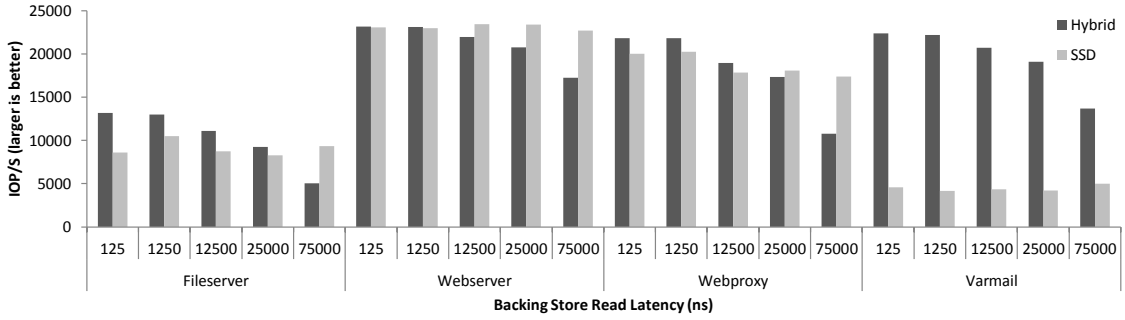

Figure 6.4: The effect of different backing store read latencies on System Performance for the representative file system benchmarks. The Y-Axis is IO operations per second, so larger is better.

Comparing the results in Figure 6.3 it is clear that for the random workload

GUPS, the Hybrid architecture performs considerably better at all backing store latencies. This suggests that the cost of a cache miss is considerably greater for the storage implementation than it is for the memory one. In addition, the Hybrid approach is clearly advantageous at all backing store latencies for the single threaded case of MMAP. However, increasing the thread count from 1 to 16 improved the performance of the SSD based system considerably more than the Hybrid system and resulted in the SSD system out performing the Hybrid system. This was not the case with GUPS and that suggests that tmpfs may be responsible for some of the lost performance in the multi-threaded Hybrid MMAP runs.

DD_Read shows the performance of the SSD to be slightly faster than the Hybrid system. At this point we hypothesized that the superior sequential performance of the SSD system was probably due to its prefetching and associativity which helped to keep its miss rate lower. This also explains why DD_Write does not show a similar advantage for the SSD. The prefetching and associativity of the storage implementation cannot help with a write heavy workload even if it is streaming. Also, because the writes are triggering page faults and cache misses there is a slight edge for the Hybrid system in the DD_Write benchmark due to the additional page fill overhead suffered by the storage system.

The results in Figure 6.4, however, tell a slightly different story. In these representative workloads we see the SSD system pulling ahead in quite a few benchmarks. At MLC latencies the Hybrid system is only better in the Varmail workload. This indicates that many realistic workloads benefit from the prefetching and associativity advantages that enabled the SSD system to outperform the Hybrid system

in the DD_Read workload. Also, even though DD_Read is only one of three target workloads, this result demonstrates how important this workload is for capturing the more easily prefetched and cached address streams of many common storage workloads.

An important feature of this experiment is that we can see the crossover points where the performance of the storage system surpasses the performance of the memory architecture. The crossover points are really only visible in the representative workloads as the targeted workloads tend to always favor either the SSD or the Hybrid system. Among those workloads that have a crossover point we see that at SLC latencies the Hybrid and SSD architectures are very close in terms of performance while the storage system performs much better at MLC latencies in almost all cases. These results show that technologies such as SLC NAND Flash could be used more aggressively as the backing store technology of some Hybrid systems. SLC NAND backed Hybrid systems break even with SSD systems in most cases and achieve significant performance gains in others.

6.4 The Software Overhead of the Storage System

Table 6.2: Software and Hardware Access Time

	Total Time (ns)		**Hardware Time (ns)**		**Software Time (ns)**	
	Mean	**Stdev**	**Mean**	**Stdev**	**Mean**	**Percent Software Delay**
SLC Latency	85360.93	33837.39	38900.67	6689.59	46460.26	54.43
MLC Latency	162148.72	61060.33	88750.12	13016.83	73398.60	45.27

From the latency sweep results we can clearly see that at lower backing store

latencies there is some overhead in the storage system that is negatively affecting its performance. In order to characterize the nature of the storage overhead we need to quantify the contribution of the software portion of a storage access. We accomplished this by instrumenting our MMAP benchmark to log when a request began and ended at the application level. We created the instrumentation by using x86 rdtsc instructions that ran immediately before and after each access. We also implemented code in the SSD host interface to record when accesses began and ended at the hardware level. The hardware time includes the host interface time, the time it took to process the access in the SSD controller, and the time it took to perform the DMA. Since the raw time from the software level logs include the hardware time, we must subtract the hardware time from those raw values to compute the actual time spent processing the access in software. To provide an accurate picture of the associated delays, 10000 accesses were measured and their delays were averaged. Also, the same analysis was performed with a backing store that had a read latency roughly equivalent to SLC NAND Flash (25000 ns) and MLC NAND Flash (75000 ns). The resulting values are presented in Table 6.2.

From these results it is clear that the software level of a storage system access represents a significant portion of the total delay. As the latency of the backing store increases, the relative percentage of the delay that is due to software decreases because it remains relatively constant. However, even at MLC NAND Flash latencies the software delays represent almost half of the time it takes to access the backing store.

It is also worth pointing out that the standard deviation of the total delay is

roughly five times the standard deviation of the hardware delay. This indicates that the software layer introduces a significant amount of nondeterminism to the system. This same effect can clearly be seen in our other results where the traditional SSD approach exhibits considerably more nondeterminism than the Hybrid architecture.

6.5 The Effect of Random Access

The impact of high miss rate applications on the performance of both Hybrid and SSD systems led to questions about the importance of miss rate on the overall performance of both systems. As a result, the following experiment was developed to gauge the effect of varying degrees of randomness in the memory access pattern on both the Hybrid and SSD systems when they are swapping. For this experiment, each access has a probability of being either sequential or random and by changing the probability, we can adjust the degree of randomness in the workload. For this experiment we held the latency constant at 25000ns for all data points.

In Figure 6.5, we can see that only 10 percent random reads introduces significant performance degradation compared to the purely sequential case. This suggests that even programs which are largely sequential can benefit from the Hybrid architecture's efficient handling of random reads. In addition, the multithreaded version of the workload appears to introduce additional randomness that further degrades performance for the SSD versions of the system when the workload is relatively sequential. This is because the relatively sequential threads interfere with one another and produce memory traffic that is largely random. However, as the randomness of

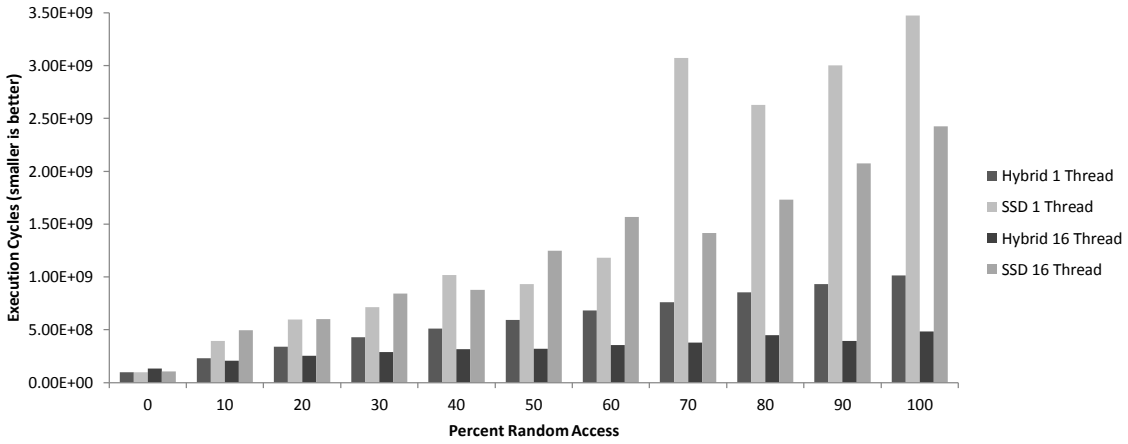

Figure 6.5: The effects on system performance from different percentages of random access for the GUPS benchmark. The Y-Axis is the execution time in cycles, so smaller is better.

the workload increases the performance benefits of the multithreaded version begin to outweigh the randomness introduced by the multithreading interference. This transition appears to occur at around 70 percent random access. However, the considerable involvement of the OS in the SSD version of the system introduces some nondeterminism to the results. The Hybrid architecture, on the other hand, is much more deterministic due to the reduced dependence on the OS. The Hybrid architecture also benefits from the additional threads for all levels of random access. As the degree of randomness increases, the performance boost provided by the Hybrid architecture increases from around 2X at the 10 percent randomness point to 5x when the workload is totally random. Furthermore, the Hybrid architecture handles the purely sequential workload almost as well as the SSD despite lacking many of the optimizations that have been developed for sequential disk accesses.

It is important to note though that the performance differences seen in this experiment do not reflect the results from the representative workloads. In those workloads the SSD performed better or equal in most cases at 25000ns but in this experiment it is surpassed by the Hybrid organization at just 10 percent random reads. To further investigate these results we next analyze the aspects of the cache that tend to affect the miss rate in an effort to determine if some other factor besides randomness is contributing to the representative workload performance of the SSD.

6.6 The Effect of Associativity

Associativity is often an effective way to reduce the miss rate of a workload. However, while the page table of the OS is functionally fully associative the Hybrid system was limited to 16 ways of associativity in the prior experiments. To investigate the effects of associativity we swept the degree of associativity available to the Hybrid system from direct mapped to 64 way set associative. For this experiment and the remaining two experiments we set the backing store latency to 17000 ns. This value was selected because it represents a middle value that does not overly favor any particular aspect of the system that has been studied thus far.

The results in Figure 6.6 show that associativity has a relatively minimal effect on performance. Both MMAP and DD_Read experience a roughly 30% speedup as a result of increasing the associativity from direct mapped to 64 way. It is interesting to note that the multi-threaded workloads appear to benefit more from the added associativity. This indicates that the different threads are creating some

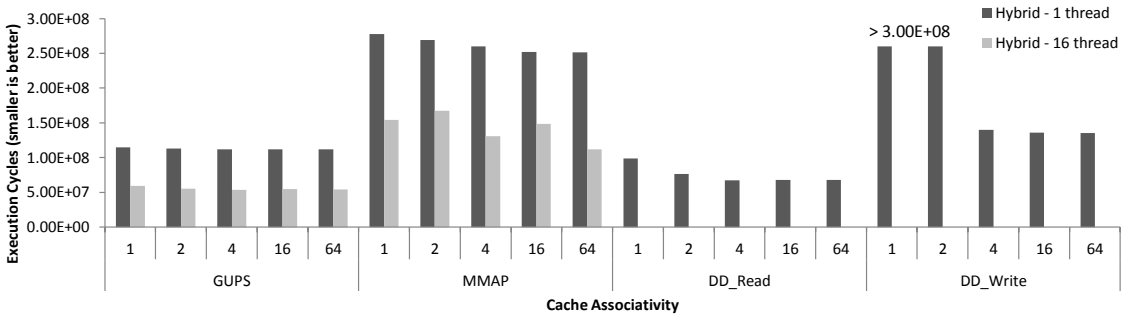

Figure 6.6: The effect of different levels of DRAM cache associativity on the performance of the Hybrid system for the targeted benchmarks. The Y-Axis is the execution time in cycles, so smaller is better.

set contention in the lower associativity cases. However, the multi-threaded MMAP runs exhibit some additional non-determinism as a result of the OS scheduler and so it is difficult to accurately gauge the precise speedup. Finally, the direct mapped and 2 way associative runs of DD_Write failed to complete in all attempts. The additional write pressure on the backing store created by the lack of associativity seems to have overwhelmed the backing store resulting in extremely long access latencies.

Given the 30% improvement seen in DD_Read it seemed that the superior associativity of the storage implementation might have been a contributing factor to its performance. So, we repeated this same experiment with the representative workloads. However, we found that only fileserver exhibited any sensitivity to associativity. Moreover, fileserver was slowed by reducing the associativity to direct mapped but did not receive any boost from increasing associativity beyond 4 way. Therefore, associativity on its own could not be the source of the SSD's performance

advantage.

6.7 The Effect of Cache Size

Another possible source of misses could come from the cache being too small for the working set. However, this can be mediated by clever cache management schemes that retain only the most useful data thereby maximizing the available cache space. By varying the size of the cache we can expose the cache management scheme and determine how effective it is at utilizing the space in the cache.

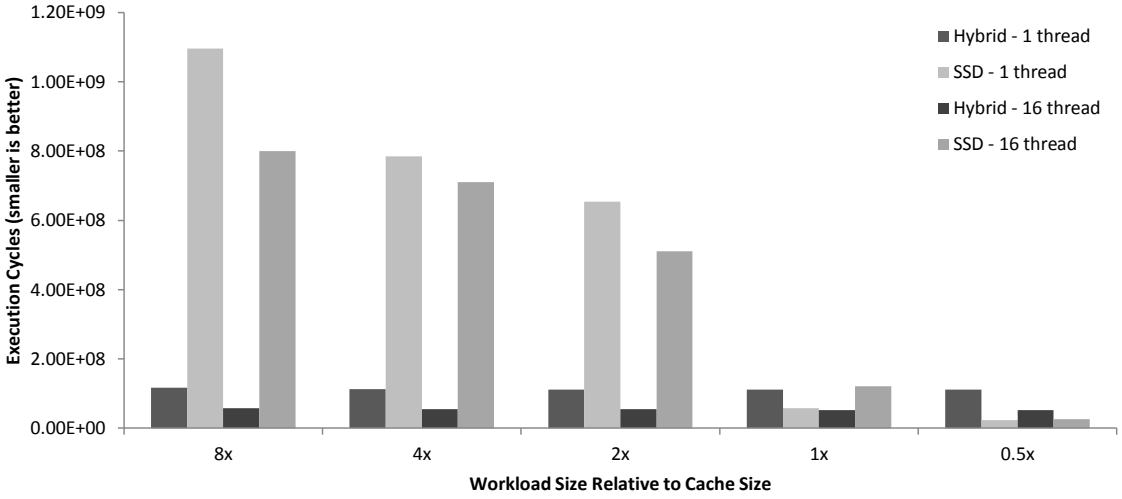

Figure 6.7: The effect of cache size relative to workload size on System Performance for the GUPS targeted benchmark. The Y-Axis is the execution time in cycles, so smaller is better.

We can see right away in Figure 6.7 that the cache management schemes in the SSD and Hybrid approaches are doing something very different. The size of the cache has almost no effect on the Hybrid system while the SSD benefits greatly

from increasing the cache size. This seems to confirm that the SSD is prefetching very aggressively as the randomness of GUPS should limit the effectiveness of most prefetches. We confirmed this by checking the size of the requests being issued by the OS and noted that they can grow to be quite large (on the order of megabytes). By prefetching so much the SSD is benefiting from a birthday attack-like effect, in which prefetching the pages from the backing store is helping some future accesses with a certain probability. However, this prefetching results in greatly reduced performance when the cache is small because the replacement policy is unable to prevent the cache from being polluted by useless prefetched data.

6.8 The Effect of Cache Concurrency

In addition to potentially polluting the cache, aggressive prefetching can also place a greater strain on the concurrency of the cache. In order to determine if the available cache concurrency was a bottleneck we performed an experiment where we swept the number of channels in the DRAM cache from 1 to 16.

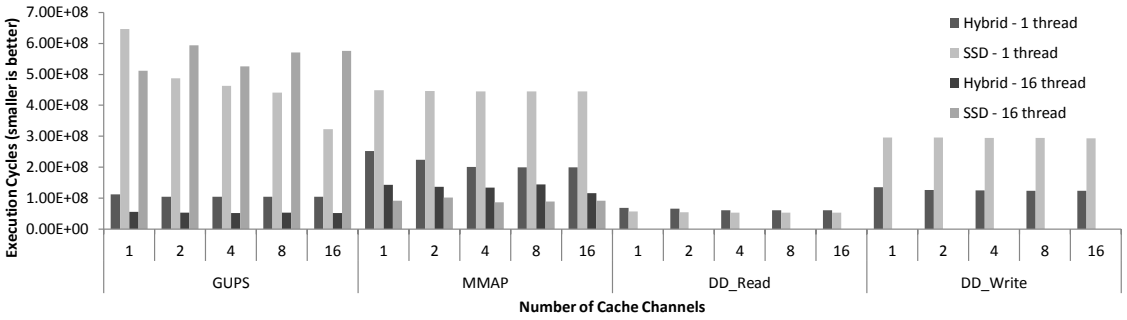

Figure 6.8: The effect of cache concurrency on system performance for the targeted benchmarks. The Y-Axis is the execution time in cycles, so smaller is better.

From the results in Figure 6.8 we can see that increasing the available concurrency in the cache does not significantly speed up most of the SSD runs. So, we can be reasonably sure that though the SSD is exerting a lot of prefetch pressure on the cache, that pressure is not interfering with normal requests in most cases. However, the single threaded GUPS SSD benchmark shows a nearly 2x speedup due to increasing the concurrency. This suggests that cache pressure is a major bottleneck for the SSD in the single threaded GUPS runs. The ability to switch to other threads while waiting on a DRAM cache accesses appears to negate this effect in the multithreaded GUPS runs. Interestingly, some of the Hybrid runs also show some improvement with increased cache concurrency suggesting that the Hybrid system is using the cache less efficiently in certain workloads. In particular, the MMAP workload shows a considerable speed up. This could account for some of the performance difference between the SSD and Hybrid systems that was noted earlier for this workload.

6.9 Comparison to DRAM-only

Finally, we are also interested in how both of the Hybrid and SSD systems compare to a system that had enough DRAM to store the entire workload of the benchmarks we're using. This test gives us an idea of how closely these two alternative systems come to the traditional main memory architecture in terms of performance. In addition, we perfromed these experiments with SSDs using both the older IDE drivers and the newer AHCI drivers to demonstrate the impact that

driver efficiency can have on disk performance. We used the representative file system benchmarks for these tests because they provide the most accurate representation of the sort of traffic that real world system might encounter.

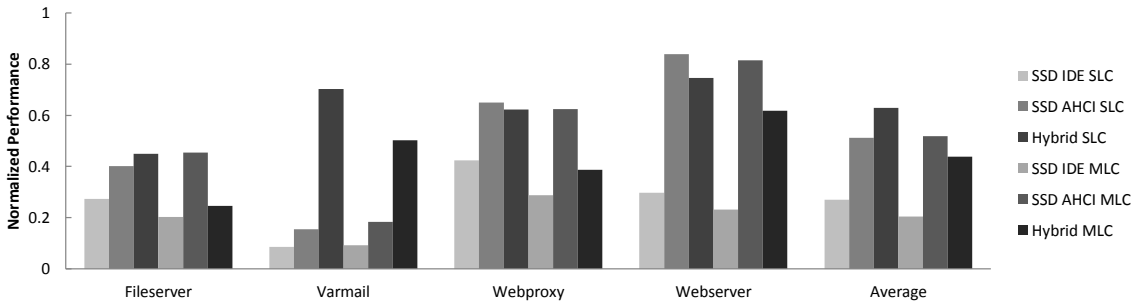

Figure 6.9: The performance of both the Hybrid and SSD system organizations normalized to a system with more than enough DRAM to contain the working sets of each of the workloads.

The results in Figure 6.9 show that even with the considerably longer access latencies of SLC flash, the Hybrid system is able to achieve roughly 50% of the performance of the DRAM only system on average. We can also see that employing the AHCI driver greatly improves the efficiency of the SSD as it enables native command queuing and allows the system to fully utilize the available hardware parallelism.

6.10 Summary

In this chapter, we have presented a series of experiments which clearly illustrate the advantages and disadvantages of Hybrid and SSD-based multi-level memory architectures. These experiments have demonstrated that SSD-based sys-

tems perform best when workloads are highly sequential, DRAM cache sizes are large, and backing store technologies are slow. Conversely, the Hybrid-based system performs best when the workload is more random and the backing store is fast. In addition, we have quantified the direct delay introduced by the software during storage system access and have shown that the total cost of a software overhead (file system, task switch, etc.) is a significant component of the storage approachs access latency. Finally, we have also shown that slower technologies such as SLC NAND Flash can be successfully utilized in more aggressive Hybrid systems. Overall, this work has shown that there is a clear place for both the hardware and software managed approaches to building multi-level memory systems. Furthermore, by taking certain specific system and workload attributes into account it is possible to safely decide which approach is best suited to a particular use case.

Chapter 7: DRAM Caches

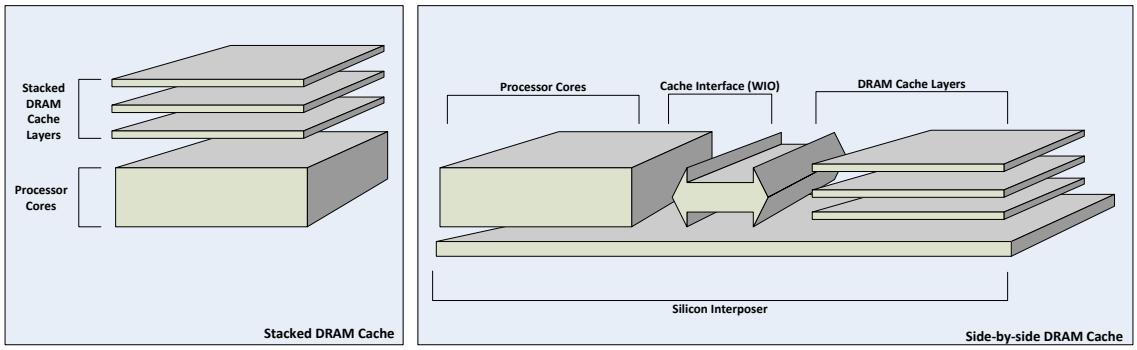

Figure 7.1: Some examples of recently proposed in package DRAM caches.

The advent of in-package DRAM caches has opened the door for a wide variety of potential system architectures which utilize different memories and organizations to build faster, larger, and more energy efficient main memory systems. However, DRAM caches have their own set of unique design decisions that are different from both memory and cache design considerations. This places the design of DRAM caches in an interesting space that simultaneously involves cache-like trade-offs such as tag lookup time versus miss rate and main memory concerns such as channel utilization. The interplay between these different aspects of the DRAM cache architecture can result in some complex performance effects at the system level. As we have seen in the previous chapter, the DRAM cache often plays a significant role

in the performance of the multi-level main memory system. Therefore, effectively balancing all of these concerns is critical to the performance of the cache and to the performance of the system as a whole. In this chapter, we will provide an overview of these different DRAM cache design considerations.

7.1 DRAM Cache Design

The design of DRAM caches involves both cache-like and main memory-like design elements. In this section we will provide some examples of those decisions and explain why they are important. In addition, we will also cover several design considerations that are unique to DRAM caches.

Some of the cache-like design considerations that must be taken into account when designing a DRAM cache include determining the degree of associativity, deciding on a replacement policy, and settling on whether or not to use virtual addresses for either the tags or indexes of the cache. As is the case with SRAM caches, implementing associativity in DRAM caches is a possible source of performance, especially in systems with longer latency backing stores. In addition, in some multi-level main memories the size of a cache fill from the backing store could be potentially much larger than the block size of the cache. As a result, replacement policies that take into account the differences in block size, fill size, and reuse distance of the DRAM cache are another important design consideration. Finally, like other caches, the tags and indexes of the DRAM cache can be derived from virtual or physical addresses. In all of the designs that we consider in this dissertation, we

use only physical addresses for both the tags and index because all of the requests seen by the cache controller are for physically addressed memory accesses. However, it is possible to conceive of a system that does not perform the virtual to physical translation until after the DRAM cache level which would enable the utilization of virtual tags or indexes in the DRAM cache.

The main memory-like design considerations are more related to the actual structure of the DRAM memory that we are using to implement the cache and include providing enough bandwidth for the system, providing enough concurrency in the system and deciding whether or not to provide open page support. A typical DRAM transaction takes much longer to complete than a normal SRAM transaction and so ensuring that there are enough concurrent units to satisfy all of the outstanding requests at each point in time is very important. Also, because the DRAM channel is shared between many different concurrent units it is critical that there is enough bandwidth to handle all of the traffic. Finally, open page support can provide some latency speedup in certain situations but can also result in lost performance if the address stream does not take advantage of it. Therefore, implementing an open page policy DRAM cache requires the design of an effective address mapping scheme and the careful analysis of the types of workloads that the system might encounter to ensure that the open page policy will result in a speedup.

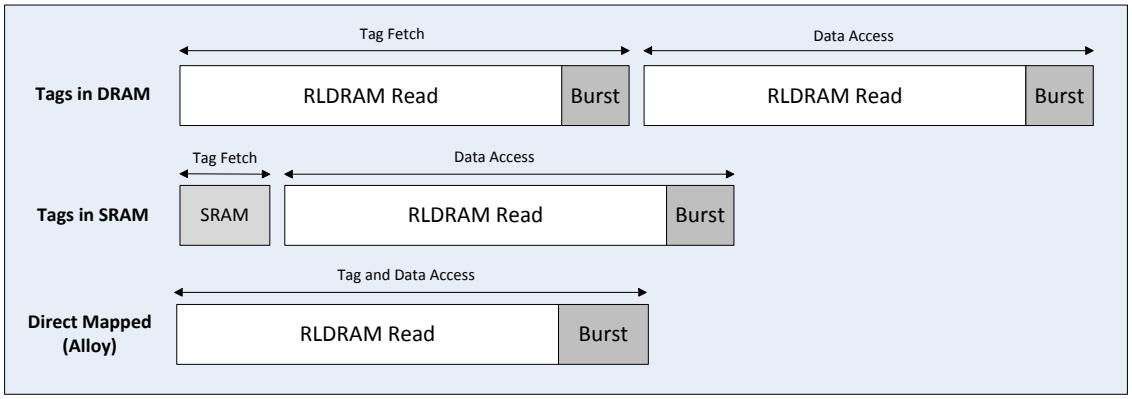

Figure 7.2: A comparison of the different tag access schemes for DRAM caches.

7.1.1 Meta-Data Storage

One of the most important decisions that must be made when designing a DRAM cache is where to store the meta-data (such as tags and valid bits). The simplest approach is to store the meta-data in SRAM, either on chip or off. By storing the tags in SRAM we can ensure that the tag lookup time will be relatively negligible compared to the data access time in the DRAM. However, the amount of SRAM required to store all of the tags is prohibitively large. For instance in a 48-bit address space, the tags for a 128MB cache would need roughly 6MB of SRAM for storage. This represents a significant portion of the on-chip SRAM that is typically used for caches. More importantly, though, the size of the SRAM tag store would need to scale with the size of the DRAM cache. So, a 1GB DRAM cache would need roughly 8x the SRAM to store its tags or 48MB, which is larger than most CPU caches today. Clearly then, a straightforward SRAM tag store is impractical in the long run and some other solution for tag storage is needed.

The primary alternative to an SRAM tag store is to store the tags in DRAM along with the data. The problem with storing the meta-data in DRAM, though, is that it negatively impacts the latency of the cache. This is because two DRAM accesses are required for each cache hit access, one to fetch the tags and another to fetch the data. Figure 7.2 provides a comparison of the DRAM access process to the fast SRAM tag store access process. As we will see in the next chapter, this two stage access process can result in a system that is relatively slow due to the long hit access latency and the extra bandwidth utilization of this approach.

The most straightforward way to address the problem of storing meta-data is to simply make the DRAM cache direct mapped. This eliminates most of the meta-data that is needed in associative caches. However, a tag is still needed to determine whether the data accessed is the desired data. Extending the size of a DRAM cache access is one way that has been proposed to enable the fetching of tags and data in the same operation [81]. Making the DRAM cache direct mapped solves the problem of tag accesses but it introduces a potential problem of miss rates. In many real world applications associativity is a valuable aspect of cache design that can help to prevent a considerable number of misses. Removing the possibility of associativity from the DRAM cache design space, therefore, is only acceptable as long as the miss rate remains low for the DRAM cache or the miss penalty is not significantly greater than the hit latency. The is the case in the current usage model for DRAM caches today, which frequently pairs a high bandwidth in-package DRAM cache with a low bandwidth DRAM backing store. The miss penalty in this system organization is not that much worse than the hit latency and so the direct

mapped DRAM cache provides acceptable performance. However, as alternative memory technologies are utilized to build the backing store, this miss penalty will increase and so will the importance of associativity. So, in the long run, a direct mapped DRAM cache may be unacceptable for many systems.

Storing all of the tags for the entire cache in an SRAM store has been shown to be impractical. However, a middle solution exists that utilizes a smaller SRAM to store only a portion of the total tags for the DRAM cache. The remainder of the tags are then stored in DRAM. This approach has several advantages when compared to the other three methods of storing meta-data that we have discussed thus far. First, it allows for SRAM latency tag look ups in many cases, provided that the correct tags are currently being stored in the small SRAM buffer. Second, it allows for the size of the DRAM cache to scale without necessarily needing to increase the size of the tag store at the same rate. And finally, it allows for the implementation of associativity. The problem with this approach is that it is vital that the tag store contain the desired tags most of the time. If the tags are not in the SRAM tag store then they must be fetched from the DRAM and the overall access latency approaches that of the tags in DRAM implementation. Therefore, maximizing the efficiency of the SRAM tag store is the key to achieving good performance with these designs.

7.1.2 Address Mapping

Another aspect of DRAM cache design that differs from SRAM cache design is the importance of correctly utilizing the available concurrency and bandwidth in the system. In SRAM caches the access times are relatively short compared to DRAM and so it is relatively uncommon to have more than a few accesses in flight at any given time. The DRAM system, on the other hand, features many separate levels of concurrent units and often has many operations in flight. As a result, making good use of the available concurrency is critical to achieving acceptable performance in a DRAM system. This is generally accomplished by setting up the address mapping of the DRAM controller so that it spreads the addresses evenly across as many different concurrent units as possible. Because the underlying system is the same in a DRAM cache, setting up the address mapping is also just as important. Treating the DRAM cache as a flat address space without regard for its channels, ranks, and banks can result in multiple accesses contending for the same resources and significant increases in average access latency.

The row buffer in the typical DRAM device is yet another aspect of the DRAM systems internal structure that needs to be kept in mind when deciding how to map addresses. This row buffer can provide a significantly reduced access latency on sequential accesses to the same row in the DRAM if the system is using a open row policy. Therefore, ensuring the nearby addresses hit in the same row is an important aspect to achieving good performance in open page DRAM systems.

Interestingly, many DRAM cache designs recognize the importance of the row

buffer and take care to utilize it with their mapping schemes. However, most of the same designs appear to not incorporate channels, ranks, and banks into their mapping schemes. And this can have serious negative impacts on performance as utilizing channels is often more important than utilizing row buffers.

7.1.3 Commodity Versus Custom Parts

DRAM based main memory systems generally utilize commodity parts that are relatively cheap and widely available. DRAM caches however, could potentially be built using customized DRAM cache devices that provide features that are beneficial to DRAM caches. One such potential customization would be to include additional concurrency in the device, either by adding additional channels or banks to the die. This additional concurrency would allow for more complex tag storage organizations such as putting tags on dedicated channels or putting tags on different channels from their corresponding data. This would enable simultaneous access of both tags and data at the cost of significant bandwidth utilization. However, if the cache provided enough excess bandwidth, this simultaneous access might provide enough of a performance improvement to justify the additional bandwidth utilization. Also, the additional bandwidth could be used to enable more extensive prefetching algorithms. The available bandwidth of current DRAM devices limits the potential speedup that can be gained using prefetching because prefetching often leads to a large increase in bandwidth utilization. This can result in longer access latencies for demand accesses because all of the available concurrency and band-

width is being used by prefetch accesses. Increasing the available bandwidth and concurrency could help to alleviate this problem and provide for more aggressive prefetching approaches.

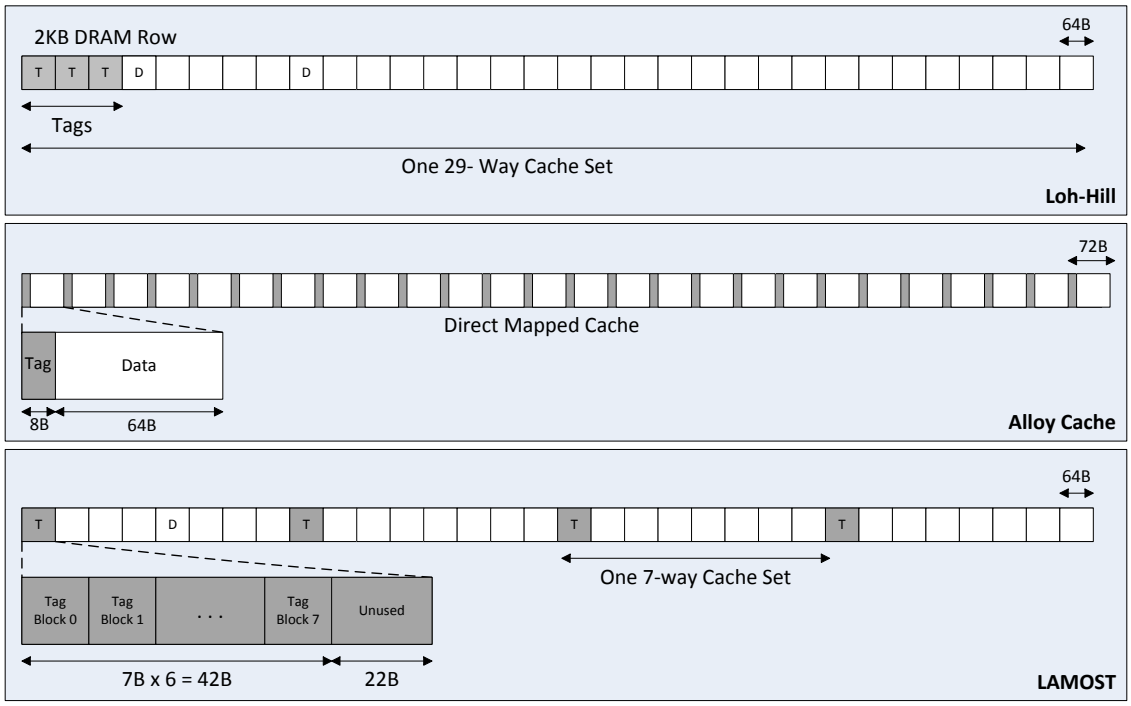

Figure 7.3: A comparison of some of the different possible row layouts for DRAM caches.

An additional potential DRAM cache device customization could be to implement wider row buffers than the standard 2KB row buffers found in most DRAM devices today. These wider row buffers could increase the chance of row buffer hits, which could provide a significant performance improvement for many potential DRAM cache designs. They could also allow for more complex data and tag layouts which could be used to increase the available associativity without increasing the number of tag accesses. For instance, in Figure 7.3 we can see several different ways

of organizing tags and data within a single DRAM cache row. One of the major limiting factors of these designs is that the standard 2KB row buffer will only hold 32 64 byte DRAM accesses so any organization has to fit neatly into that number of accesses. This limits the degree of associativity that can be implemented without introducing unused portions of the DRAM row. By allowing for wider row buffers or row buffers that are not a power of 2, more organizations and higher degrees of associativity would be possible.

However, there are several downsides to utilizing custom DRAM cache parts. One of the primary concerns facing the development of custom parts is that they would have be specially fabricated and could therefore be very expensive. This is because such special parts would not benefit from the economy of scale that helps keep most memory prices down. Also, adding wider row buffers or additional concurrency would increase the overall complexity of the device significantly and that could reduce the potential density of the devices and also increase their cost. As a result, custom DRAM cache devices could provide some benefits in terms of DRAM cache organization and performance but could also limit the capacity of the cache and greatly increase its cost. In this work, we focus on DRAM cache designs that utilize commodity parts but the potential benefits of custom DRAM cache devices make them an promising area of future research.

7.2 Preliminary Studies

In order to understand the role that associativity plays in stacked DRAM cache systems we conducted a series of simple experiments using a selection of different benchmarks from the SPEC CPU2006, NPB, and PARSEC suites. In these experiments we try to establish the role of associativity by examining how cache performance varies when both associativity and size or miss penalty are varied.

7.2.1 The Effect of Cache Size on the Impact of Associativity

Table 7.1: The differences in hit rate for different levels of associativity for a 128MB cache that is smaller than the working sets of all of the benchmarks used in this chapter. The percent improvement column indicates the improvement of each successive increase in associativity over the previous level.

Associativity	Direct	2-Way		4-Way		8-Way		16-Way		32-Way	
Benchmark	Hit %	Hit %	% Improve	Hit %	% Improve	Hit %	% Improve	Hit %	% Improve	Hit %	% Improve
ft	45.0%	46.5%	2.7%	47.3%	1.7%	47.5%	0.3%	47.5%	0.0%	47.4%	0.0%
is	37.7%	33.6%	-6.3%	30.7%	-4.2%	28.9%	-2.7%	27.9%	-1.3%	27.3%	-0.8%
mg	39.6%	45.4%	10.2%	46.4%	1.8%	46.1%	-0.6%	46.1%	-0.1%	46.0%	0.0%
blackscholes	12.4%	12.4%	0.0%	12.4%	0.0%	12.4%	0.0%	12.4%	0.0%	12.4%	0.0%
bodytrack	47.8%	53.2%	10.8%	51.9%	-2.7%	49.6%	-4.6%	47.8%	-3.6%	46.1%	-3.2%
canneal	64.6%	69.2%	13.9%	71.4%	7.3%	71.8%	1.4%	72.0%	0.9%	72.1%	0.4%
freqmine	53.3%	55.5%	4.8%	55.8%	0.8%	55.9%	0.1%	55.9%	0.1%	55.9%	0.0%
bzip2	42.2%	44.6%	4.3%	45.3%	1.3%	45.4%	0.2%	45.4%	0.0%	45.4%	0.0%
gcc	47.2%	52.0%	9.5%	53.9%	4.0%	54.9%	2.0%	55.2%	0.8%	55.2%	0.0%
leslie3d	40.6%	44.4%	6.6%	45.9%	2.6%	46.5%	1.1%	46.6%	0.2%	46.7%	0.1%
milc	33.7%	35.1%	2.2%	35.6%	0.7%	35.6%	0.1%	35.6%	0.0%	35.6%	0.0%
average	42.2%	44.7%	5.3%	45.1%	1.2%	45.0%	-0.2%	44.8%	-0.3%	44.6%	-0.3%

Table 7.2: The differences in hit rate for different levels of associativity for a 256MB cache that is only slightly smaller than the working sets of all of the benchmarks used in this chapter. The percent improvement column indicates the improvement of each successive increase in associativity over the previous level.

Associativity	Direct	2-Way		4-Way		8-Way		16-Way		32-Way	
Benchmark	Hit %	Hit %	% Improve	Hit %	% Improve	Hit %	% Improve	Hit %	% Improve	Hit %	% Improve
ft	49.9%	50.6%	1.5%	50.2%	-0.8%	49.6%	-1.2%	49.4%	-0.4%	49.3%	-0.2%
is	92.6%	91.4%	-15.2%	89.6%	-19.0%	86.6%	-25.1%	82.5%	-26.6%	76.9%	-27.5%
mg	51.5%	64.4%	30.5%	58.0%	-16.4%	54.5%	-8.1%	53.5%	-2.0%	52.7%	-1.8%
blackscholes	79.6%	76.7%	-13.3%	72.4%	-16.9%	60.7%	-35.0%	41.3%	-39.5%	22.5%	-27.6%
bodytrack	55.2%	61.4%	15.0%	62.3%	2.2%	62.3%	0.2%	62.4%	0.1%	62.4%	0.0%
canneal	84.6%	86.5%	13.2%	87.5%	7.6%	87.9%	3.6%	88.0%	0.8%	88.1%	1.0%
freqmine	60.1%	60.7%	1.5%	61.0%	0.6%	60.9%	-0.1%	60.9%	-0.1%	60.9%	0.0%
bzip2	44.0%	45.2%	2.2%	45.4%	0.4%	45.4%	0.0%	45.5%	0.0%	45.5%	0.0%
gcc	60.7%	66.4%	15.6%	67.9%	4.6%	68.5%	2.0%	68.9%	1.1%	69.0%	0.4%
leslie3d	56.8%	57.8%	2.5%	57.9%	0.0%	57.3%	-1.3%	57.1%	-0.4%	57.0%	-0.2%
milc	33.7%	35.1%	0.7%	35.6%	0.1%	35.6%	0.0%	35.6%	0.0%	35.6%	0.0%
average	60.8%	63.3%	4.9%	62.5%	-3.4%	60.9%	-5.9%	58.6%	-6.1%	56.4%	-5.1%

Comparing the values in Tables 7.1, 7.2, and 7.3, we can see that the effects of cache size and associativity vary greatly from workload to workload. For instance, canneal is sensitive to associativity at all three cache sizes. However, freqmine reacts very differently to associativity. For all three cache sizes, freqmine shows almost no effect from increasing associativity. As a result of this, the average effects of size and associativity across all benchmarks appears to indicate that associativity has an insignificant effect for all but the smallest caches. However, this obfuscates the important fact that while associativity is not always helpful on average, it is sometimes very helpful in specific cases.

Table 7.3: The differences in hit rate for different levels of associativity for a 512MB cache that is only slightly smaller than the working sets of all of the benchmarks used in this chapter. The percent improvement column indicates the improvement of each successive increase in associativity over the previous level.

Associativity	Direct	2-Way		4-Way		8-Way		16-Way		32-Way	
Benchmark	Hit %	Hit %	% Improve	Hit %	% Improve	Hit %	% Improve	Hit %	% Improve	Hit %	% Improve
ft	56.9%	57.8%	2.1%	58.3%	1.0%	59.1%	2.0%	59.6%	1.2%	59.8%	0.5%
is	95.8%	95.8%	0.0%	95.8%	0.0%	95.8%	0.0%	95.8%	0.0%	95.8%	0.0%
mg	94.8%	94.8%	0.0%	94.8%	0.1%	94.8%	0.0%	94.8%	0.0%	94.8%	0.0%
blackscholes	86.0%	86.2%	1.5%	86.2%	0.0%	86.2%	0.0%	86.2%	0.0%	86.2%	0.0%
bodytrack	60.1%	63.4%	8.7%	64.0%	1.5%	64.5%	1.4%	64.7%	0.5%	64.8%	0.3%
canneal	91.8%	93.9%	29.7%	94.7%	14.4%	95.2%	8.8%	95.4%	3.8%	95.5%	2.3%
freqmine	73.0%	73.5%	1.7%	72.5%	-3.5%	71.8%	-2.7%	70.8%	-3.5%	70.5%	-1.0%
bzip2	46.5%	46.5%	0.0%	46.0%	-1.0%	45.6%	-0.7%	45.5%	-0.2%	45.5%	0.0%
gcc	75.9%	76.1%	1.0%	76.3%	0.8%	76.3%	0.0%	76.3%	0.1%	76.3%	0.0%
leslie3d	73.0%	78.2%	21.3%	79.9%	8.0%	82.0%	11.0%	82.9%	5.2%	83.7%	4.4%
milc	35.4%	35.7%	0.4%	35.7%	0.0%	35.7%	0.0%	35.7%	0.0%	35.7%	0.0%
average	71.8%	72.9%	6.0%	73.1%	1.9%	73.4%	1.8%	73.4%	0.7%	73.5%	0.6%

7.2.2 The Effect of Miss Latency on the Impact of Associativity

Figure 7.4 displays the effects of backing store speed on the impacts of associativity. From these results it is clear that, as the ratio of backing store latency to cache latency increases, the role of associativity in preventing misses becomes more and more important. However, even at a ratio of 8x, little benefit is seen from increasing associativity beyond 4-way. The results also show that 2-way associativity provides the greatest difference in performance. Interestingly, the performance of the 8x ratio system actually degrades after peaking at 4-way associativity. Similar trends can also be noted for the other systems but to a lesser degree. To check this

somewhat odd result we reran the experiment using the Dinero cache simulator and got similar trends [103]. We believe that this behavior is due to the additional set contention that can result from increasing the associativity. Adjusting the replacement algorithm for LRU to RRIP or something similar may help to alleviate this effect.

Overall, the preliminary experiments show us several important aspects of these stacked DRAM cache systems that we must take into account when investigating their design. First, the value of associativity varies with different system parameters, unless care is taken when evaluating associative caches that value can be inadvertently diminished. In other words, if a cache is too large or too small or its backing store is not slow enough, the effects of associativity will be obscured. Second, the value of associativity varies greatly with different workloads and averaging across all workloads often conceals the real impacts of associativity. Therefore, it is important to consider each benchmark's performance individually when exploring the benefits of associativity.

7.3 Block Based Designs

One of the first DRAM cache architectures was proposed by Loh et al. and stored the meta-data for a set in the same row as the data for the set [83]. This leveraged row buffer locality to reduce the access latency of the subsequent tag and data accesses after the initial access had opened the row. However, this design used an entire DRAM row for each set and so the row buffer locality was only exploited for

a single set access. The LAMOST design improved upon Lohs design by reducing the degree of associativity from 28-way to 7-way, which allowed for 4 sets to be placed in a row instead of just one [104]. As a result, row buffer locality could be leveraged for multiple set accesses thereby improving performance. However, both of these designs still require multiple accesses to the DRAM for each cache hit and this ultimately becomes a limiting factor in the performance of system.

To address this problem of multiple DRAM accesses to fetch tags, Qureshi et al. proposed Alloy cache which stores the tag and data together and fetches them with a single 72 bit DRAM access [81]. This approach provides good performance for many workloads at the expense of higher miss rates due to the lack of associativity.

Another approach is to store a subset of tags in SRAM while the majority of tags are stored in DRAM. In effect these designs, such as [82], create an additional structure to cache the tags from the DRAM cache. In [82], this tag cache is structured and operates in much the same way as any other cache. It attempts to retain tags that have been accessed and replaces ones that aren't used. This approach achieves some considerable speedups over tags in DRAM cache but requires 46KB of SRAM for the tag cache. Another similar design which also used an SRAM tag cache was proposed in [105].

Still another way to reduce the overhead of storing meta-data is to compress it. This is explored in the Tag Tables paper which proposes a novel compression scheme for DRAM cache tags [106]. The compression scheme achieves a significant reduction in the amount of space needed to store the tags for the DRAM cache. However, despite its notable compression ratio, this design still often requires 1-2MB of tag

storage.

7.4 Page Based Designs

All of the previous approaches that we have discussed have utilized standard 64B block sized accesses in their designs. However, another way to approach building a DRAM cache is to use large pages. This is the approach that is explored by the Unison and Footprint cache designs [107,108]. Utilizing large page sized accesses results in higher hit rates and reduced tag overhead for the cache. However, additional mechanisms are required to manage the contents of the pages in the cache to prevent wasting bandwidth to fetch data that won't be used.

Another interesting page based DRAM cache design was proposed in [109]. This design utilizes the existing TLB infrastructure to keep track of the contents of the DRAM cache, essentially indexing the cache with virtual addresses.

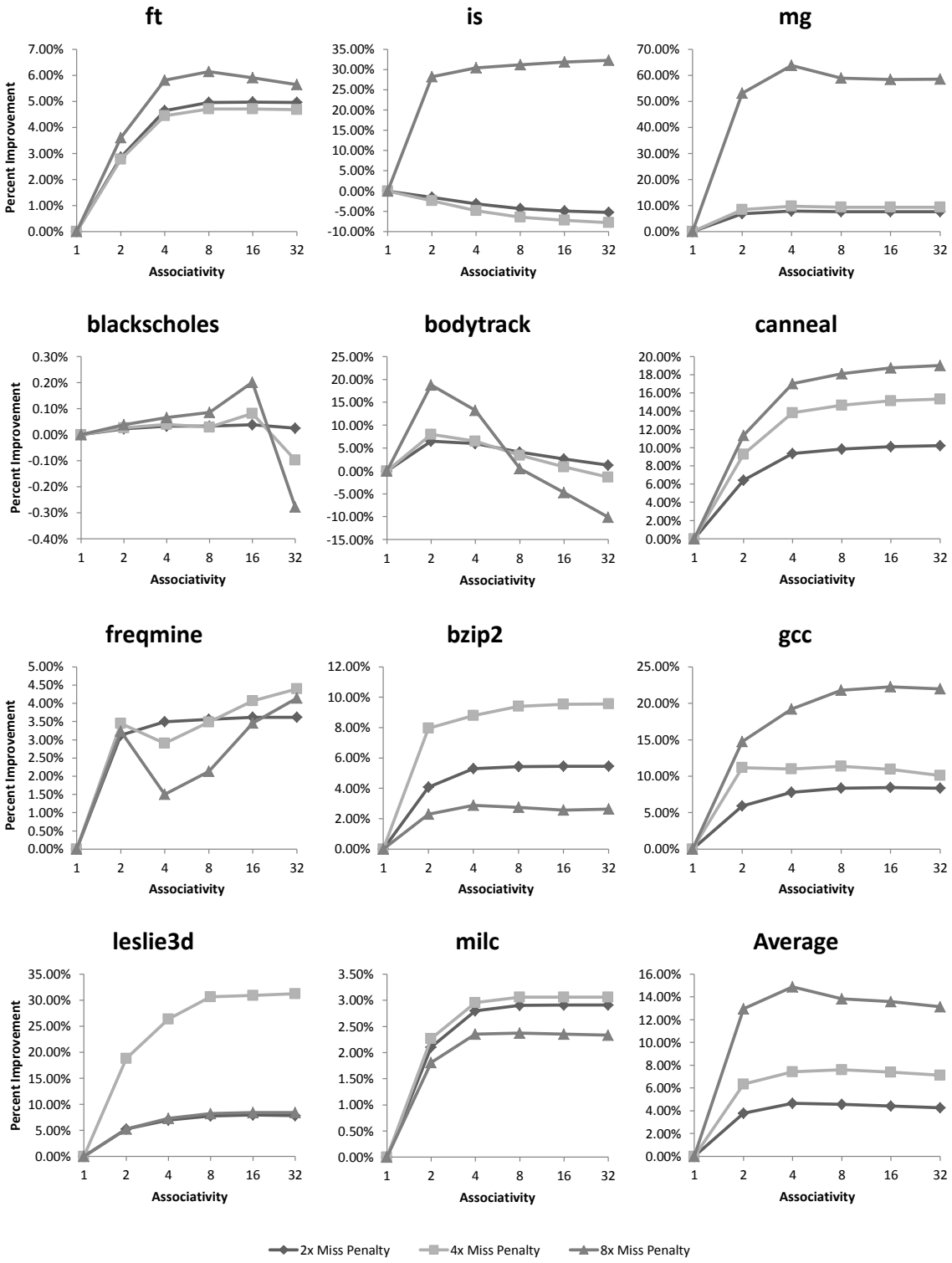

Figure 7.4: The effect of different miss penalties relative to hit latency and different degrees of associativity on the average access latency of the cache.

Chapter 8: DRAM Cache Design Analysis

In this chapter we continue the discussion on DRAM cache design by presenting our novel design for an efficient associative DRAM cache: Combo-Tag. Previously, we have talked about the importance of associativity and the difficulties of storing the tags for a large DRAM based cache. The design we present here balances the need for associativity against the necessity of maintaining a low cache hit latency. This is accomplished by organizing the layout of sets in the DRAM so that multiple set's worth of tags are combined into a single DRAM access and by utilizing a specially designed buffer to temporarily store the fetched tags in a relatively small amount of SRAM.

After presenting our design, we then perform a series of studies that compare the performance of our design to several other DRAM cache architectures. In particular, we compare our design against two versions of Alloy Cache; a serial version and an interleaved version. The serial version organizes its sets so that sequential sets will map to the same row buffer, while the interleaved version organizes its sets so that sequential sets will map to different channels. In addition, we also compare our design to LAMOST which is a 7-way associative system without a tag buffer. By comparing these different approaches we can determine the benefits of concurrency,

associativity, and compact tag buffering.

8.1 Evaluation Methodology

The primary test system used in this work is described in 8.2. This system utilizes RLDRAM as the cache technology and LPDDR as the backing store technology. Similar architectures which also use these technologies have been suggested in the literature [27, 48]. This architecture was selected because it represents a heterogeneous memory system with a large miss penalty that uses technologies with well established timing parameters. These timing parameters can be found in table 8.1 and are derived from publicly available data sheets.

For these studies we utilize a cache with a data capacity of 128MB. We use this size because we found in our preliminary work that a 256MB cache captures most of the working set of too many benchmarks. This emphasizes hit latency above all other system parameters and results in an skewed evaluation of the cache architecture that over values hit latency. By focusing on a 128MB cache, we ensure that the miss pressure will be more realistic and all aspects of the cache architecture will play a role in the evaluation. We believe that this better represents the situation that these sorts of systems will encounter in real world data center environments where working sets are often very large and miss handling plays a significant role in system performance. Futhermore, the tag buffer utilized by the combo-tag approach does not need to scale with cache size. Therefore, increasing the size of the cache would not add additional overhead to our design. In addition, for these studies the

Table 8.1: Timing Parameters [2]

Parameter	RLDRAM3	LPDDR3
tRC	12ns	64ns
tRCD	-	18ns
tRRD	-	10ns
tRP	-	18ns
tRAS	-	42ns
tFAW	-	50ns
tWTR	0ns	7.5ns
tWR	-	15ns

size of the cache only refers to the portion of the cache available for data.

Most of the studies presented in this work were performed using detailed trace based simulation using HybridSim with OMS and DRAMSim2 providing the detailed memory simulations. Each of these studies was warmed for 10 million memory transactions to eliminate any cold cache effects. The simulations were then run for an additional 100 million memory transactions or until the trace completed (this represents considerably more processor instructions).

To determine the IPC performance impact of the proposed design relative to other DRAM cache organizations we performed full system simulations using the MARSSx86 full system simulator [110]. The configuration of the DRAM cache and main memory portions are identical between the trace based and full system simulations. The full system simulations were started at the ROI of each benchmark, warmed for 100 million instructions, and then run for an additional 1 billion instructions.

Table 8.2: Baseline Simulator Configuration

Processor	
Number of cores	8-core
Issue Width	4
Frequency	3.2GHz
On Chip Caches	
L1I (private)	128 KB, 8-way, 64 B block size
L1D (private)	128 KB, 8-way, 64 B block size
L2 (private)	256 KB, 8-way, 64 B block size
L3 (shared)	32 MB, 20-way, 64 B block size
In-Package RLDRAM Cache	
Organization	128MB, 4-way, 64 B block size
Bus Frequency	DDR-3200
Bus Width	64 bits per channel
Channels	8
Ranks	1 Ranks per channel
Banks	8 Banks per rank
LPDDR Backing Store	
Organization	64 B block size
Bus Frequency	DDR-1600
Bus Width	64 bits per channel
Channels	1
Ranks	2 Ranks per channel
Banks	8 Banks per rank

8.1.1 Benchmarks

For the studies presented in this chapter we use a selection of multi-threaded workloads from the SPEC CPU2006 [114], NPB [112], and PARSEC [113] bench-

mark suites. The SPEC workloads were run in rate mode with 8 copies while the NPB and PARSEC benchmarks were run with 8 threads. Table 8.3 presents the relevant characteristics of the chosen benchmarks. These particular benchmarks were selected from the various suites because they had working sets that were significantly larger than 128MB and exhibited a particularly high MPKI.

We also include data from an additional set of benchmarks that represent real world server workloads. These workloads are included because they provide a perspective of the situation currently faced in data centers. The names of the workloads are generally illustrative of the types of activity that is taking place on the server in each case. For instance, web search is a trace representing the memory traffic of a server performing tasks for a search engine, internet support is a trace representing a server running an internet based customer support application, etc.

8.2 DRAM Set Layout

One of the central innovations of the Combo-Tag design is the way in which the tags and data are laid out in the DRAM. The key observation behind the layout strategy is that if 4-way associativity is used then more than one set's tags can fit in a single DRAM accesses. We leverage this in Combo-Tag by coalescing the tags of multiple sets in a single DRAM access. This allows us to amortize the cost of fetching tags from the DRAM because we always get more than one set's worth of tags on a tag fetch. We then store the extra tags in a small SRAM based tag buffer that enables us to often quickly look up tags and avoid a DRAM access for tags.

Table 8.3: Benchmark Characteristics

	Observed Footprint	**L3 MPKI**
ft	1287.27 MB	7.115501
is	264.87 MB	10.424386
mg	430.87 MB	13.5972049
blackscholes	269.53 MB	0.6400292
bodytrack	534.28 MB	0.4995586
canneal	497.60 MB	6.50089792
freqmine	894.04 MB	1.31842398
bzip2	2559.93 MB	61.0071412
gcc	441.31 MB	6.0996348
leslie3d	601.15 MB	18.724911
milc	1909.66 MB	22.4704558

We can fit multiple sets worth of tags in a single DRAM access in our approach because each tag + other meta-data is less than 4 bytes in size. Figure 8.1 shows how the tags and data are laid out in a row and compares our design to several other designs that have been proposed in the literature. Figure 8.2 provides sizes for the various components of the meta-data and shows how 4 sets worth of tags can fit in a single 64B DRAM access. In addition, in order to leverage the spacial locality of tags, we group sets sequentially per DRAM row. This helps to ensure that the tag buffer will have an acceptable hit rate. However, to leverage the concurrency available in the system, we interleave the rows of sets across the channels, ranks and banks. So, sets 0-6 will map to channel 0 but sets 7-13 will map to channel 1. This is critical to the performance of the system as concurrency is a major source of performance in the

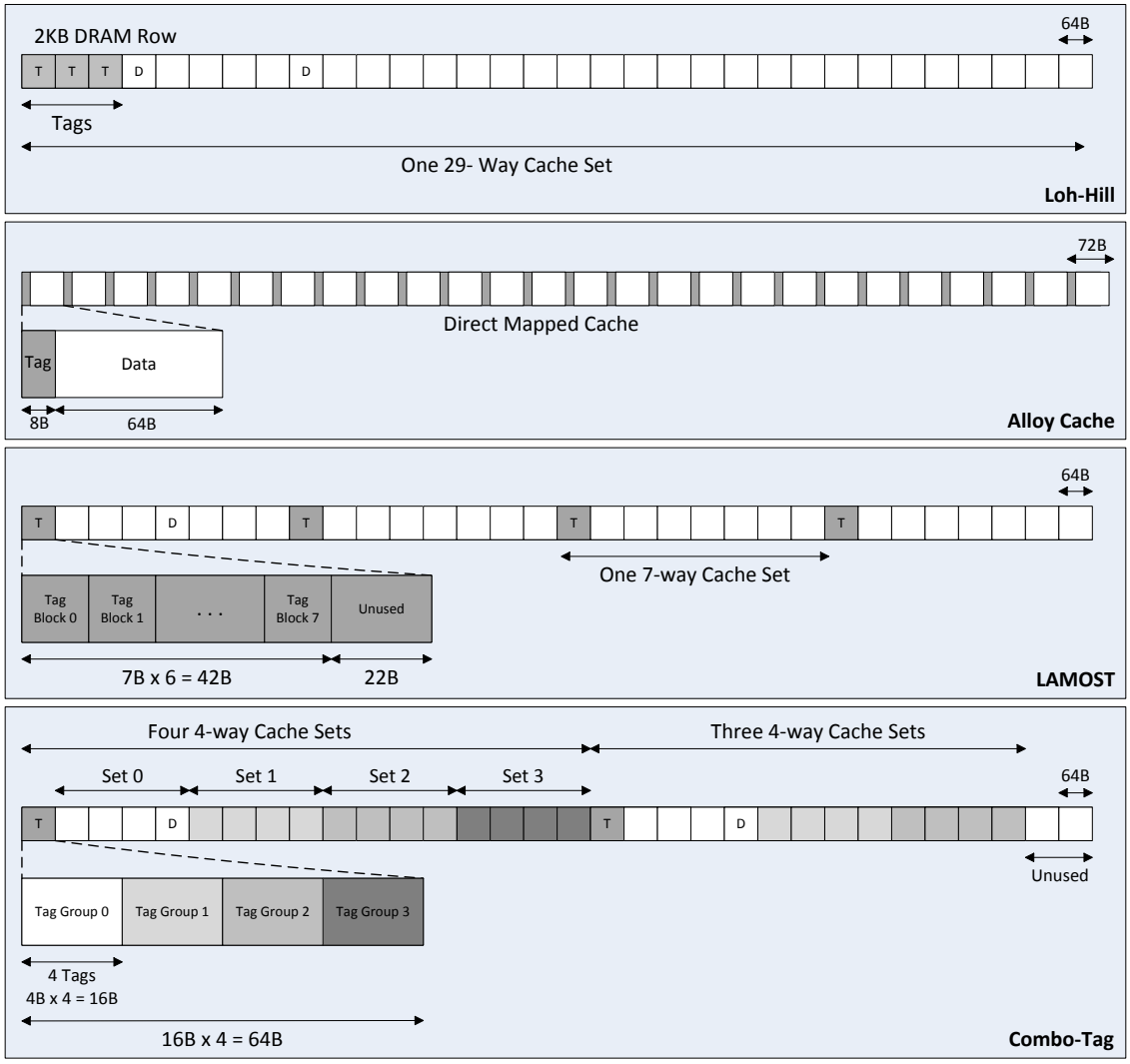

Figure 8.1: The DRAM row layouts of different DRAM cache approaches.

DRAM cache and so mapping sets sequentially results in a significant reduction in performance. We will see this effect later in the results section. We also investigated fully interleaving the sets across channels in the Combo-Tag design so that set 0 would map to channel 0 and set 1 would map to channel 1 and so forth. But this resulted in a significant drop in tag buffer hit rate and unacceptable performance. Finally, in all of the associative DRAM cache designs there is some lost DRAM

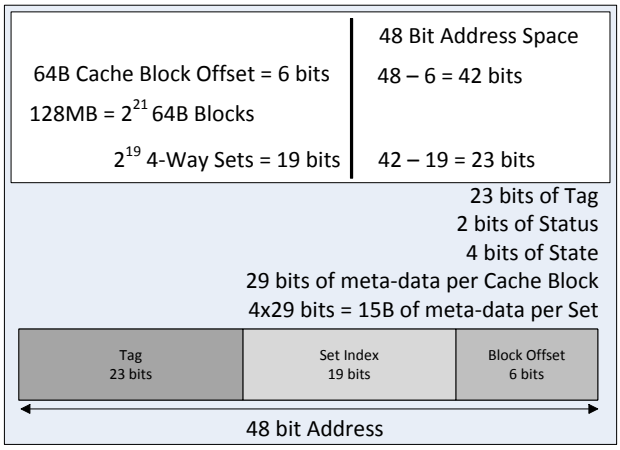

Figure 8.2: An explanation of the sizes of tags and other meta-data.

capacity due to the tags taking up space in the cache. Our design has roughly the same meta-data storage overhead as the other designs.

There are two minor drawbacks to our method of mapping sets into the DRAM row. First, we are unable to use 2 columns per standard DRAM row due to the size of the rows. We can fit 7 4-way sets in a single 2KB DRAM row (28 of 32 available columns) because we need to reserve several DRAM columns for tags. By coalescing the tags we can reduce the required number of tag columns to 2 which leaves 2 unused columns in the row. These columns could potentially be used to store other information, though, and so represent an interesting area for future research. Alternatively, the DRAM row could be expanded so that it would contain 3 additional columns for tags. This would allow for 8 sets of data to be stored per row and would eliminate the need to leave columns unused. This would not be an overly difficult change to make since DRAM caches will almost certainly need to be built using custom devices anyway. It is also worth mentioning that using 2-way

sets allows for full utilization of a DRAM row at the cost of some performance. Second, we are only able to efficiently accommodate up to 8-way associativity with this scheme. Beyond 8-way associativity, we can no longer fit more than one set's worth of tags into a single DRAM access and the benefits of coalescing tags are lost.

8.3 Tag Buffer Design

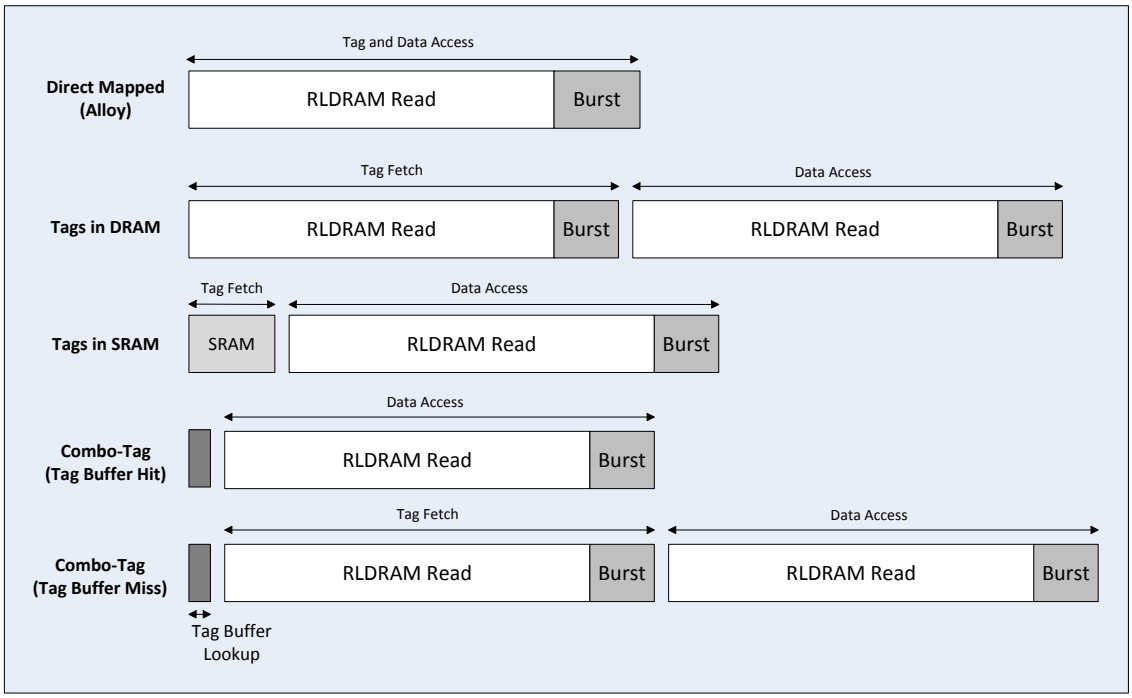

Figure 8.3: The access latencies of different DRAM cache approaches (not to scale).

In order to take advantage of the additional sets of tags that are fetched on each tag access we implement a small prefetched tag buffer which stores the tags temporarily. We implement the tag buffer as a cache-like structure that stores each set's tags as a separate tag group entry. In other words, the total tags for each set are stored as a single block in the cache and each set's tags (a tag group) are

managed independently of the others. In our design multiple sets worth of tags are fetched with each DRAM tag access so each DRAM tag access results in multiple evictions from the tag buffer. However, each entry is a separate set's tags so the entries which are evicted could have been from different DRAM tag accesses.

Figure 8.3 presents the access process of several different DRAM cache architectures. For Combo-Tag, some accesses will result in a hit in the tag buffer and so the tag lookup operation will occur very quickly. We assume that because the SRAM tag buffer is so small, it will be possible to access it in roughly 1 ns. In contrast, the accesses to a large SRAM tag store that contains all of the tags will take considerably longer. These assumptions are based off of similar CACTI-based calculations performed in [82]. Therefore, by utilizing the tag buffer, we can have some tag accesses complete essentially instantaneously. But in order for the tag buffer and coalesced DRAM tag accesses to be worth while, the tag buffer must have a decent hit ratio. This is because a miss in the tag buffer results in a DRAM tag access process that takes slightly longer than a standard tags in DRAM access.

Initially, however, it was not clear how this buffer should be structured in order to maximize its hit rate. For instance, should the buffer simply be a direct mapped cache or was associativity necessary at this level of the design as well. To investigate the impact of size and associativity on tag buffer performance, we performed a series of experiments. The results of these experiments are summarized in Figure 8.4.

From the data presented in Figure 8.4, we can see that 256 tag group entries (or 4KB) is enough capacity to provide roughly a 40% hit rate for the tags on average across all benchmarks. Some additional benefit can be achieved by increasing the size

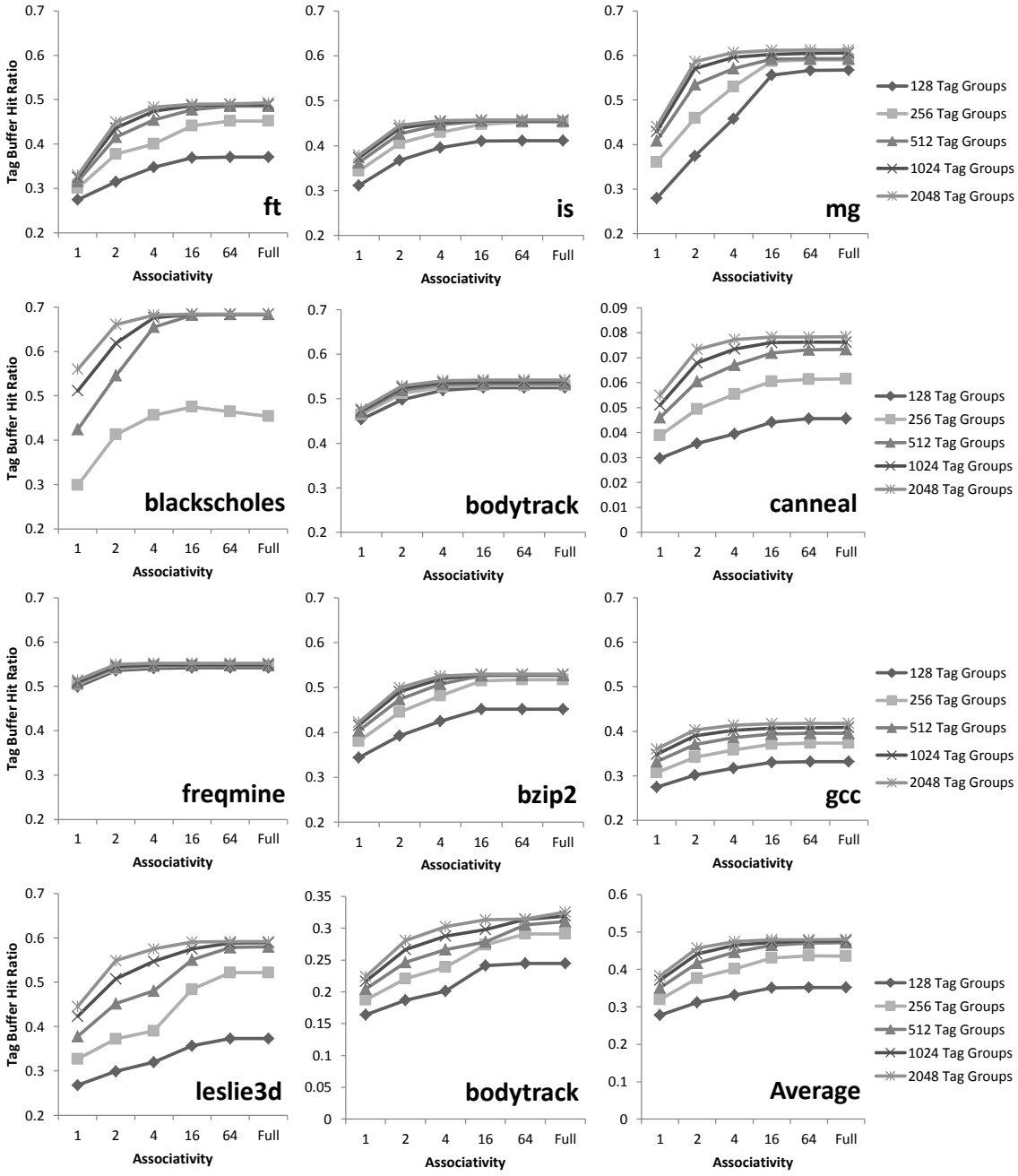

Figure 8.4: The effect of size and associativity on the tag hit rate of the tag buffer.

of the tag buffer but 256 entries represents the best balance of size to performance. Associativity also plays an important role in the performance of the tag buffer. We find that 16-way set associativity represents the best balance between complexity and performance.

Another interesting aspect of these experiments is that the impact of size and associativity vary greatly depending on the workload. Some workloads, like mg, are very sensitive to changes in size or associativity while others, like freqmine, are not. Also, while most workloads see a tag hit rate of greater than 50%, some workloads, such as caneal, have very low tag hit rates that drag down the average hit rate. Therefore, for many workloads our tag buffer achieves a hit rate comparable to the much larger tag cache presented in [82]. Also, unlike a traditional cache, the tag buffer's hit rate does not vary with the size of the DRAM cache, it is purely a product of the spatial locality of the workloads. Therefore, while other designs must scale their tag stores to accomodate larger caches, Combo-Tag's tag buffer does not. Instead, the tag buffer will need to be scaled in proportion to the number of threads that can run in a system in order to capture enough of the spatial locality of each thread.

8.4 Tag Buffer Management

In addition to the physical structure of the tag buffer, the way in which the buffer is managed can also have a significant effect on its hit rate. To maximize the performance of the tag buffer, we develop two novel replacement policies that

Algorithm 1 UF-LRA/NNN Replacement Algorithm

```
1:  found_used ← false
2:  victim_pointer ← first tag_group in tag_set
3:  for each tag_group in tag_set do
4:      if found_used = false then
5:          if i.used = true then
6:              found_used ← true
7:              victim_pointer ← i
8:          else if i.age = victim_pointer.age then
9:              victim_pointer ← i
10:     else if i.age = victim_pointer.age then
11:         victim_pointer ← i
```

take into account the unique nature of the tag buffer's contents. We base our first replacement policy design for the tag buffer on the observation that the likelihood is quite low that a tag will be accessed again after it is used. As a result of this it is typically not useful to keep tags around in the tag buffer after they have been used. To take advantage of this, we propose a Used First - Least Recently Accessed (UF-LRA) replacement policy which prefers to evict the oldest used tags first. If no used tags are available, then the oldest unused tag is evicted. A pseudo code representation of our algorithm can be found in Algorithm 1. This policy results in up to a 12% improvement in hit rate in the tag buffer versus the next best policy which is a basic FIFO queue.

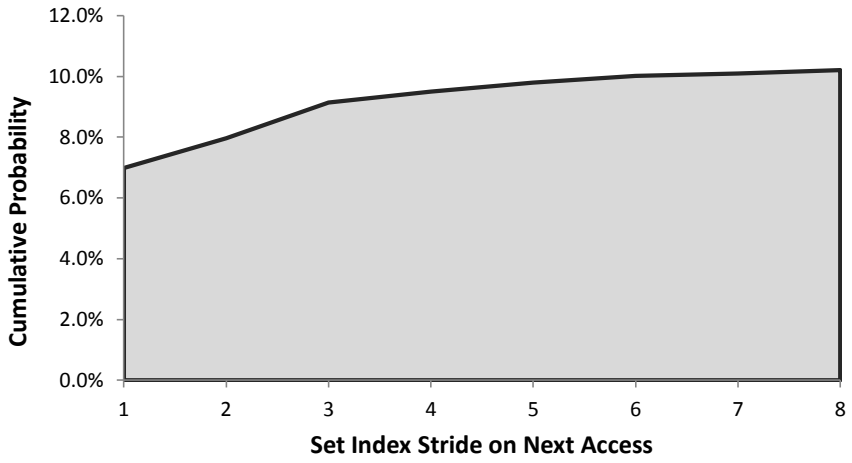

Figure 8.5: The probability of a next accesses arriving with an address that is a stride of 8 or less away from the last access.

The second replacement policy that we propose is based upon the observation that, while tags are not often reused, they do tend to display spatial locality. By this we mean that if a set is accessed, it is highly likely that a nearby adjacent set will

Algorithm 2 NNN Update Algorithm

```
1: for each set in set_accessed+1 : set_accessed+8 do
2:     for each tag_group in tag_set for set i do
3:         if j = i then
4:             j.age is reset
```

also be accessed in the near future. In our studies we found there was a roughly 10% chance that the very next access to the cache would hit a set index that was within a distance of 8 from the previously accessed set index. To leverage this fact, we modify the UF-LRA policy described above to also update the eviction priority of the next 8 sets when a set is accessed (provided that the tag groups for those sets are present in the tag buffer). This ensures that if the tags for any nearby sets are present in the tag buffer, then they won't be evicted in the near future, even if they've been present in the tag buffer for a long time. The process of searching and updating the nearby sets is expensive but it can be done in the background and should not affect the critical path of the system. This policy provides an additional 3-5% improvement in tag hit rate. The differences between the policies mentioned here are for a very small 256 entry tag buffer. These differences vary with the size and associativity of the tag buffer. If the size of the tag buffer is increased, eventually, it becomes large enough to accommodate enough tags to make the replacement policy irrelevant. Similarly, if the tag buffer is too small then the replacement policy also becomes irrelevant as the pressure on the tag buffer is too great. The 256 entry buffer used in these evaluations was selected because it represents the most economical tag buffer solution as determined by the physical structure experiments discussed earlier. A

comparison of the tag hit rates of several different replacement policies can be found in Table 8.4.

8.5 Miss Map and Compression

Unlike other similar DRAM cache designs, our design does not currently include a miss map or incorporate any tag compression schemes. The miss map and compression techniques described in several other papers are orthogonal to this work. We chose not to include them in these evaluations in order to keep the focus on the design decisions being investigated and to avoid any noise that those approaches might have introduced to the results. We leave investigations on the interaction between our proposed approach and those techniques to future work.

8.6 Trace Based Design Comparison

8.6.1 Standard Benchmarks

Figures 8.6, 8.7, 8.8, and 8.9 compare several of the different types of DRAM cache architectures that we have discussed in this work. From these results we can see that Combo-Tag does quite well when compared to the other approaches. The 4-way version of Combo-Tag is as fast or faster than the direct mapped approaches in all but two cases. Furthermore, in several cases, such as mg and blackscholes, the 4-way Combo-Tag approach is roughly 30% faster. Interestingly, the speedup of the Combo-Tag approach does not seem to be the result of the improvement in hit rate. As we can see in Figure 8.7, the addition of associaitivity only really seems to benefit

Table 8.4: Percentage improvement of replacement policies compared to LRU

		LRU	Random	MRU	FIFO	UF_LRA	NNN
ft	128 Entry	-	-97.50%	-97.32%	1.25%	2.04%	5.01%
	256 Entry	-	-91.96%	-91.35%	2.01%	3.12%	5.63%
is	128 Entry	-	-165.76%	-165.23%	0.57%	0.82%	7.64%
	256 Entry	-	-125.61%	-124.47%	0.23%	0.04%	0.94%
mg	128 Entry	-	-187.57%	-187.32%	2.79%	3.92%	4.96%
	256 Entry	-	-164.79%	-163.98%	0.28%	0.21%	0.38%
blackscholes	128 Entry	-	-180.83%	-176.55%	25.16%	26.48%	14.06%
	256 Entry	-	-197.69%	-196.29%	20.36%	31.72%	34.90%
bodytrack	128 Entry	-	-45.83%	-45.67%	0.07%	-0.63%	-0.39%
	256 Entry	-	-28.00%	-27.80%	0.01%	-0.35%	-0.15%
canneal	128 Entry	-	-174.69%	-174.66%	0.10%	-0.26%	2.38%
	256 Entry	-	-161.58%	-161.52%	0.13%	-0.49%	1.14%
freqmine	128 Entry	-	-28.99%	-28.90%	0.05%	0.36%	0.49%
	256 Entry	-	-18.69%	-18.47%	0.02%	0.09%	0.12%
bzip2	128 Entry	-	-140.65%	-140.42%	2.38%	5.92%	10.97%
	256 Entry	-	-119.76%	-119.04%	0.64%	1.46%	1.99%
gcc	128 Entry	-	-86.37%	-86.25%	0.78%	2.62%	5.62%
	256 Entry	-	-79.14%	-78.85%	0.40%	1.67%	3.35%
leslie3d	128 Entry	-	-103.20%	-103.15%	4.69%	10.30%	17.34%
	256 Entry	-	-111.19%	-110.76%	3.59%	7.91%	12.27%
milc	128 Entry	-	-133.20%	-132.59%	2.45%	5.84%	10.93%
	256 Entry	-	-120.36%	-119.26%	0.96%	2.23%	3.79%
average	128 Entry	-	-122.23%	-121.64%	3.66%	5.22%	7.18%
	256 Entry	-	-110.80%	-110.16%	2.60%	4.33%	5.85%

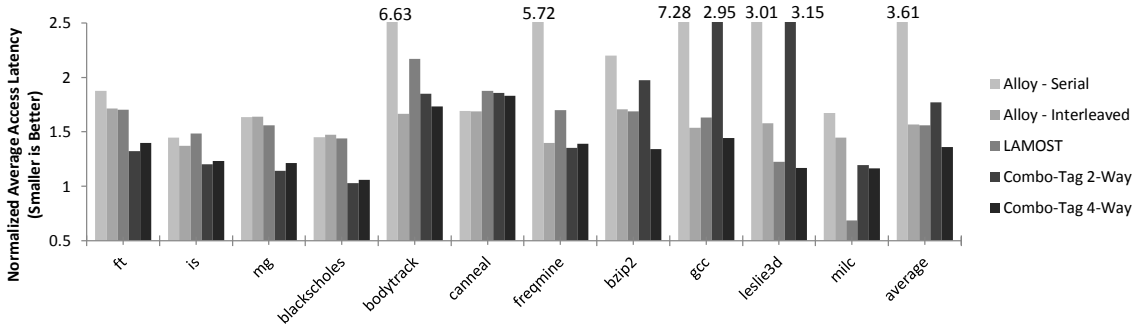

Figure 8.6: A comparison of the average access latencies of different DRAM cache architectures for normalized to an ideal system with 4-way associativity and SRAM latency tag lookups. These results are for a system with an RLDRAM cache and a LPDDR backing store. The selected benchmark suite workloads are used for this evaluation.

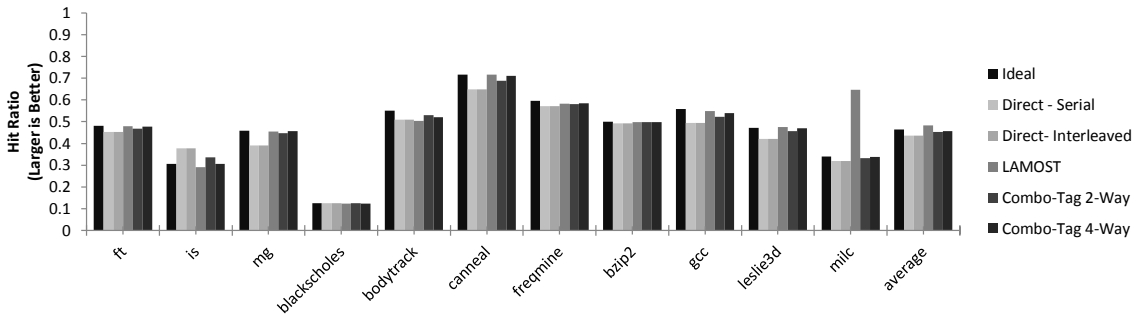

Figure 8.7: A comparison of the hit rates of different DRAM cache architectures. The ideal system has 4-way associativity and SRAM latency tag lookups. These results are for a system with an RLDRAM cache and a LPDDR backing store. The selected benchmark suite workloads are used for this evaluation.

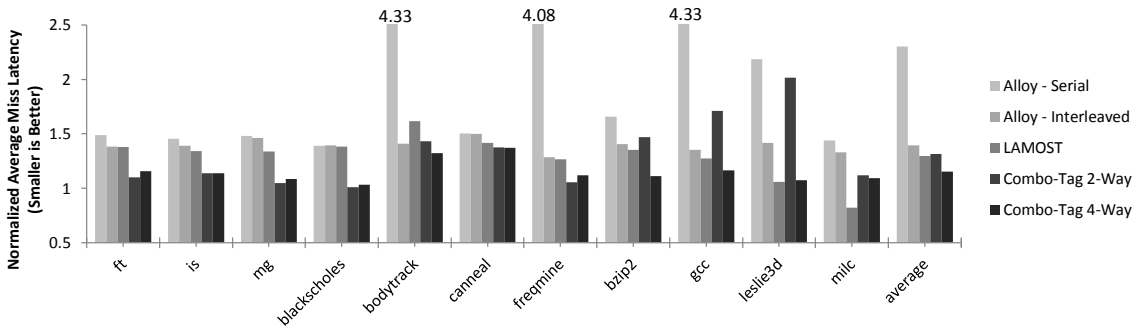

Figure 8.8: A comparison of the average miss latencies of different DRAM cache architectures normalized to an ideal system with 4-way associativity and SRAM latency tag lookups. These results are for a system with an RLDRAM cache and a LPDDR backing store. The selected benchmark suite workloads are used for this evaluation.

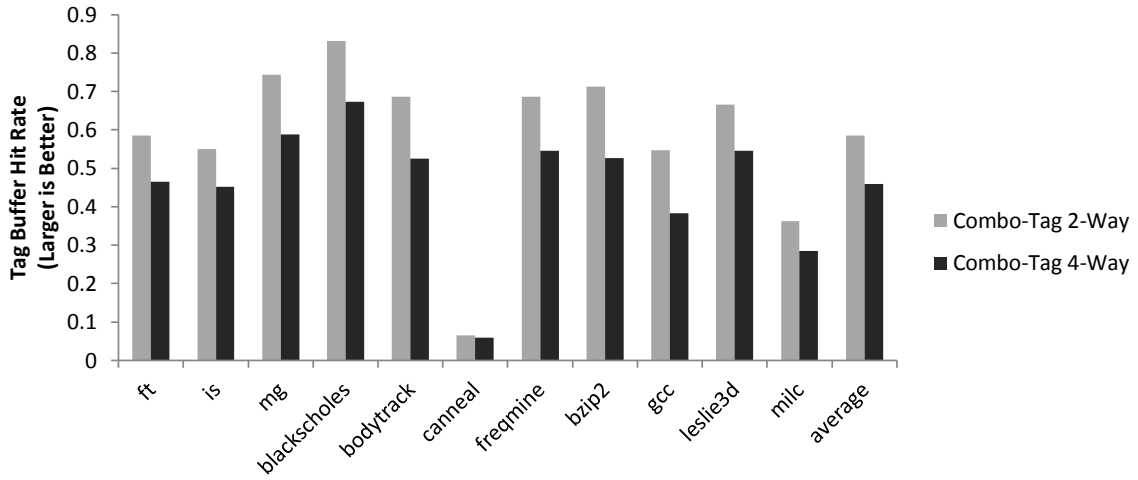

Figure 8.9: A comparison of the tag buffer hit rate of the two Combo-Tag cache architectures. These results are for a system with an RLDRAM cache and a LPDDR backing store. The selected benchmark suite workloads are used for this evaluation.

milc. Instead, it appears to be the case that the speedup is due to the reduction in miss latency that we can see in Figure 8.8. We believe that this reduction in miss latency in the Combo-Tag case is due to the tag buffer which allows some misses to be detected without an access to the DRAM cache thereby reducing the miss latency even compared to the Alloy cache approach. The tag buffer hit rate is higher for the 2-way case in Figure 8.9 because it is possible to fit more tags into a single DRAM access in the 2-way organization. This results in additional tag prefetches and improves the odds of getting a useful set of tags. Also, because the sets are half as large, we can fit twice as many sets worth of tags into the same size tag buffer compared to the 4-way case.

It is also interesting to note the severe performance difference between the serial and interleaved versions of the direct mapped DRAM cache. This may be partially due to the lack of a row-buffer in RLDRAM, which eliminates the primary reason for organizing the sets sequentially within rows. However, it also emphasizes the importance of concurrency in the effective utilization of these DRAM caches.

8.6.2 Server Benchmarks

The benchmarks in Figures 8.10, 8.11, 8.12, and 8.13 represent real-world server workloads and demonstrate the increased importance of associativity in real-world applications. We can see that these benchmarks are much more sensitive to associativity by comparing the change in hit ratios in Figure 8.11 to those in Figure 8.7. As a result of this sensitivity to associativity, we see even larger improvements

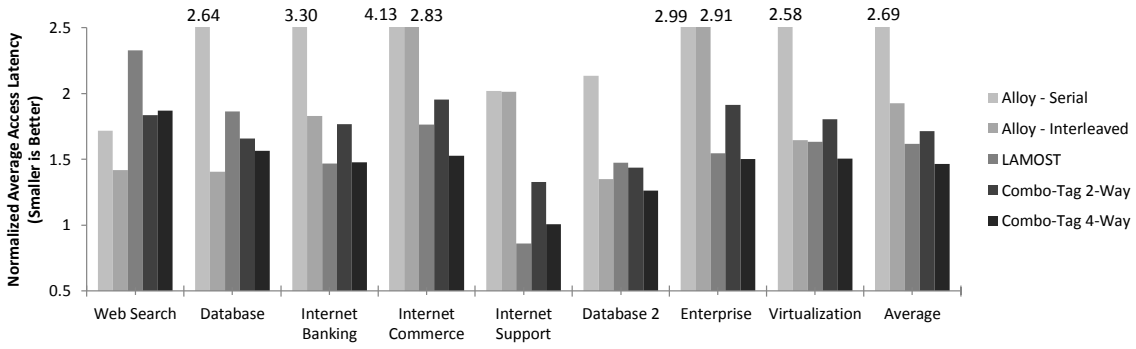

Figure 8.10: A comparison of the average access latencies of different DRAM cache architectures normalized to an ideal system with 4-way associativity and SRAM latency tag lookups. These results are for a system with an RLDRAM cache and a LPDDR backing store. The server trace workloads are used for this evaluation.

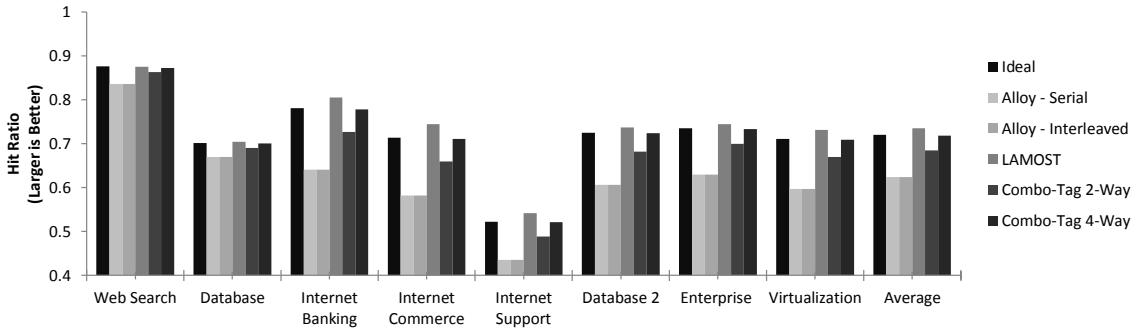

Figure 8.11: A comparison of the hit rates of different DRAM cache architectures. The ideal system has 4-way associativity and SRAM latency tag lookups. These results are for a system with an RLDRAM cache and a LPDDR backing store. The server trace workloads are used for this evaluation.

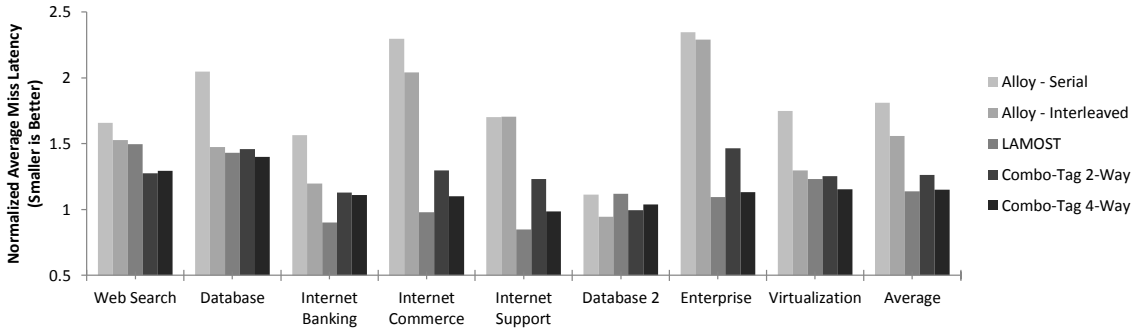

Figure 8.12: A comparison of the average miss latencies of different DRAM cache architectures normalized to an ideal system with 4-way associativity and SRAM latency tag lookups. These results are for a system with an RLDRAM cache and a LPDDR backing store. The server trace workloads are used for this evaluation.

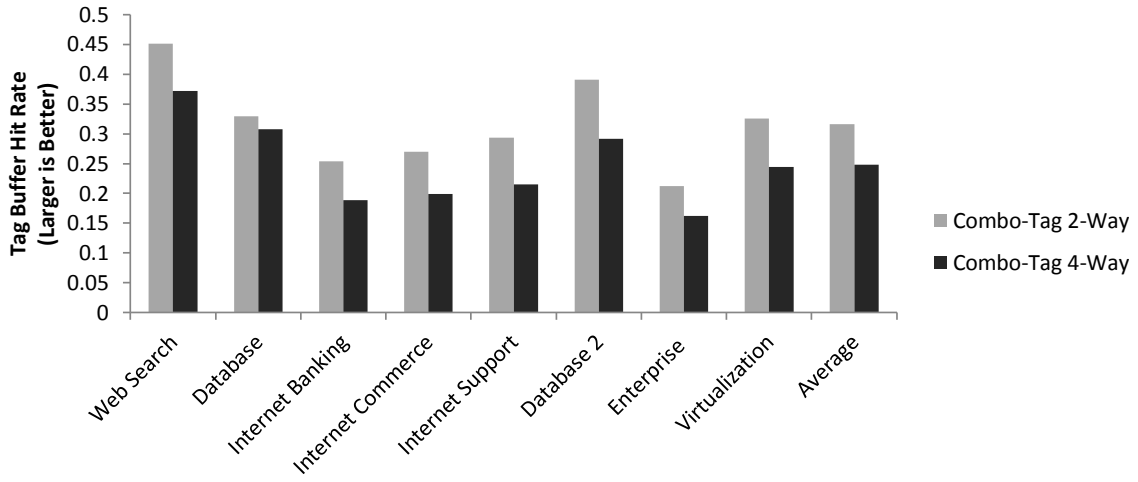

Figure 8.13: A comparison of the tag buffer hit rate of the two Combo-Tag cache architectures. These results are for a system with an RLDRAM cache and a LPDDR backing store. The server trace workloads are used for this evaluation.

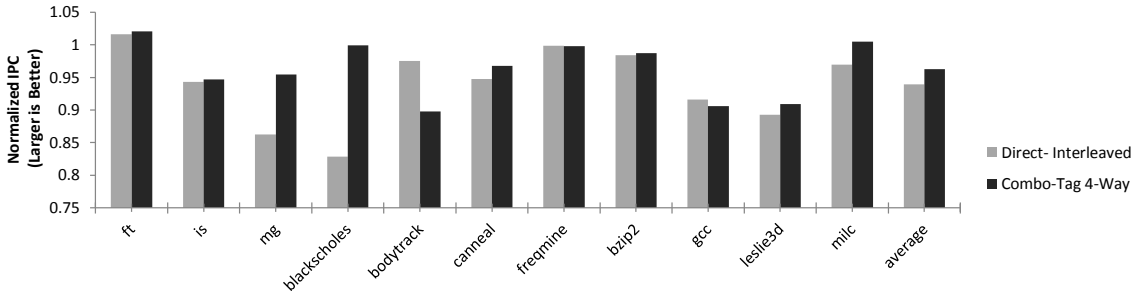

Figure 8.14: A comparison of the IPC of different DRAM cache architectures normalized to an ideal system with 4-way associativity and SRAM latency tag lookups. These results are for a system with an RLDRAM cache and a LPDDR backing store. They were produced with a full system simulation using MARSSx86.

for the 4-way Combo-Tag approach in these results. Both commerce and support see around a 45% improvement while enterprise shows a 60% improvement. Meanwhile, there is only one case where the interleaved Alloy cache approach outperforms Combo-Tag.

These results seem to suggest that Combo-Tag could significantly improve performance for many real-world workloads because it can significantly reduce the average access latency seen by those workloads. However, full system simulation is needed to accurately gauge the effect that those latency reductions will have on actual workload performance.

8.7 Full System Design Comparison

The results of our full system simulation experiments can be found in Figure 8.14. These results confirm that Combo-Tag does provide a considerable perfor-

mance speedup for some workloads. The workloads mg and blackscholes in particular show considerable IPC speedups of 11% and 18% respectively. There are a few benchmarks that show a slight slowdown compared to Alloy cache and one benchmark, bodytrack, that shows a significant slowdown. This is primarily due to bodytrack's low sensitivity to associativity. However, for the most part the performance of the Combo-Tag architecture is either comparable to or better than the direct mapped architecture's performance. This fits the expected effects of associativity, which will benefit some workloads a great deal and others not at all.

8.8 Alternative Technology and Organization Choices

In addition to the primary RLDRAM-LPDDR results that we have already presented in this chapter, we also investigated several other memory technology choices and organizations in our attempts to understand the dynamics of this system. In particular, we looked at the effects of adding or removing complexity from the backing store, of using standard DDRx DRAM at both levels of the hierarchy, and of using an open or closed page cache. In the next sections we present the trace based results for these different organizations for the selected benchmark suite workloads.

8.8.1 Simplified LPDDR Backing Store

In the primary results, we use an LPDDR backing store that has 2 ranks. This allows the backing store to perform more operations per unit time than if it only had 1 rank but also adds some complexity and potential cost to the backing store.

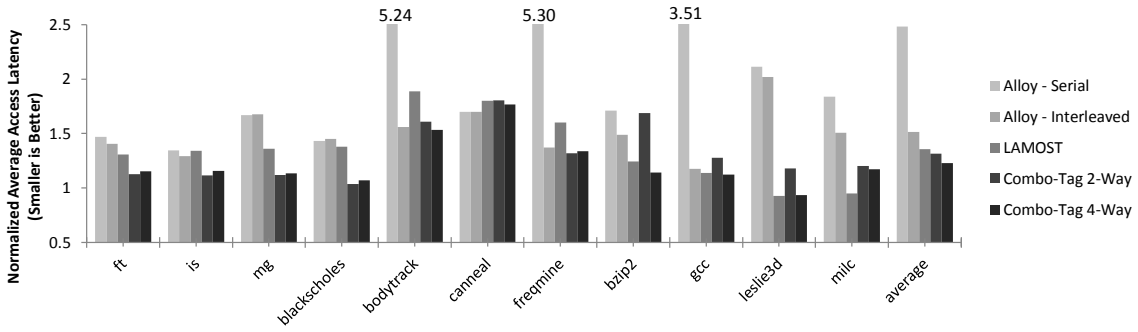

Figure 8.15: A comparison of the average access latencies of different DRAM cache architectures normalized to an ideal system with 4-way associativity and SRAM latency tag lookups. These results are for a system with an RLDRAM cache and a LPDDR backing store (with 1 Rank). The selected benchmark suite workloads are used for this evaluation.

Reducing the available concurrency in the backing store could result in longer access latencies due to increased resource contention but only if the access pressure is high enough to utilize the all of the available concurrency. In many situations it might simply be the case that there are not enough outstanding accesses to warrant the additional rank. To investigate the impact of reducing the available concurrency we performed an experiment where we left all of the system parameters the same as in the earlier experiments but reduced the backing store from 2 ranks to 1 rank (keeping overall system capacity constant). The results for this experiment are presented in Figures 8.15, 8.16, 8.17, and 8.18.

At first glance the results in Figure 8.15 seem to indicate that switching to 1 rank has improved the performance of the system, since the values are closer to ideal in this case. However, this is just the product of the increased miss latency slowing

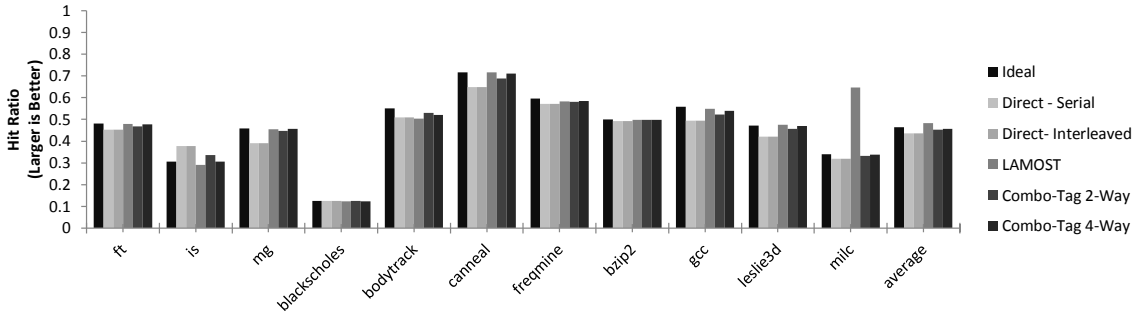

Figure 8.16: A comparison of the hit rates of different DRAM cache architectures. The ideal system has 4-way associativity and SRAM latency tag lookups. These results are for a system with an RLDRAM cache and a LPDDR backing store (with 1 Rank). The selected benchmark suite workloads are used for this evaluation.

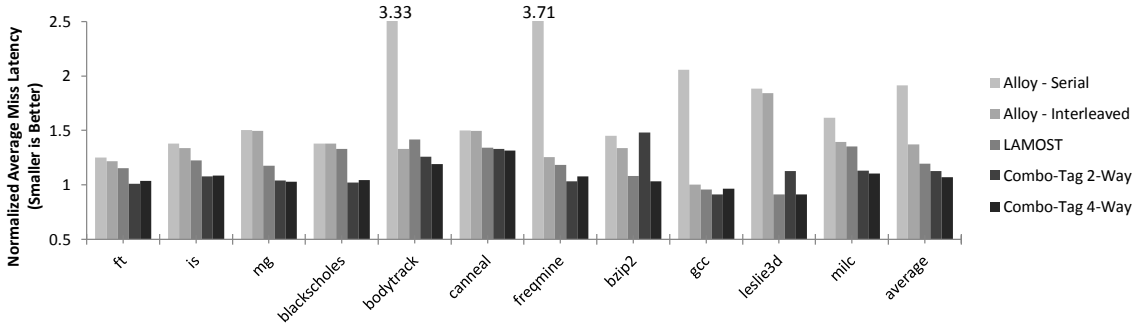

Figure 8.17: A comparison of the average miss latencies of different DRAM cache architectures normalized to an ideal system with 4-way associativity and SRAM latency tag lookups. These results are for a system with an RLDRAM cache and a LPDDR backing store (with 1 Rank). The selected benchmark suite workloads are used for this evaluation.

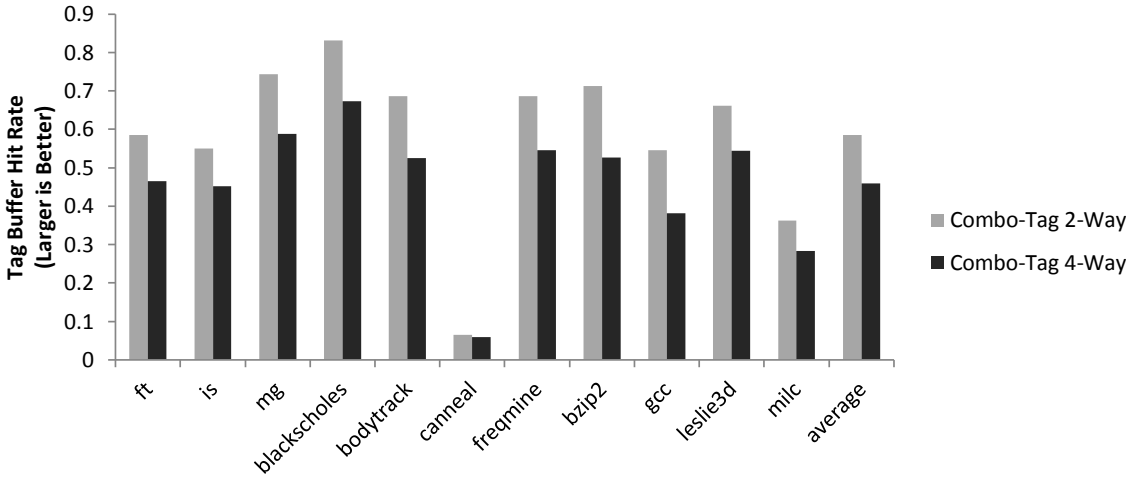

Figure 8.18: A comparison of the tag buffer hit rate of the two Combo-Tag cache architectures. These results are for a system with an RLDRAM cache and a LPDDR backing store (with 1 Rank). The selected benchmark suite workloads are used for this evaluation.

down all of the systems and reducing the significance of the design differences in some cases. In other words, the slower backing store has hindered the performance of the ideal case and made it easier for the other designs to approach its performance as a result. On average, the 4-way Combo-Tag design achieves roughly the same speedup compared to the Alloy cache as it did in the 2 rank case. The increase in miss latency, however, has led to an increase in the importance of lowering miss rates. As a result, the higher associativity of the LAMOST design allows it to achieve a performance that is closer to that of the Combo-Tag 4-way system.

8.8.2 DDR DRAM Cache and Backing Store

Most of the existing studies on DRAM caches utilize only the standard DRAM technology for both the cache and backing store levels. This different technology choice results in hit and miss latencies that are much closer and should theoretically result in a smaller impact from associativity due to the reduced importance of miss avoidance. To analyze if combo-tag would still perform well against the other state-of-the-art DRAM cache designs in this environment we also performed an experiment where the cache and backing store had the same timings derived from a typical DDR3 SDRAM part. The organization of the system in this experiment is exactly the same as in the other experiments. Another interesting side effect of utilizing DDR DRAM for the cache technology is the introduction of the possibility to utilize the cache in an open page configuration to take advantage of row buffer hits. We present the results for this experiment in Figures 8.19, 8.20, 8.21, and 8.22.

From the results in Figure 8.19 we can see that even with the relatively similar hit and miss latencies of this system, the Combo-Tag design still provides some performance benefits over the other designs. The open page policy of the cache is at least partially responsible for this because it allows for some cache accesses to be relatively fast compared to the backing store latency thereby increasing the relative miss penalty. Interestingly, the 2-way design outperforms the 4-way design in this case. This highlights the importance of the tag buffer in this organization and its ability to reduce load on the cache. Each hit in the tag buffer means that the relatively slow DDR3 cache does not need to be accessed in order to retrieve

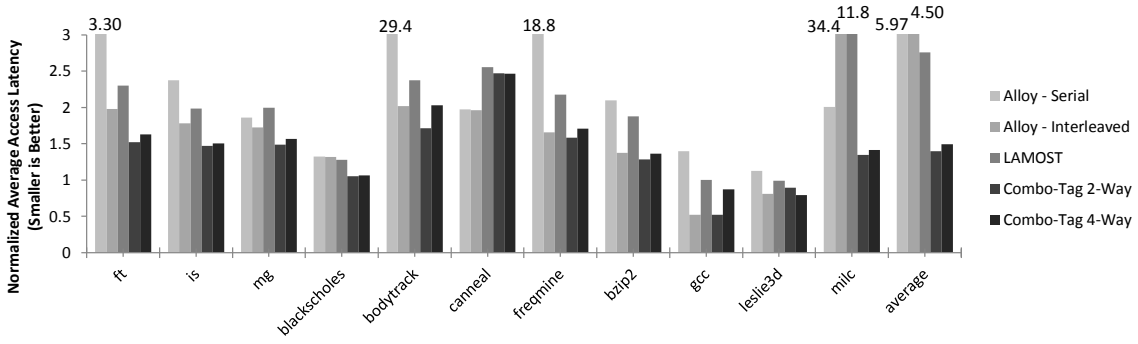

Figure 8.19: A comparison of the average access latencies of different DRAM cache architectures normalized to an ideal system with 4-way associativity and SRAM latency tag lookups. These results are for a system with an open-page DDR3 DRAM cache and a DDR3 DRAM backing store. The selected benchmark suite workloads are used for this evaluation.

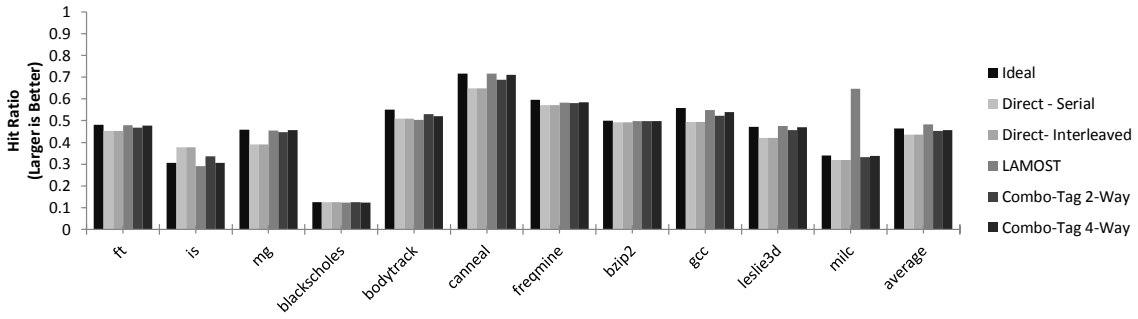

Figure 8.20: A comparison of the hit rates of different DRAM cache architectures. The ideal system has 4-way associativity and SRAM latency tag lookups. These results are for a system with an open-page DDR3 DRAM cache and a DDR3 DRAM backing store. The selected benchmark suite workloads are used for this evaluation.

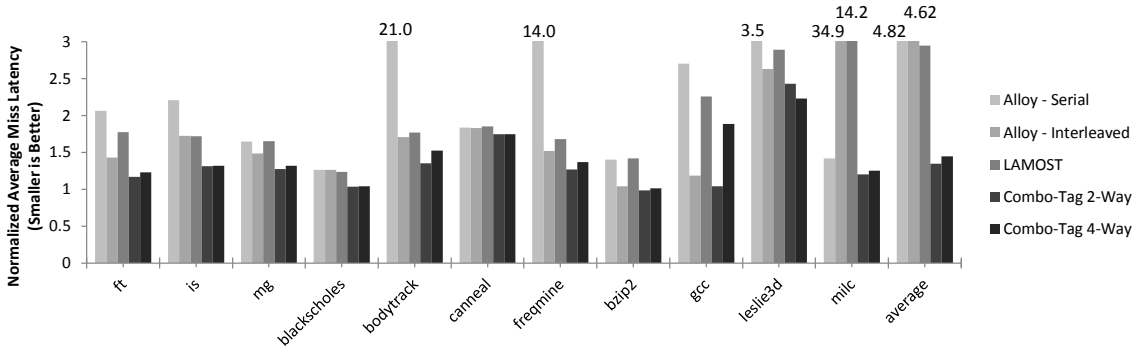

Figure 8.21: A comparison of the average miss latencies of different DRAM cache architectures normalized to an ideal system with 4-way associativity and SRAM latency tag lookups. These results are for a system with an open-page DDR3 DRAM cache and a DDR3 DRAM backing store. The selected benchmark suite workloads are used for this evaluation.

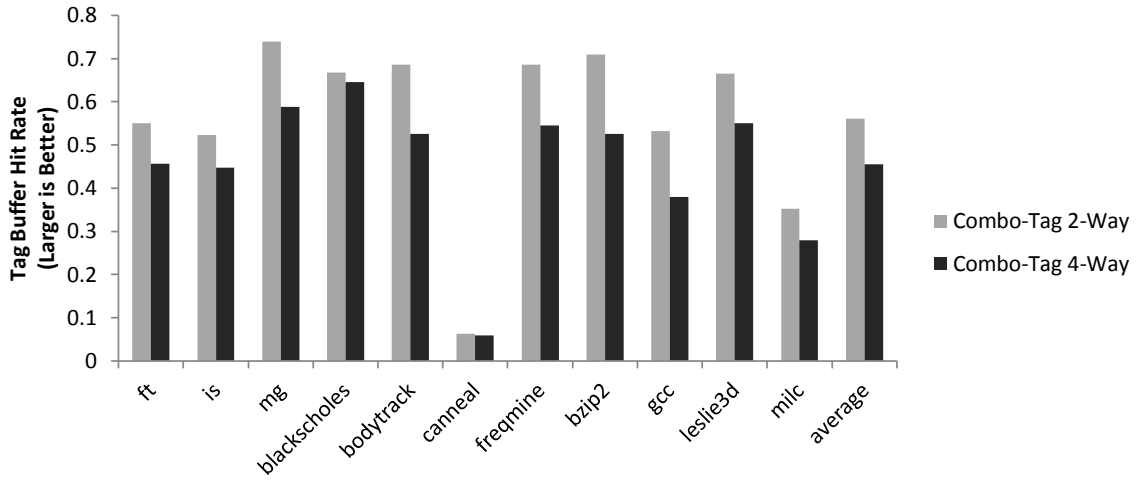

Figure 8.22: A comparison of the tag buffer hit rate of the two Combo-Tag cache architectures. These results are for a system with an open-page DDR3 DRAM cache and a DDR3 DRAM backing store. The selected benchmark suite workloads are used for this evaluation.

the tags. As a result, the higher tag hit rate seen in Figure 8.22 for the 2-way case provides a greater benefit than the reduction in miss rate granted the by 4-way case's higher associativity.

Another interesting feature of these results is that they demonstrate the value of concurrency over row buffer locality in terms of average access latency. The serial Alloy cache implementation places sequential sets in the same DRAM row in order to maximize the number of row buffer hits experienced by the system. Alternatively, the interleaved approach first spreads sets out across channels, ranks, and banks before placing them in the same row in order to leverage the available concurrency in the DRAM system. It is evident from the results in Figure 8.19 that the serial approach provides only moderate performance benefits even in an open page environment. The interleaved approach, on the other hand, out performs the serial approach in every case except for milc. This clearly demonstrates the importance of utilizing concurrency in the DRAM system.

8.8.3 More Complex DDR DRAM Backing Store

Due to the reduced relative miss penalty of DDR3 cached, DDR3 backed multi-level organizations, we expected to see a marked reduction in the improvement of combo-tag over the alloy cache implementation. However, the miss latencies observed in the DDR3-DDR3 experiment show that a lack of concurrency can lead to increased resource contention and longer access latencies. To investigate this, we next performed an experiment in which we greatly increased the concurrency of the

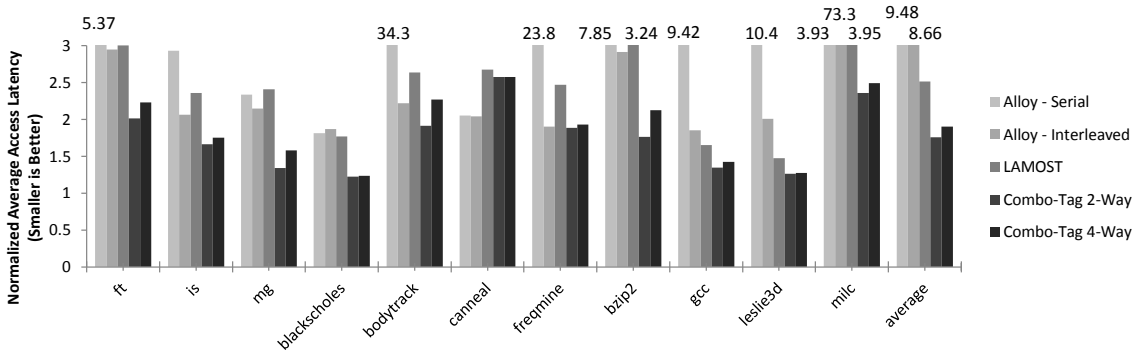

Figure 8.23: A comparison of the average access latencies of different DRAM cache architectures normalized to an ideal system with 4-way associativity and SRAM latency tag lookups. These results are for a system with a DDR3 DRAM cache and an expanded DDR3 DRAM backing store (4 Ranks). The selected benchmark suite workloads are used for this evaluation.

backing store from 1 rank in the previous experiment to 4 ranks. The results of this experiment can be found in Figures 8.23, 8.24, 8.25, and 8.26.

Like the results for the 1 rank RLDRAM-LPDDR system configuration, these results appear to be worse than the previous 1 rank DDR3-DDR3 system results despite having a much more complex and potentially faster backing store. However, as was the case with the 1 rank RLDRAM-LPDDR system, the 4 rank system results presented in Figure 8.23 indicate that the ideal system's performance changed much more dramatically than the other designs. In this case, the performance of all of the designs improved but the performance of the ideal system improved by a considerably larger amount. As a result, the relative distance between the other

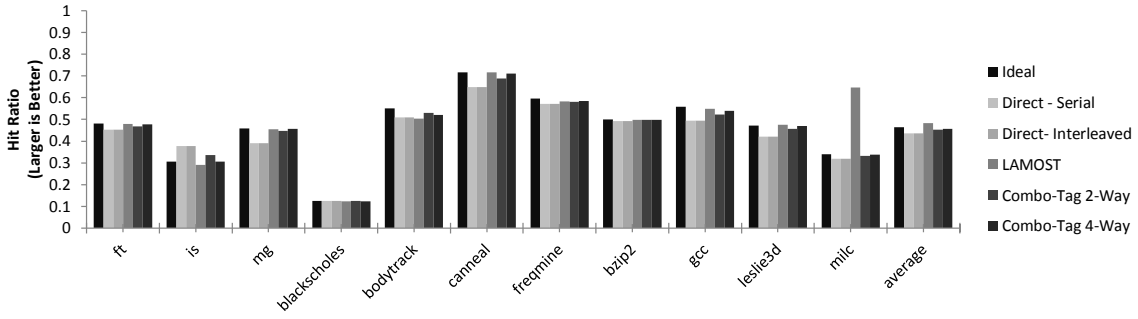

Figure 8.24: A comparison of the hit rates of different DRAM cache architectures. The ideal system has 4-way associativity and SRAM latency tag lookups. These results are for a system with a DDR3 DRAM cache and an expanded DDR3 DRAM backing store (4 Ranks). The selected benchmark suite workloads are used for this evaluation.

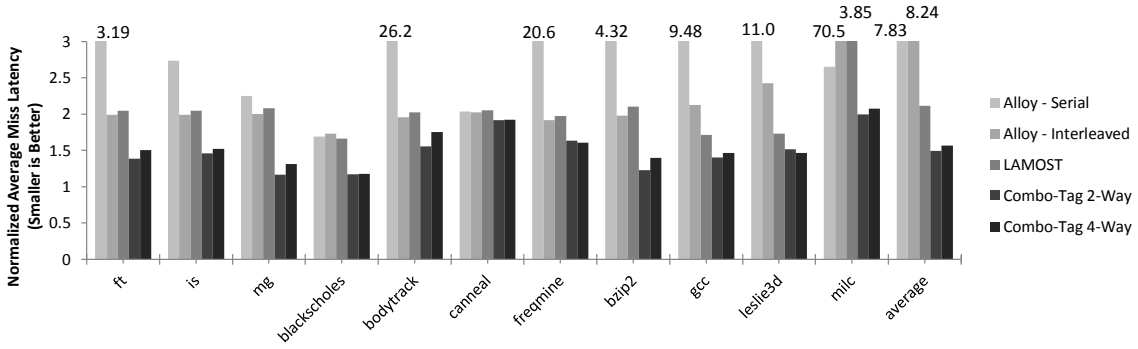

Figure 8.25: A comparison of the average miss latencies of different DRAM cache architectures normalized to an ideal system with 4-way associativity and SRAM latency tag lookups. These results are for a system with a DDR3 DRAM cache and an expanded DDR3 DRAM backing store (4 Ranks). The selected benchmark suite workloads are used for this evaluation.

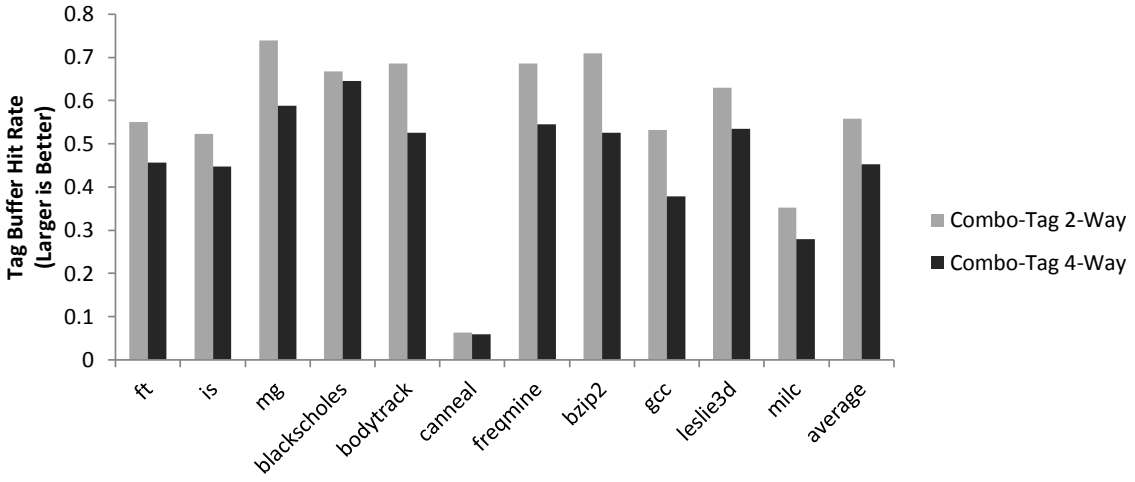

Figure 8.26: A comparison of the tag buffer hit rate of the two Combo-Tag cache architectures. These results are for a system with a DDR3 DRAM cache and an expanded DDR3 DRAM backing store (4 Ranks). The selected benchmark suite workloads are used for this evaluation.

designs and the ideal design has increased rather than decreased.

The relative differences between the designs have also changed as well though. In particular, both Alloy cache implementations appear to benefit significantly less from the improved backing store than the Combo-Tag implementations. In many cases we can see the distance between the performance of the Interleaved Alloy design and the 4-way Combo-Tag design in Figure 8.23 has increased relative to the results in Figure 8.19. The LAMOST design also appears to benefit from the increased backing store concurrency. This suggests that the ability of the associative designs to reduce the miss rate helps to keep the pressure on the backing store to more manageable levels and results in the improved miss latencies seen in Figure 8.25.

8.8.4 DDR DRAM Closed Page Cache and Backing Store

One of the more interesting differences between the RLDRAM cache and the DDR3 DRAM cache is the possibility to use an open page policy to speed up sequential accesses to the same row. Many of the potential DRAM cache designs attempt to leverage the potential for row buffer hits by carefully laying out their sets in rows such that sequential sets will result in row buffer hits. In the earlier DDR DRAM cache experiments we also used an open row policy in our cache to take advantage of row buffer hits. However, in order to understand the full impact of the open page policy we need to see what happens when the system in switched to a closed page policy instead. To this end we performed an experiment where we used the same DDR3 cached, DDR3 backed organization as our earlier experiment but switched the cache to a closed page policy. We show the results for this experiment in Figures 8.27, 8.28, 8.29, and 8.30.

The lack of row buffer hits clearly results in a massive slowdown for the serial case in Figure 8.27. This makes sense because the serial implementation relied solely on row buffer hits to provide performance and the closed page system has eliminated those. This same effect also appears to have affected the 2-way case as well because its relative performance in this system is worse than it was in the other DDR3-DDR3 organizations. This suggests that at least some of the performance benefits of the 2-way approach compared to the 4-way approach were coming from an increased number of row buffer hits in the 2-way case. The interleaved Alloy system, however, appears to be relatively unaffected by the switch to a closed page system because

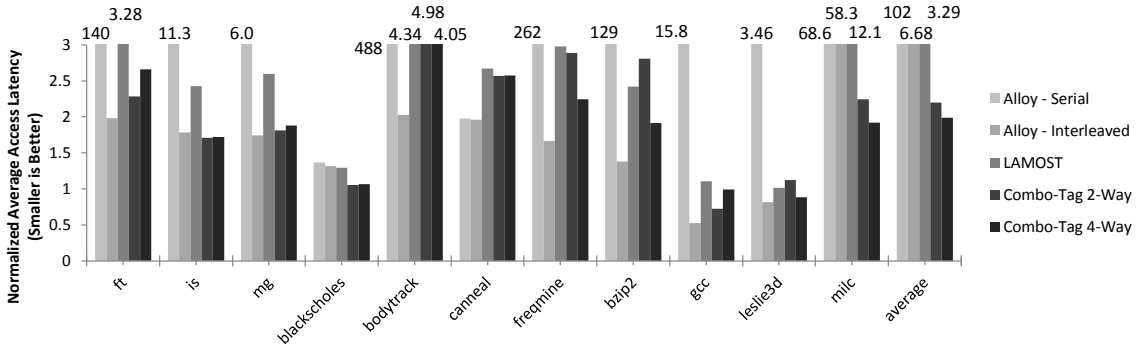

Figure 8.27: A comparison of the average access latencies of different DRAM cache architectures normalized to an ideal system with 4-way associativity and SRAM latency tag lookups. These results are for a system with a closed-page DDR3 DRAM cache and a DDR3 DRAM backing store. The selected benchmark suite workloads are used for this evaluation.

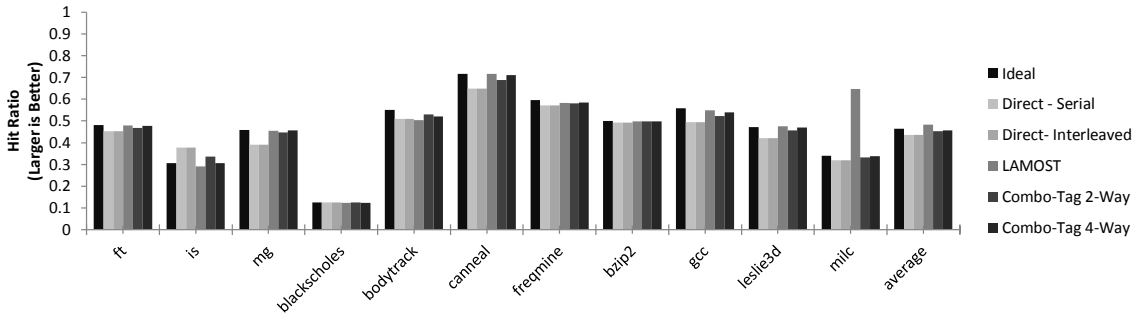

Figure 8.28: A comparison of the hit rates of different DRAM cache architectures. The ideal system has 4-way associativity and SRAM latency tag lookups. These results are for a system with a closed-page DDR3 DRAM cache and a DDR3 DRAM backing store. The selected benchmark suite workloads are used for this evaluation.

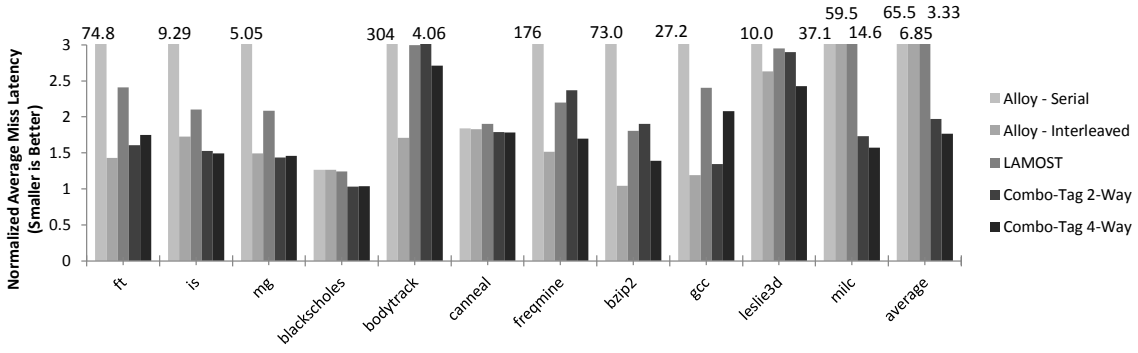

Figure 8.29: A comparison of the average miss latencies of different DRAM cache architectures normalized to an ideal system with 4-way associativity and SRAM latency tag lookups. These results are for a system with a closed-page DDR3 DRAM cache and a DDR3 DRAM backing store. The selected benchmark suite workloads are used for this evaluation.

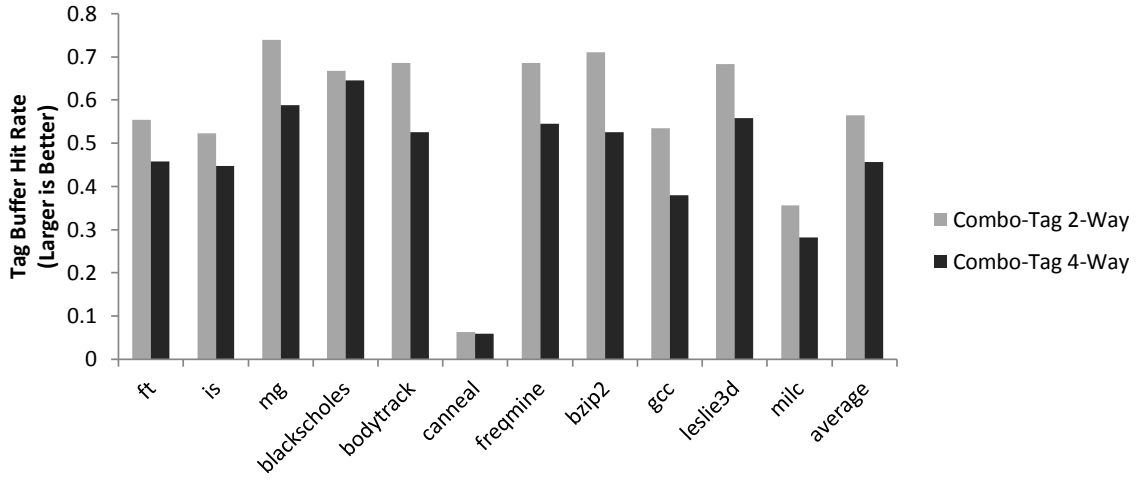

Figure 8.30: A comparison of the tag buffer hit rate of the two Combo-Tag cache architectures. These results are for a system with a closed-page DDR3 DRAM cache and a DDR3 DRAM backing store. The selected benchmark suite workloads are used for this evaluation.

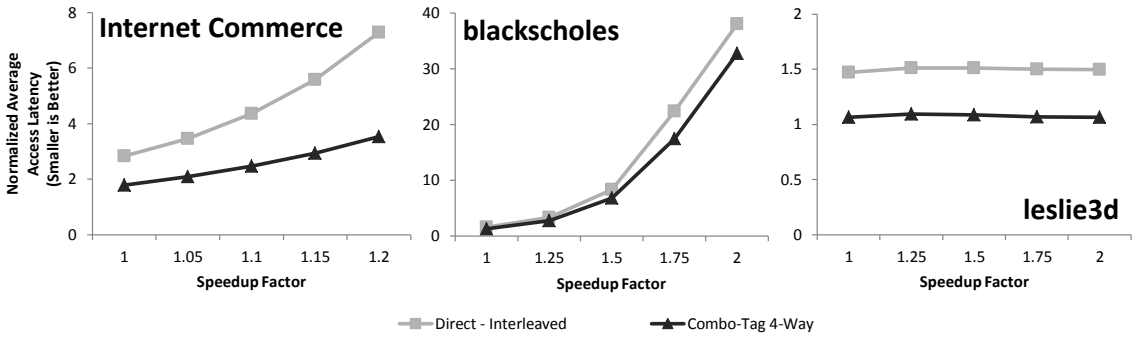

Figure 8.31: The effect of increasing the pressure on the cache by marginally speeding up the trace normalized to an ideal system system with 4-way associativity, SRAM latency tag lookups, and a speedup factor of 1.

most of its performance was derived from concurrency utilization rather than row buffer hits.

8.9 Bandwdith Sensitivity

Finally, another aspect of these systems that we are interested in is how they react to increased cache pressure. Bandwidth pressure on DRAM caches is likely to continue to increase in the future as more cores and threads are added to systems. Therefore, it is important to understand how the different stacked cache approaches will react as the cache pressure is increased. Figure 8.31 shows the effects of gradually increasing the cache pressure by incrementally speeding up the traces. This simulates the effect of having additional threads generating requests. Here we can see again that the effects are highly workload dependent, with some workloads, like leslie3d, showing almost no response and others, like blackscholes, that do. More

importantly, though, we can see from the internet commerce result that for some workloads, the ability of the different DRAM cache architectures to handle additional pressure varies greatly. As the pressure is increased, Alloy cache is quickly overwhelmed and its performance degrades. Combo-tag, on the other hand, handles the pressure better because the tag buffer acts as a sort of filter which reduces the amount of transactions that are going to the cache. Every access, regardless of whether it is a hit or a miss, is handled by the cache in the direct mapped system but some misses are handled only by the tag buffer in the Combo-Tag system. This reduces some of the load on the cache in the Combo-Tag case. Therefore, the Combo-Tag architecture is better able to handle cache pressure and better suited to handle the additional traffic of future workloads.

8.10 Summary

The architecture presented in this chapter enables DRAM cache associativity in a way that is efficient both in terms of tag storage and bandwidth utilization. Our novel tag buffer design and replacement policies enable the temporary buffering of tags in just 4KB of SRAM. Additionally, our method of mapping sets within the DRAM row and coalescing tags into a single DRAM access reduces the bandwidth load on the cache due to tag lookups while also amortizing the latency cost of a tag lookup. As a result of these features, Combo-Tag is able to achieve a speedup over direct mapped caches while more efficiently utilizing cache bandwidth and tag storage space than the alternative associative DRAM cache architectures. We also

demonstrate the importance of taking into account the concurrency in the DRAM address space when designing DRAM based caches.

Chapter 9: Backing Store Design Analysis

Having optimized the cache to minimize misses as much as possible, we now turn our attention to reducing the miss penalty by optimizing the backing store. In this chapter we will investigate the effects of different organizations, page sizes, prefetching degrees, and host channel organizations on the performance of the backing store. We perform this analysis for a range of cache sizes and potential backing store technologies in order to determine how the design of the backing store needs to change to accommodate the operating characteristics of different kinds of memory. In this chapter we focus on backing store technologies that have characteristics which lie between the extremes of NAND Flash and DRAM. This is because NAND Flash and DRAM have typical organizations and page sizes that are the result of years of development whereas the correct parameters for other types of memory are less well understood. The results of these studies reveal the complex relationships that exist in the backing store between access size, access latency, and concurrency.

9.1 Evaluation Methodology

The parameters of the test system used in this chapter were selected to maximize the stress on the backing store while remaining realistic. To this end we utilize

a direct mapped organization for our DRAM cache to provide the highest possible miss rate for a reasonably sized DRAM cache. Alternatively we could have used even smaller caches to create additional miss pressure on the backing store. However, the resulting system and miss traffic would have been less indicative of the actual systems that would employ a multi-level main memory architecture. We also provide the DRAM cache with enough concurrency to ensure that the cache will not be a system performance bottleneck in these simulations. Table 9.1 provides the parameters of the test system used in this chapter.

We present results for three different backing store technologies in these experiments. They are classified according to their read latency and represent memories that are twice as slow as DRAM, twenty times as slow as DRAM, and twice as fast as SLC Flash. These three hypothetical memory technologies therefore represent the full range of possible memory latencies between the two established extremes in terms of memory technology. Furthermore, because we assume these hypothetical technologies to be similar to the proposed non-volatile main memory technologies, we simulate them with an asymmetric write latency that is 10x the read latency. This is similar to the latencies that we used in Chapter 6.

The studies presented in this chapter were performed using detailed trace based simulation using HybridSim with OMS and DRAMSim2 providing the memory simulations. Each of the experiments was warmed for 1 million memory transactions in order to eliminate most of the cold cache effects. After the cache was warmed, the simulations were run for an additional 10 million memory transactions or until the trace completed. The number of accesses is lower in these experiments than

they were in earlier chapters of this dissertation because the page sizes used in these studies can be much larger. For instance, the page size in chapter 8 was just 64B while the page sizes in this chapter can reach up to 16KB. As a result, the amount of data that is transfered in some of these experiments is actually 25 times greater than it was in Chapter 8 despite the smaller number of overall transactions.

9.1.1 Benchmarks

The studies in this chapter use the same benchmarks that where used in Chapter 8. These benchmarks were selected from the SPEC CPU2006 [111], NPB [112], and PARSEC [113] benchmark suites because they have large working sets and a particularly high MPKI. We reuse these workloads because they place pressure on the cache and as a result tend to induce a higher miss rate. The increased miss pressure is desirable for these tests because it helps to highlight the performance of the backing store. The relevant properties of these benchmarks can be found in Table 9.2.

9.2 Ranks versus Page Size

DRAM and NAND Flash represent two different extremes of memory technology in terms of their organization, page size, and access latency. DRAM, for example, utilizes a relatively small 64B access size and groups together 8 devices into a rank that then services each access as one unit. NAND flash, on the other hand, utilizes much larger access sizes to help amortize the cost of its long access

Table 9.1: Baseline Simulator Configuration

Processor	
Number of cores	8-core
Issue Width	4
Frequency	3.2GHz
On Chip Caches	
L1I (private)	128 KB, 8-way, 64 B block size
L1D (private)	128 KB, 8-way, 64 B block size
L2 (private)	256 KB, 8-way, 64 B block size
L3 (shared)	32 MB, 20-way, 64 B block size
In-Package DRAM Cache	
Organization	Direct Mapped, 64B - 16KB page size
Bus Frequency	DDR-1333
DRAM Bus Width	64 bits per Channel
DRAM Channels	4
DRAM Ranks	4 Ranks per Channel
DRAM Banks	8 Banks per Rank
Row Size	2048 Bytes
tCAS-tRCD-tRP-tRAS	10-10-10-24
Backing Store	
Organization	64B - 16 KB page size
Backing Store Channels	1-64
Backing Store Ranks	1 Ranks per channel
Backing Store Dies	1-64 Dies per Rank

Table 9.2: Benchmark Characteristics

	Observed Footprint	**L3 MPKI**
ft	1287.27 MB	7.115501
is	264.87 MB	10.424386
mg	430.87 MB	13.5972049
blackscholes	269.53 MB	0.6400292
bodytrack	534.28 MB	0.4995586
canneal	497.60 MB	6.50089792
freqmine	894.04 MB	1.31842398
bzip2	2559.93 MB	61.0071412
gcc	441.31 MB	6.0996348
leslie3d	601.15 MB	18.724911
milc	1909.66 MB	22.4704558

latencies. In addition, NAND flash systems are generally organized so that each device operates independently of the others in the system. As a result, NAND flash systems are capable of having many more operations in flight at the same than a typical DRAM system. However, this additional concurrency comes at the cost of reduced bandwidth per access as the Flash access has to be transported via only 8 pins while the DRAM rank is capable of utilizing 64 pins (8 pins per chip for a typical rank size of 8 devices). Based on the standard organizations of Flash and DRAM, this trade-off between bandwidth and concurrency seems to favor bandwidth for faster devices and concurrency for slower ones.

It is not clear, though, whether technologies that are 2, 20, or 200 times slower than DRAM will benefit more from bandwidth or concurrency in the system. To

test this we perform a series of experiments where we group together the memory devices to form ranks of various sizes. This is similar to the superblock configuration that has been utilized to improve NAND flash bandwidth [98,114]. However, in our evaluations we hold the number of dies constant in the entire system rather than the total capacity so that increasing the size of a rank decreases the amount of concurrency available in the system. In addition, we also vary the total page size for each rank so that in some configurations the contribution of each die is actually quite small. This allows us to determine the relationship between bandwidth, page size, and concurrency for different classes of memory technology from relatively fast ones to relatively slow ones. We also vary the size of the cache to expose the role that miss pressure plays in the effectiveness of each organization.

We present the results of this investigation first for each workload individually and then present an average of all of the workloads.

9.2.1 128MB Cache

Figure 9.12 shows the results for the sweep of rank sizes versus page sizes for 64 dies total. The cache in these experiments is 128MB and is smaller than all of the workloads that we're testing. We can see quickly that the different access latencies produce very different results for the different organizations. For instance, smaller pages seem to work better for the fastest memory while the longer latency memory benefits from larger pages. Interestingly, though, all of the memories do not benefit from the largest page size regardless of organization. This would seem

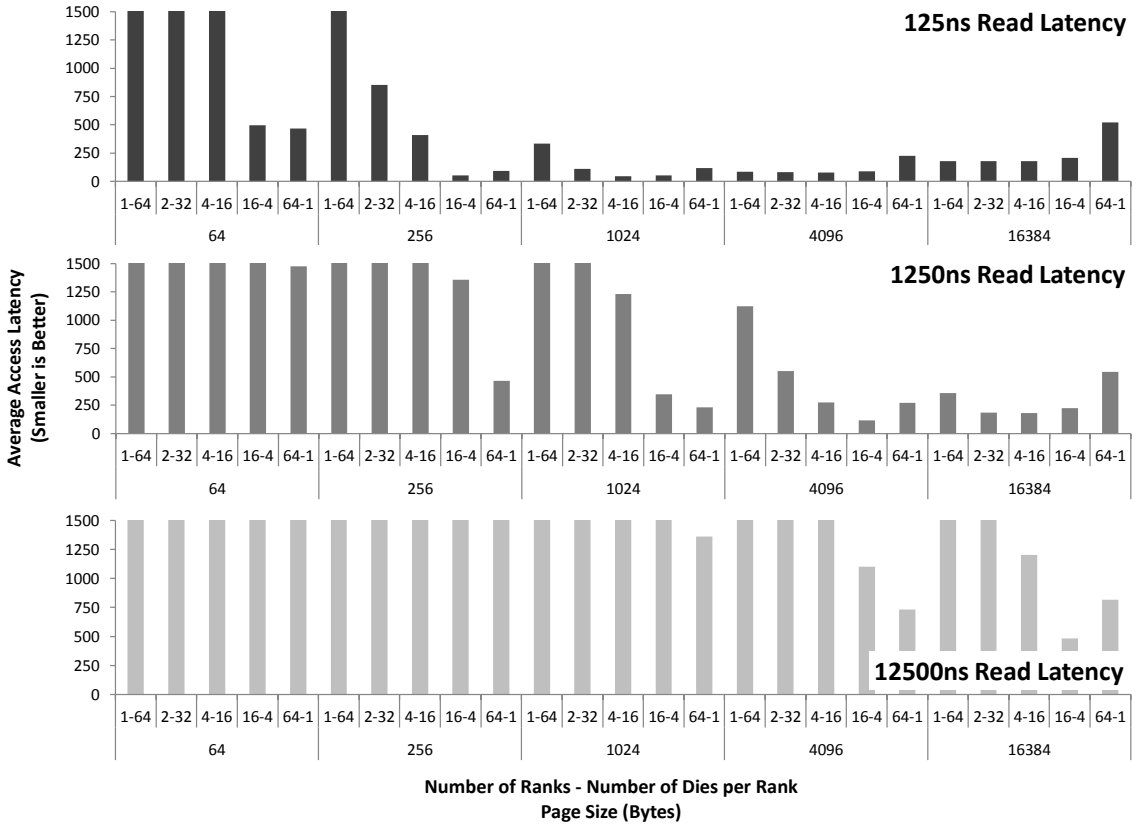

Figure 9.1: The average access latencies for *ft* that result from using different sized ranks and different sized pages with a 128MB cache that is significantly smaller than all of the working sets of the benchmarks used in this chapter. Several different read latencies are shown to demonstrate the impact that technology choice has on the potential organizations.

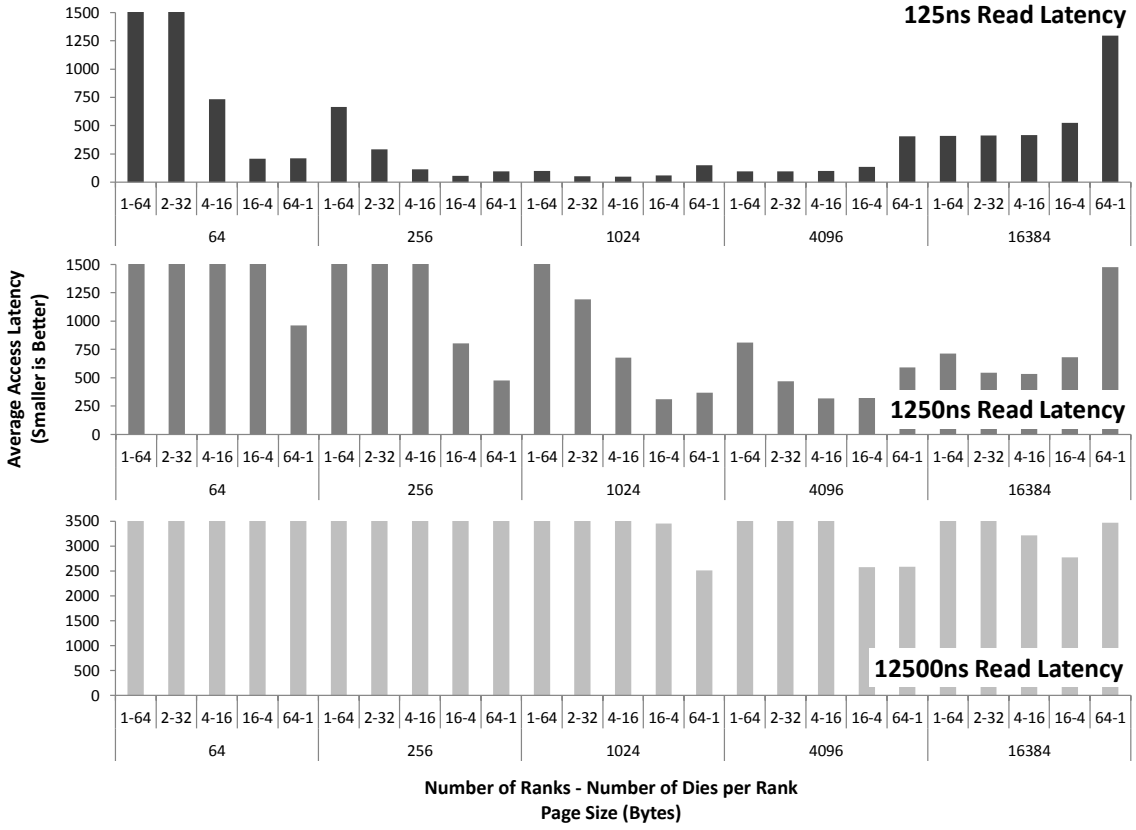

Figure 9.2: The average access latencies for *is* that result from using different sized ranks and different sized pages with a 128MB cache that is significantly smaller than all of the working sets of the benchmarks used in this chapter. Several different read latencies are shown to demonstrate the impact that technology choice has on the potential organizations.

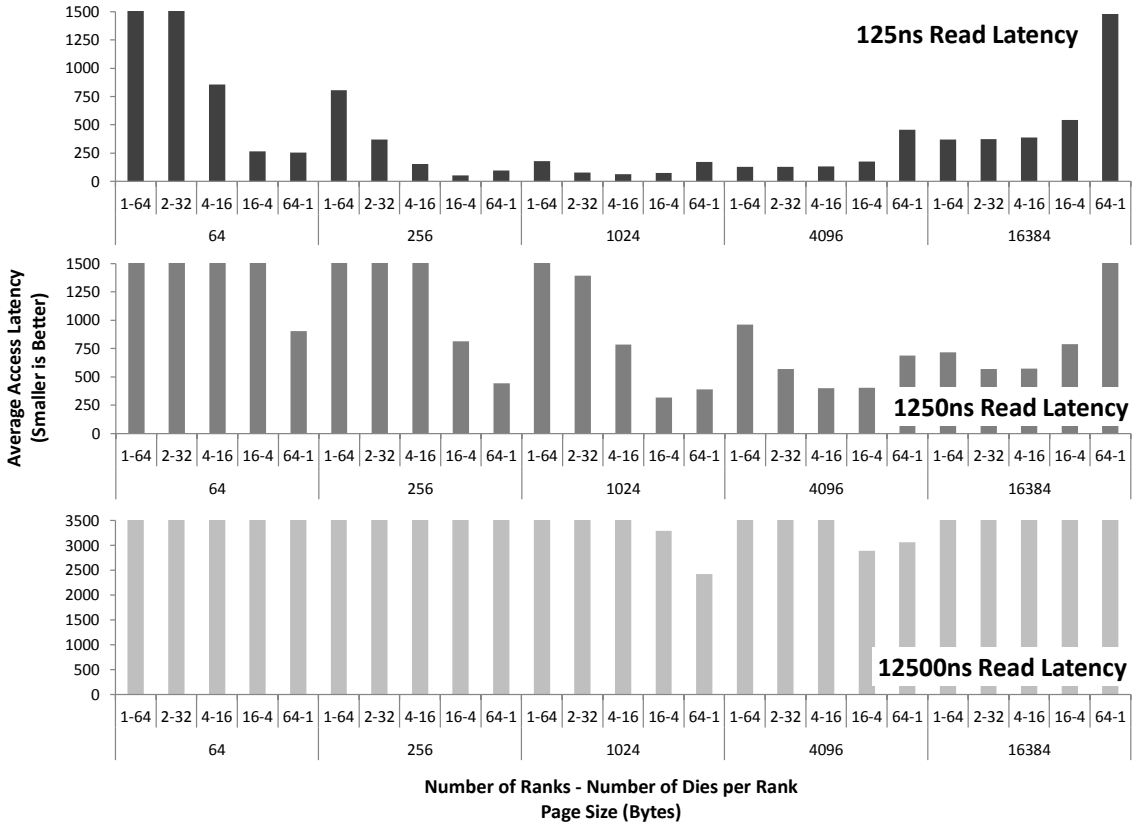

Figure 9.3: The average access latencies for *mg* that result from using different sized ranks and different sized pages with a 128MB cache that is significantly smaller than all of the working sets of the benchmarks used in this chapter. Several different read latencies are shown to demonstrate the impact that technology choice has on the potential organizations.

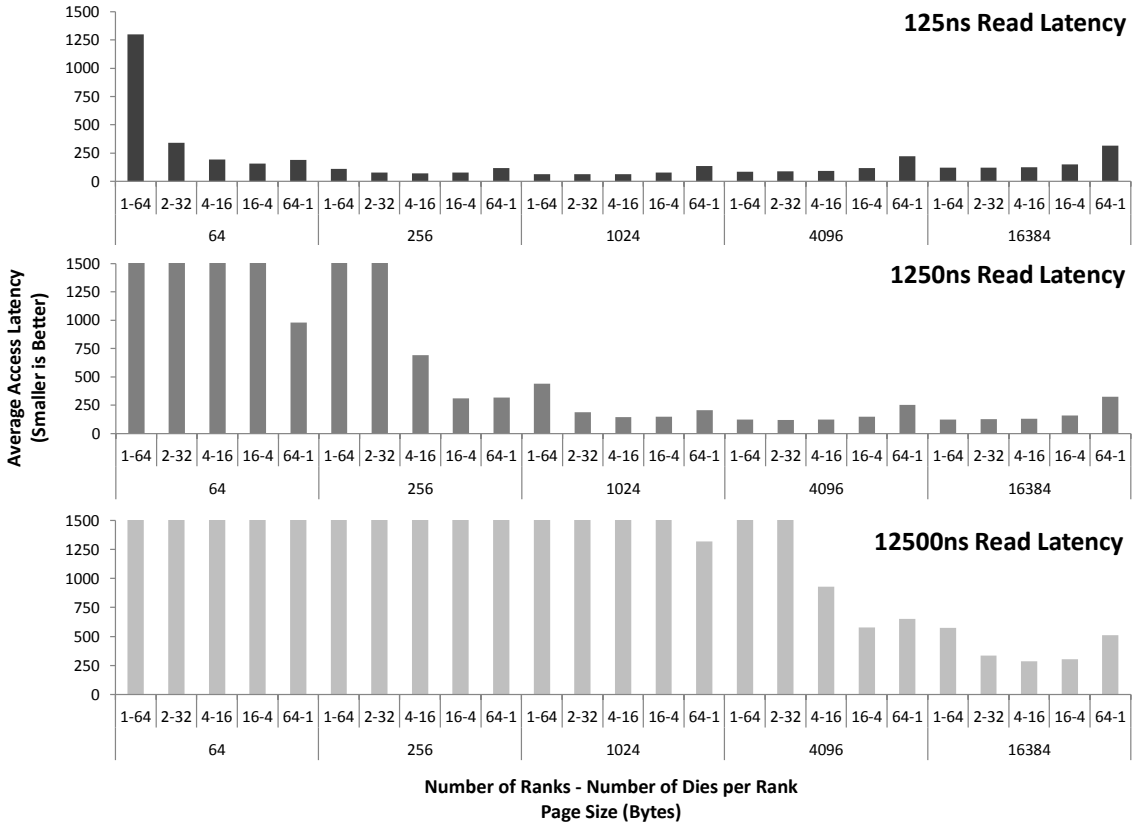

Figure 9.4: The average access latencies for *blackscholes* that result from using different sized ranks and different sized pages with a 128MB cache that is significantly smaller than all of the working sets of the benchmarks used in this chapter. Several different read latencies are shown to demonstrate the impact that technology choice has on the potential organizations.

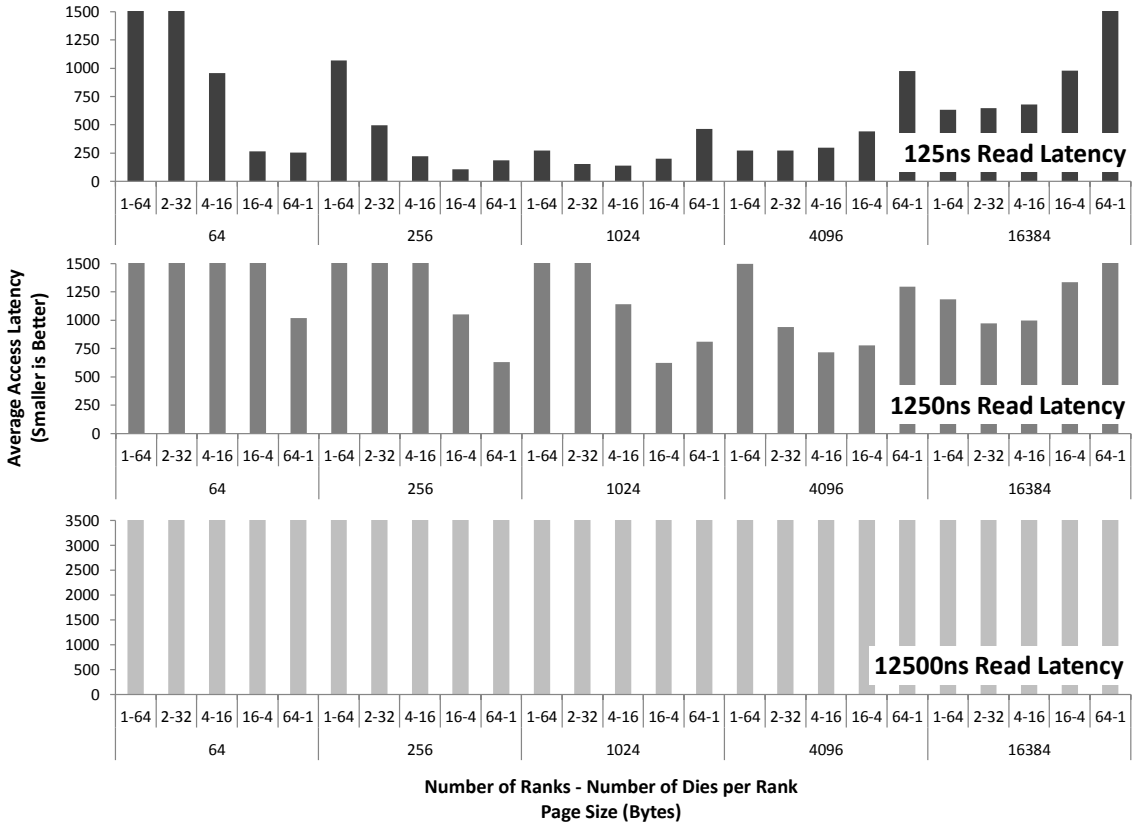

Figure 9.5: The average access latencies for *bodytrack* that result from using different sized ranks and different sized pages with a 128MB cache that is significantly smaller than all of the working sets of the benchmarks used in this chapter. Several different read latencies are shown to demonstrate the impact that technology choice has on the potential organizations.

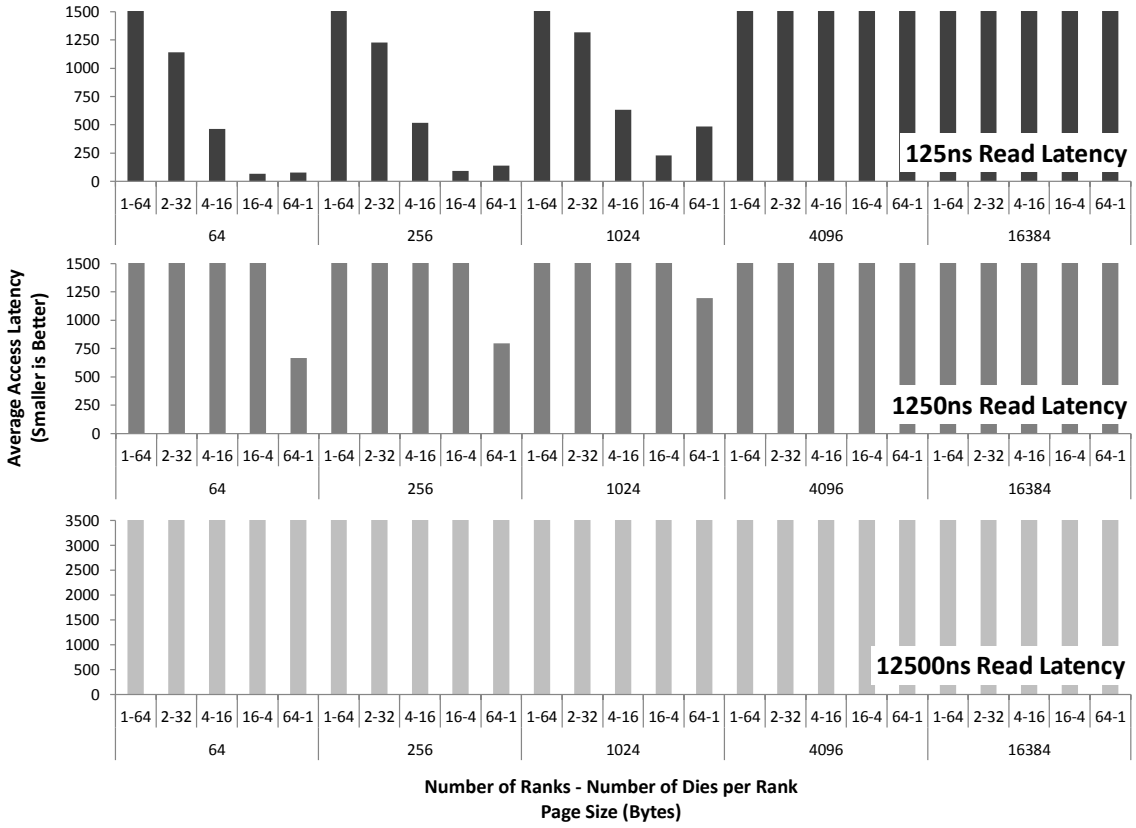

Figure 9.6: The average access latencies for *canneal* that result from using different sized ranks and different sized pages with a 128MB cache that is significantly smaller than all of the working sets of the benchmarks used in this chapter. Several different read latencies are shown to demonstrate the impact that technology choice has on the potential organizations.

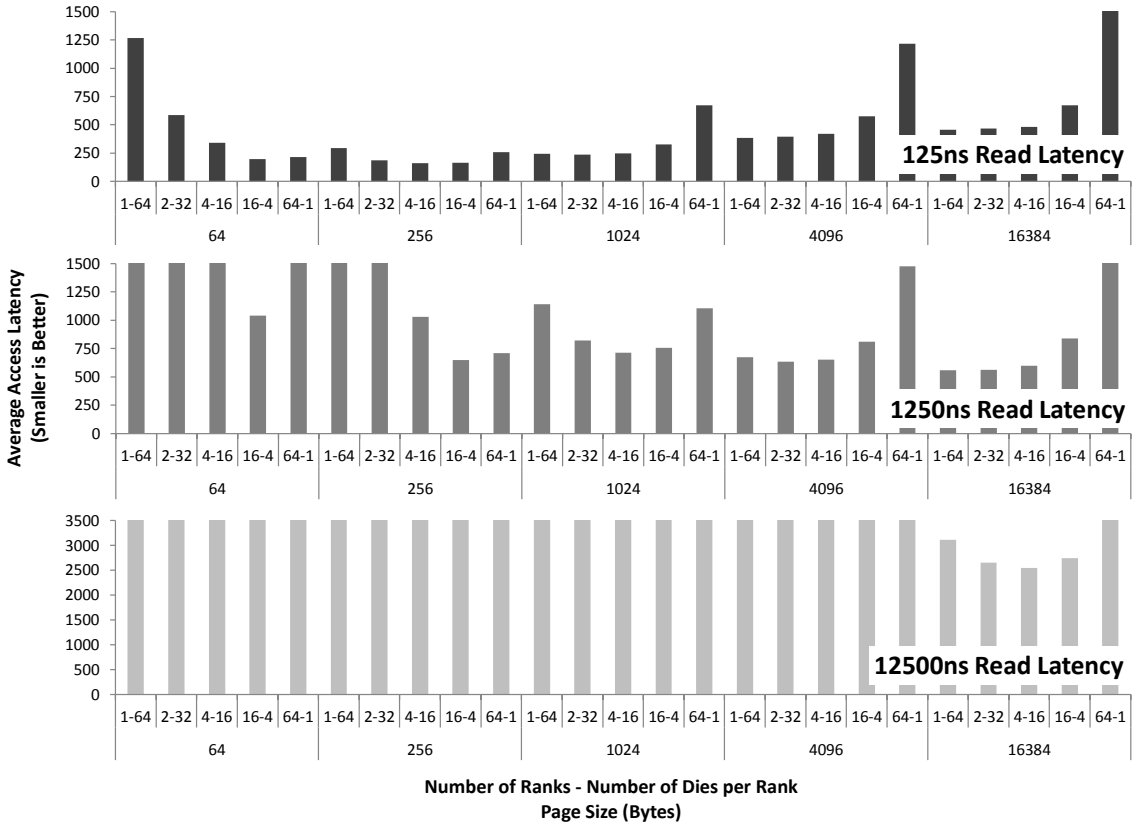

Figure 9.7: The average access latencies for *freqmine* that result from using different sized ranks and different sized pages with a 128MB cache that is significantly smaller than all of the working sets of the benchmarks used in this chapter. Several different read latencies are shown to demonstrate the impact that technology choice has on the potential organizations.

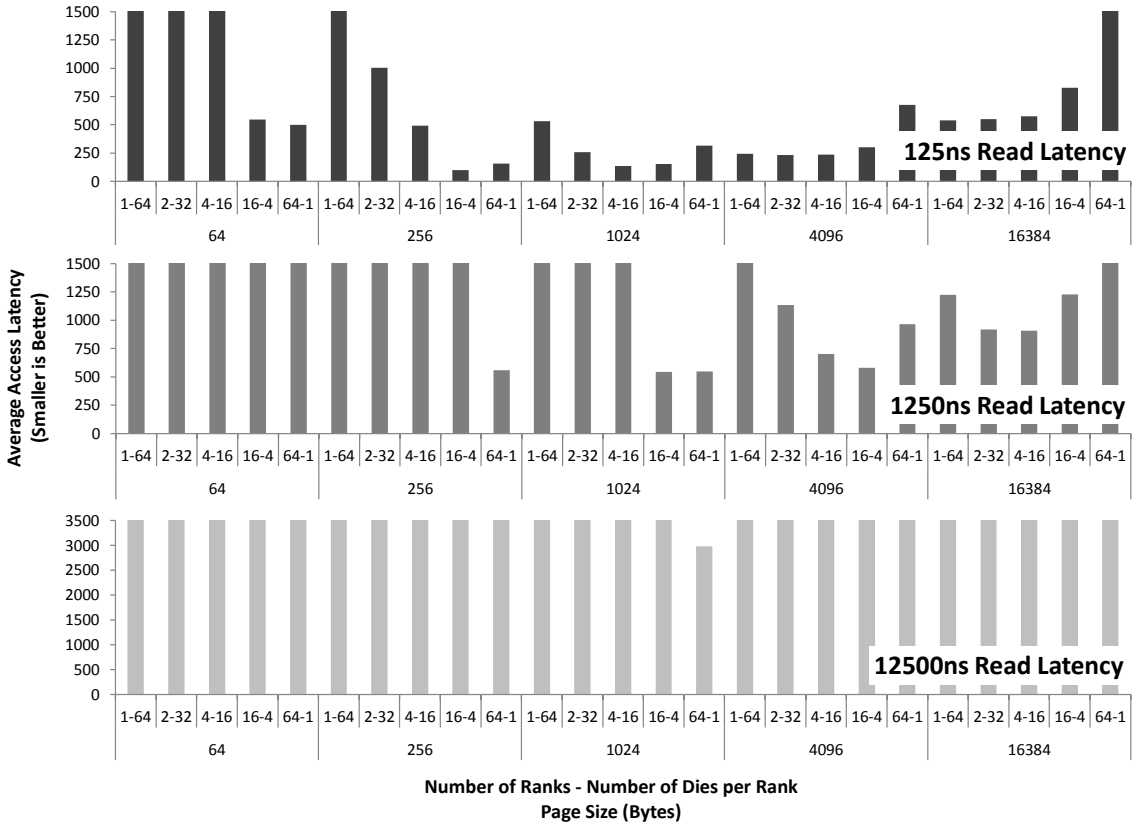

Figure 9.8: The average access latencies for *bzip2* that result from using different sized ranks and different sized pages with a 128MB cache that is significantly smaller than all of the working sets of the benchmarks used in this chapter. Several different read latencies are shown to demonstrate the impact that technology choice has on the potential organizations.

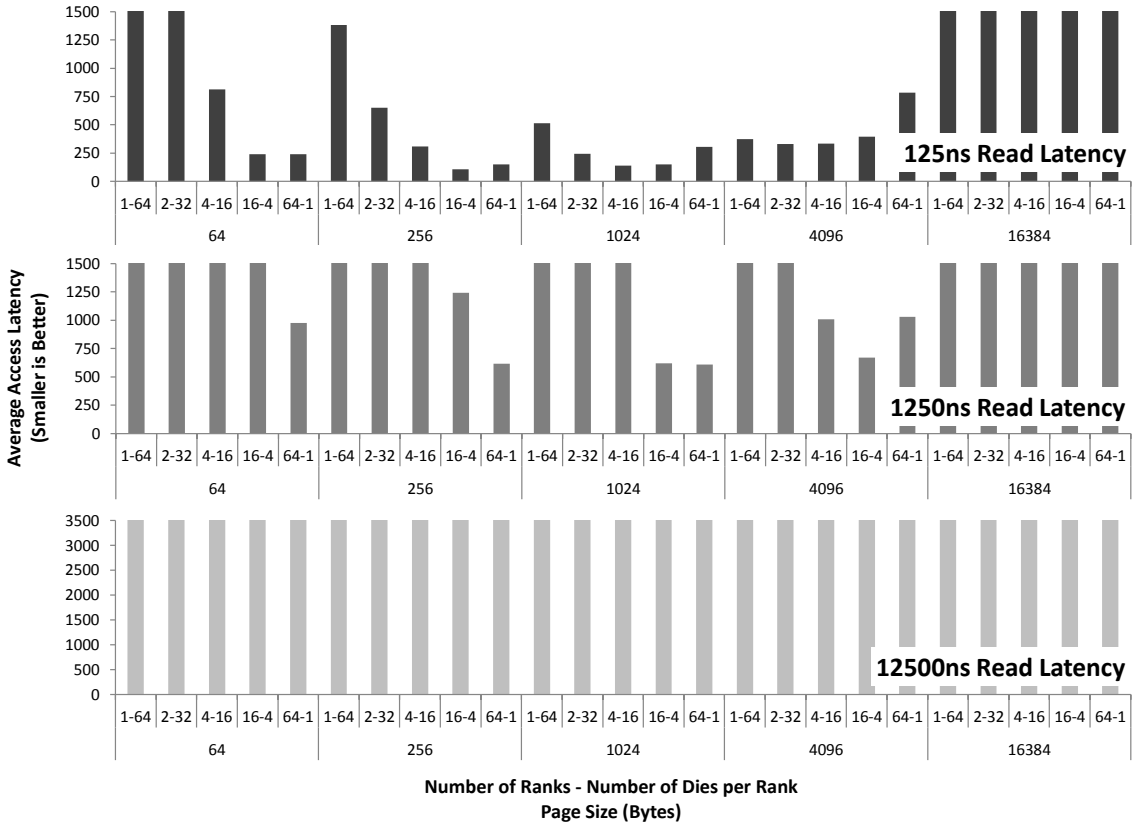

Figure 9.9: The average access latencies for *gcc* that result from using different sized ranks and different sized pages with a 128MB cache that is significantly smaller than all of the working sets of the benchmarks used in this chapter. Several different read latencies are shown to demonstrate the impact that technology choice has on the potential organizations.

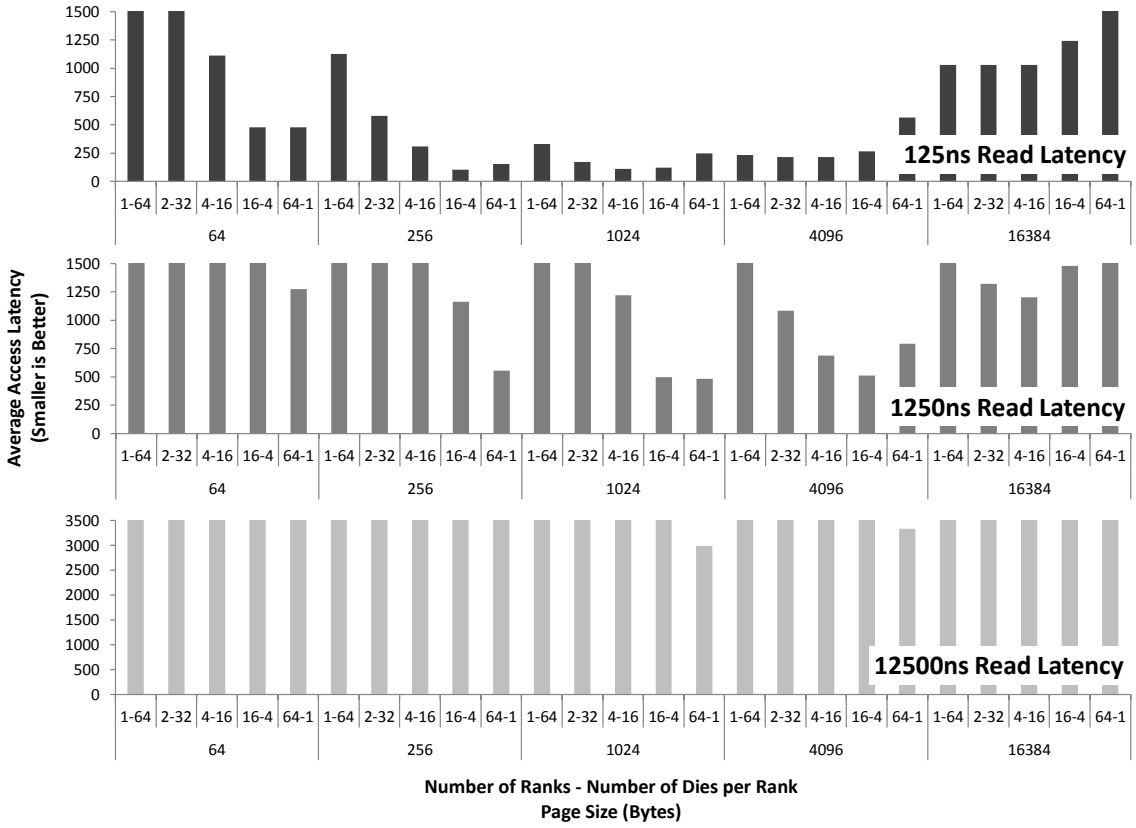

Figure 9.10: The average access latencies for *leslie3d* that result from using different sized ranks and different sized pages with a 128MB cache that is significantly smaller than all of the working sets of the benchmarks used in this chapter. Several different read latencies are shown to demonstrate the impact that technology choice has on the potential organizations.

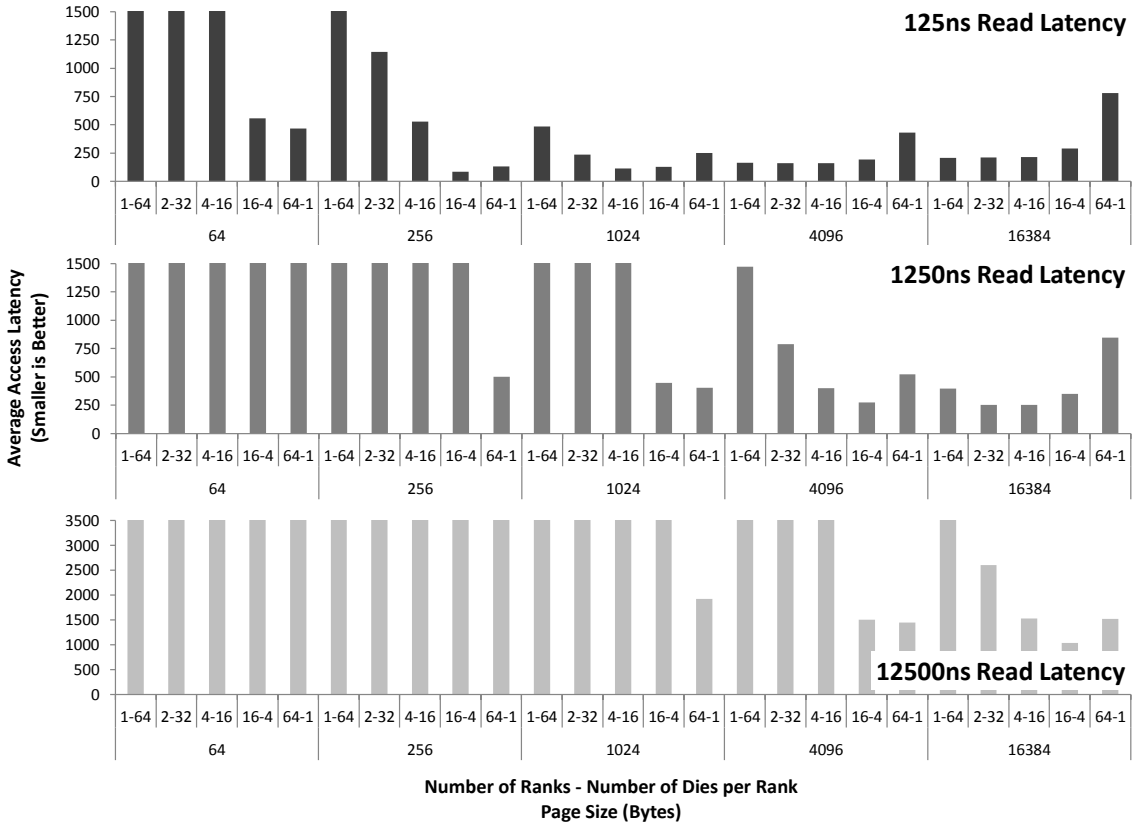

Figure 9.11: The average access latencies for *milc* that result from using different sized ranks and different sized pages with a 128MB cache that is significantly smaller than all of the working sets of the benchmarks used in this chapter. Several different read latencies are shown to demonstrate the impact that technology choice has on the potential organizations.

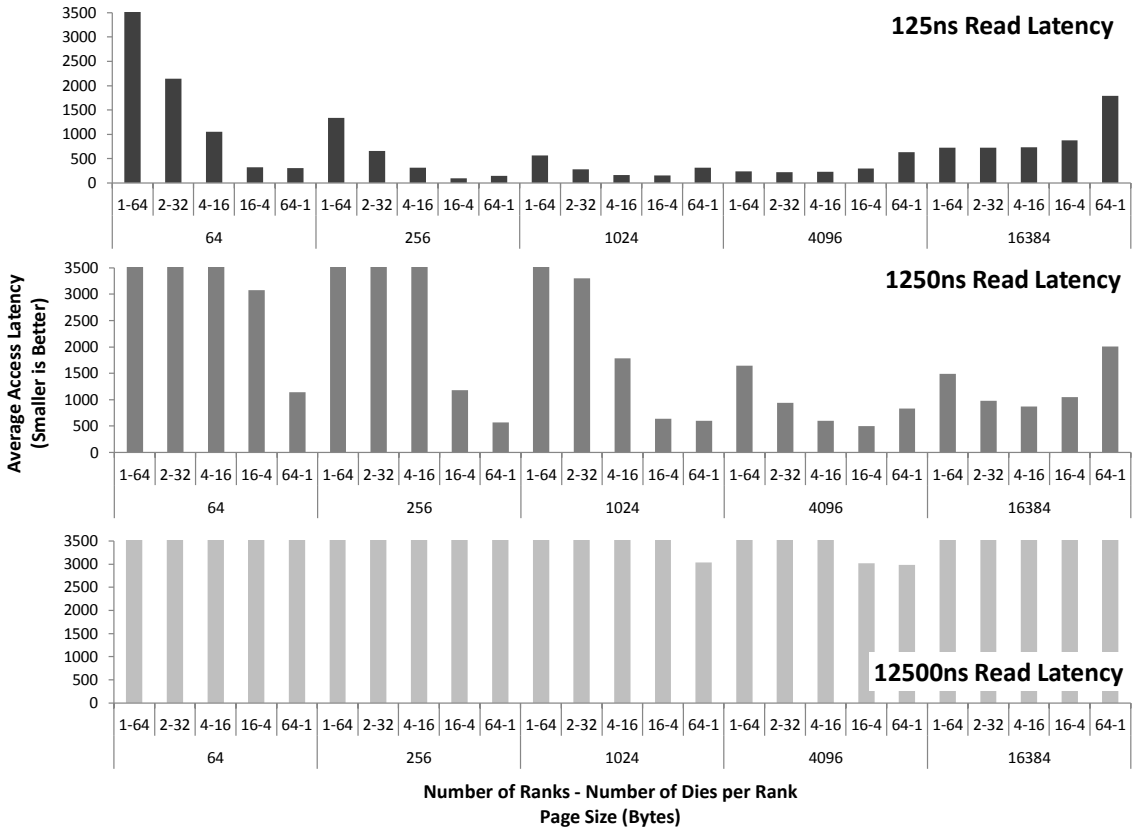

Figure 9.12: The average access latencies averaged across all workloads that result from using different sized ranks and different sized pages with a 128MB cache that is significantly smaller than all of the working sets of the benchmarks used in this chapter. Several different read latencies are shown to demonstrate the impact that technology choice has on the potential organizations.

to indicate an upper bound on the usefulness of the additional information received in larger pages. In addition, all of the memories appeared to benefit from at least some degree of concurrency with 16 ranks providing the greatest benefit for all three technologies. The 1250ns memory also shows some interesting trends regarding the impact of concurrency on systems using the 2 largest page sizes. In both cases, the

addition of concurrency helps to improve the performance of the system but only up to a point. This seems to suggest that using larger pages can result in the system becoming more sensitive to bandwidth.

9.2.2 256MB Cache

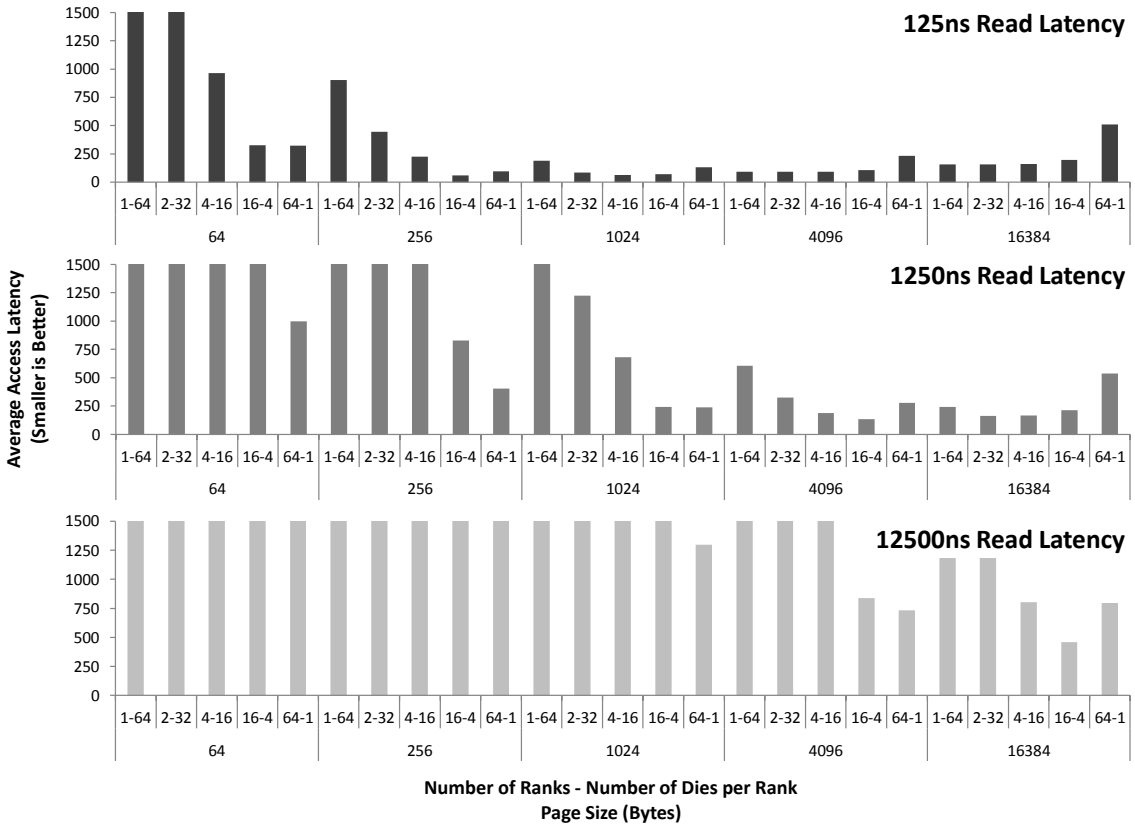

Figure 9.13: The average access latencies for *ft* that result from using different sized ranks and different sized pages with a 128MB cache that is significantly smaller than all of the working sets of the benchmarks used in this chapter. Several different read latencies are shown to demonstrate the impact that technology choice has on the potential organizations.

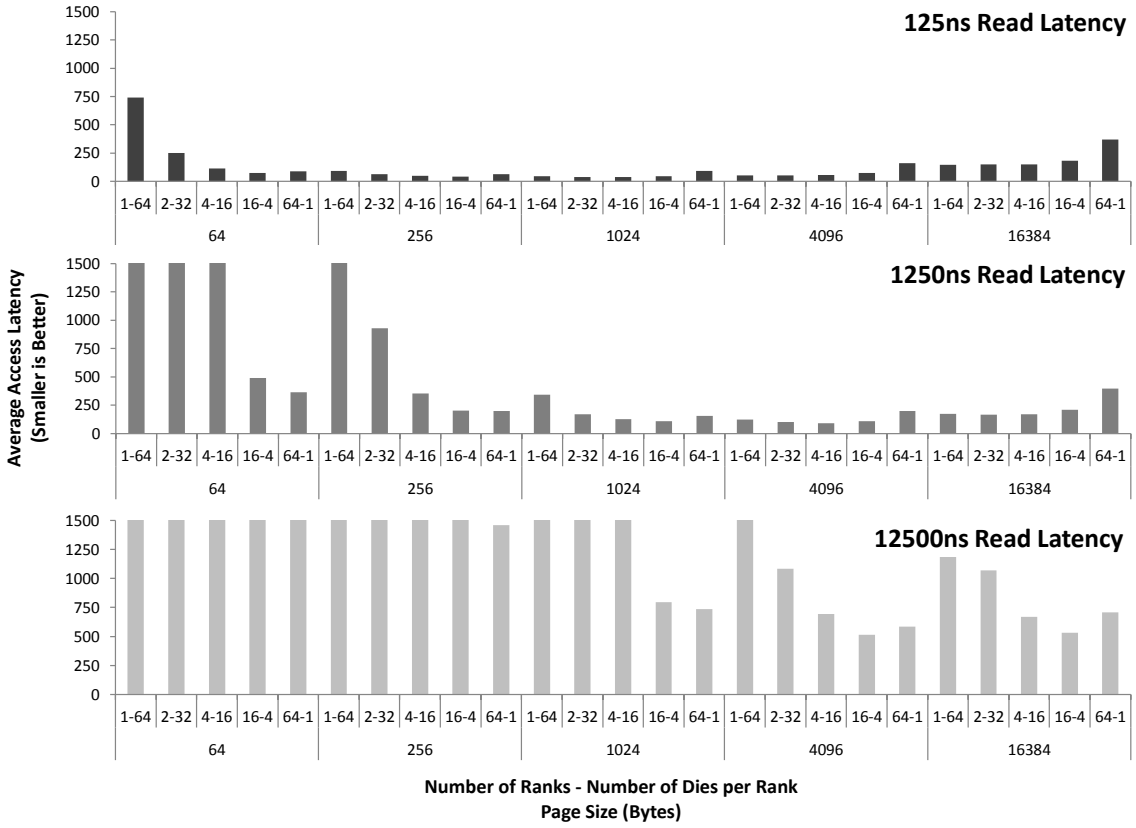

Figure 9.14: The average access latencies for *is* that result from using different sized ranks and different sized pages with a 256MB cache that is significantly smaller than all of the working sets of the benchmarks used in this chapter. Several different read latencies are shown to demonstrate the impact that technology choice has on the potential organizations.

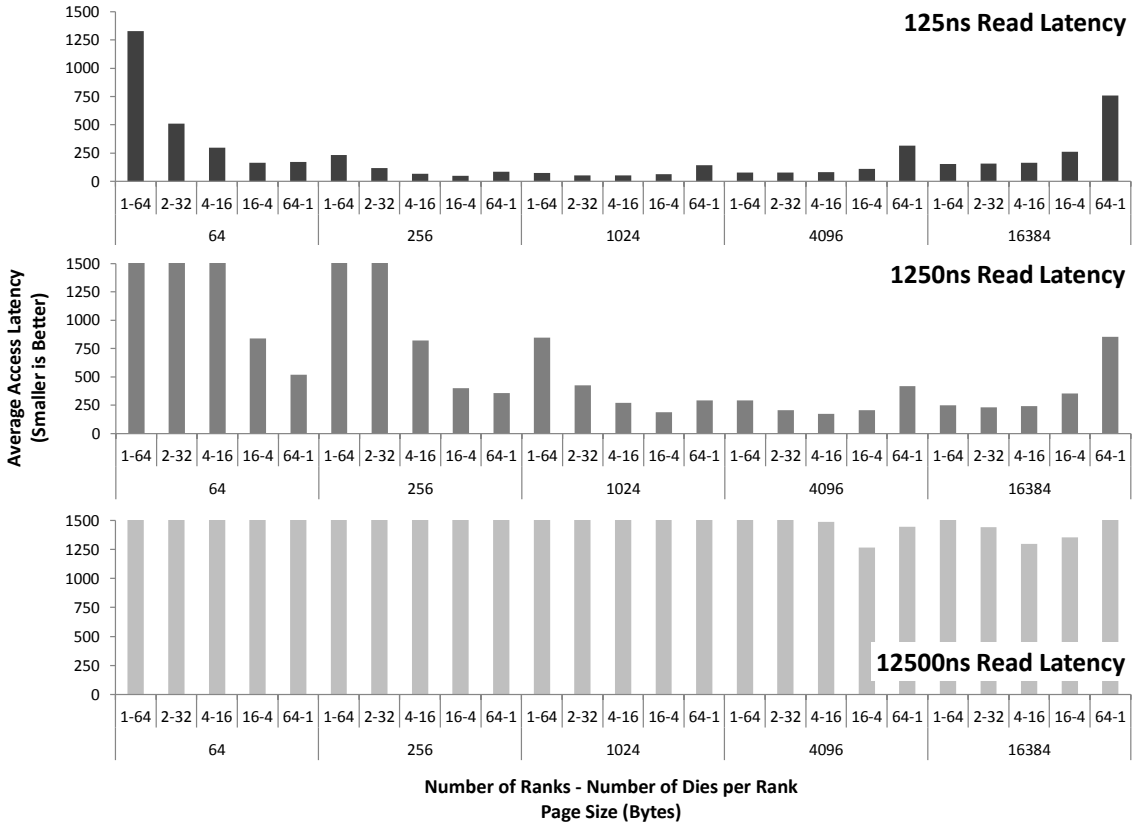

Figure 9.15: The average access latencies for *mg* that result from using different sized ranks and different sized pages with a 256MB cache that is significantly smaller than all of the working sets of the benchmarks used in this chapter. Several different read latencies are shown to demonstrate the impact that technology choice has on the potential organizations.

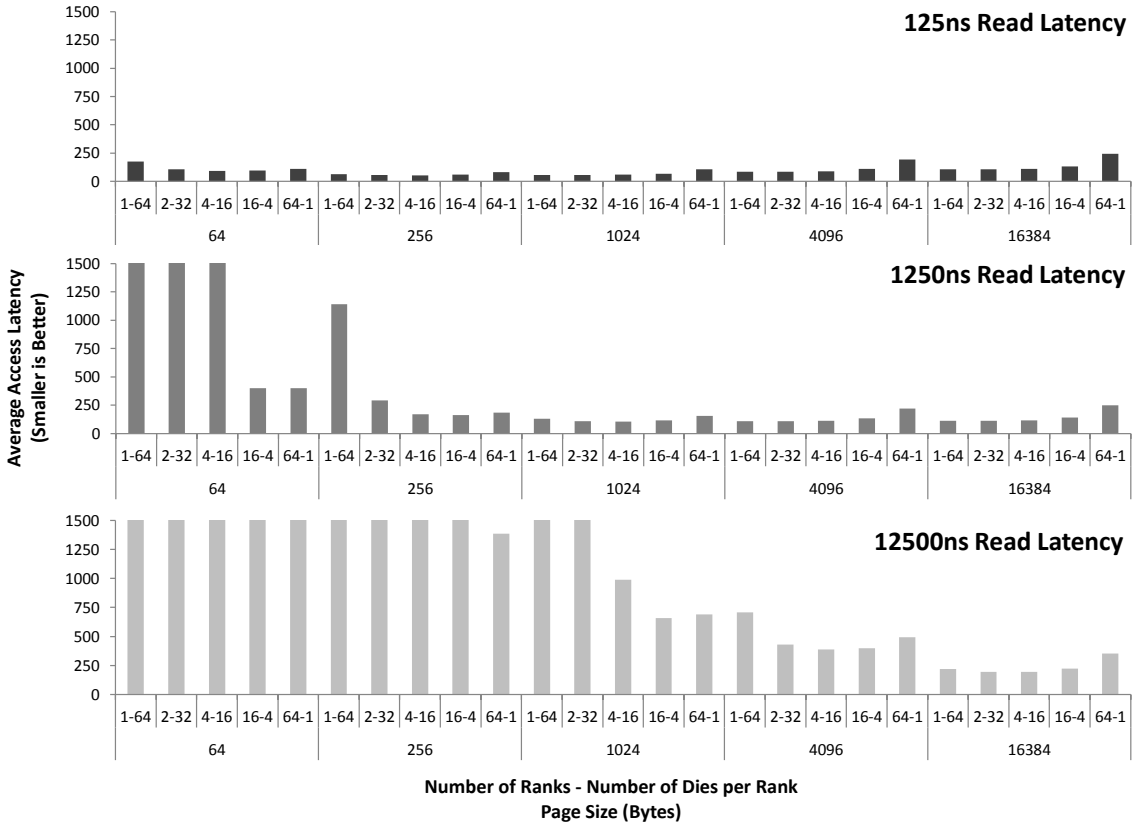

Figure 9.16: The average access latencies for *blackscholes* that result from using different sized ranks and different sized pages with a 256MB cache that is significantly smaller than all of the working sets of the benchmarks used in this chapter. Several different read latencies are shown to demonstrate the impact that technology choice has on the potential organizations.

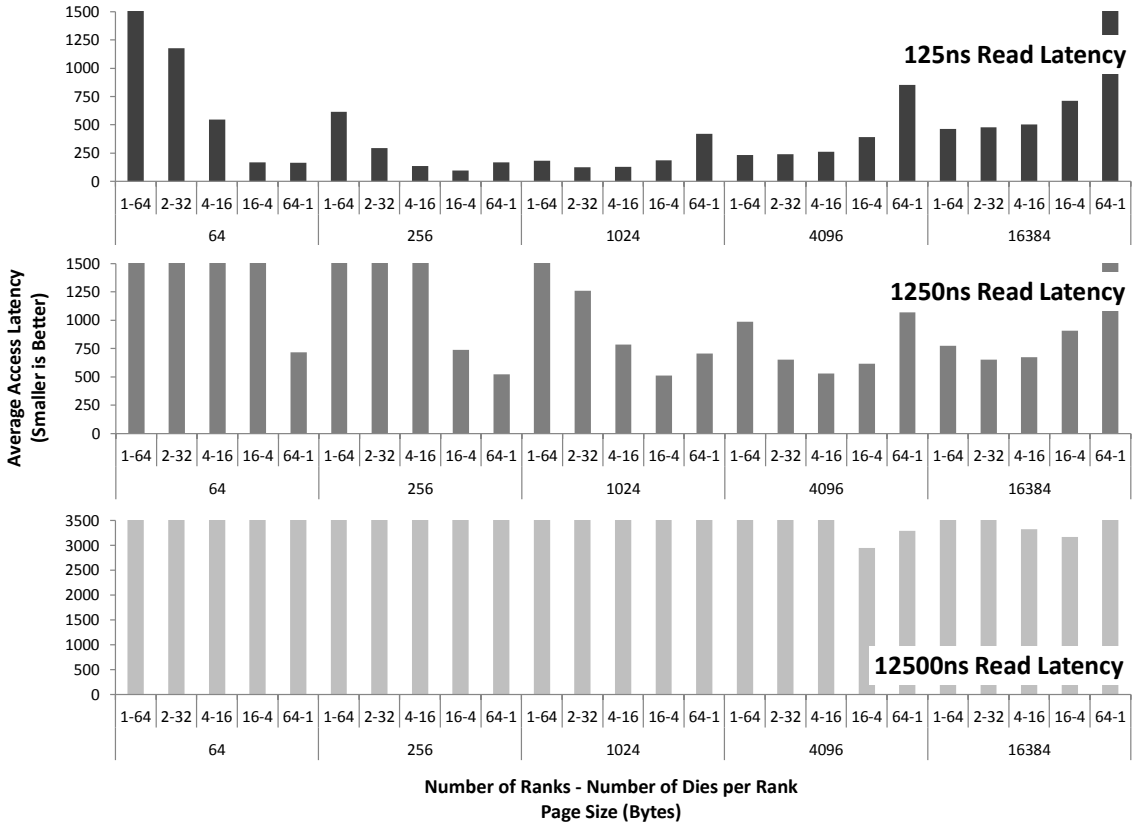

Figure 9.17: The average access latencies for *bodytrack* that result from using different sized ranks and different sized pages with a 256MB cache that is significantly smaller than all of the working sets of the benchmarks used in this chapter. Several different read latencies are shown to demonstrate the impact that technology choice has on the potential organizations.

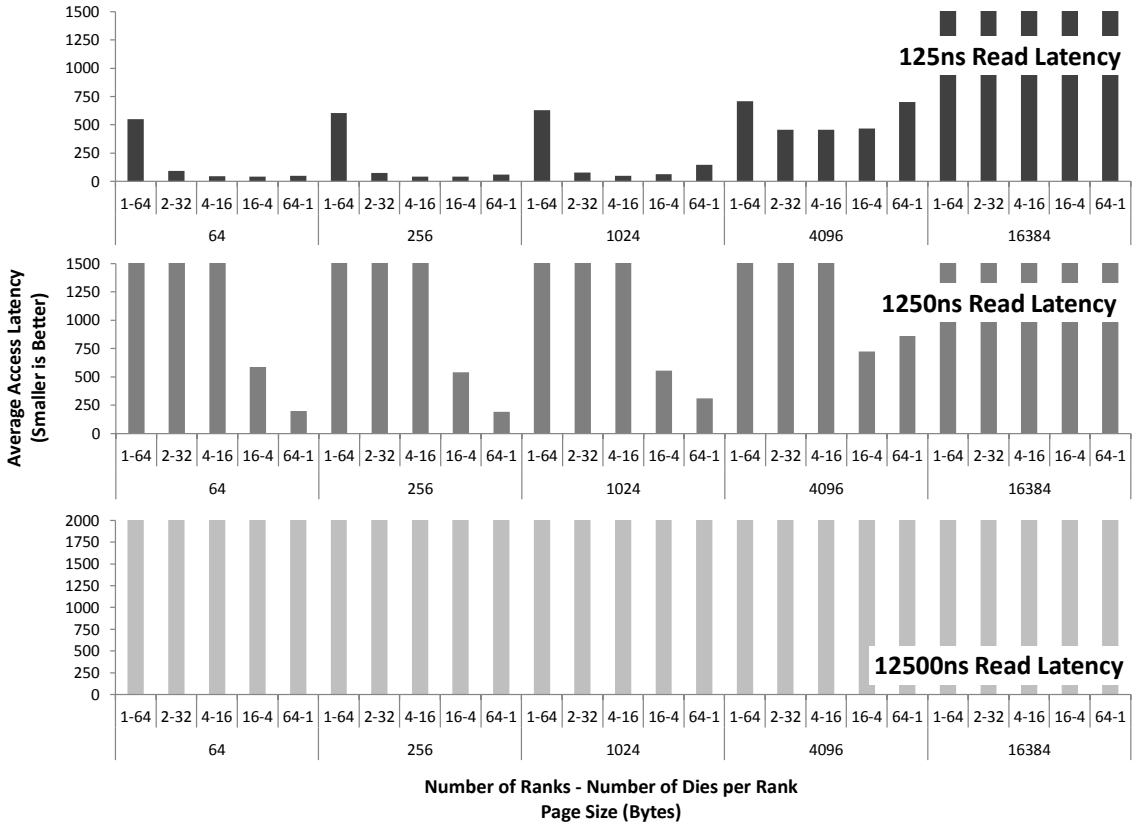

Figure 9.18: The average access latencies for *canneal* that result from using different sized ranks and different sized pages with a 256MB cache that is significantly smaller than all of the working sets of the benchmarks used in this chapter. Several different read latencies are shown to demonstrate the impact that technology choice has on the potential organizations.

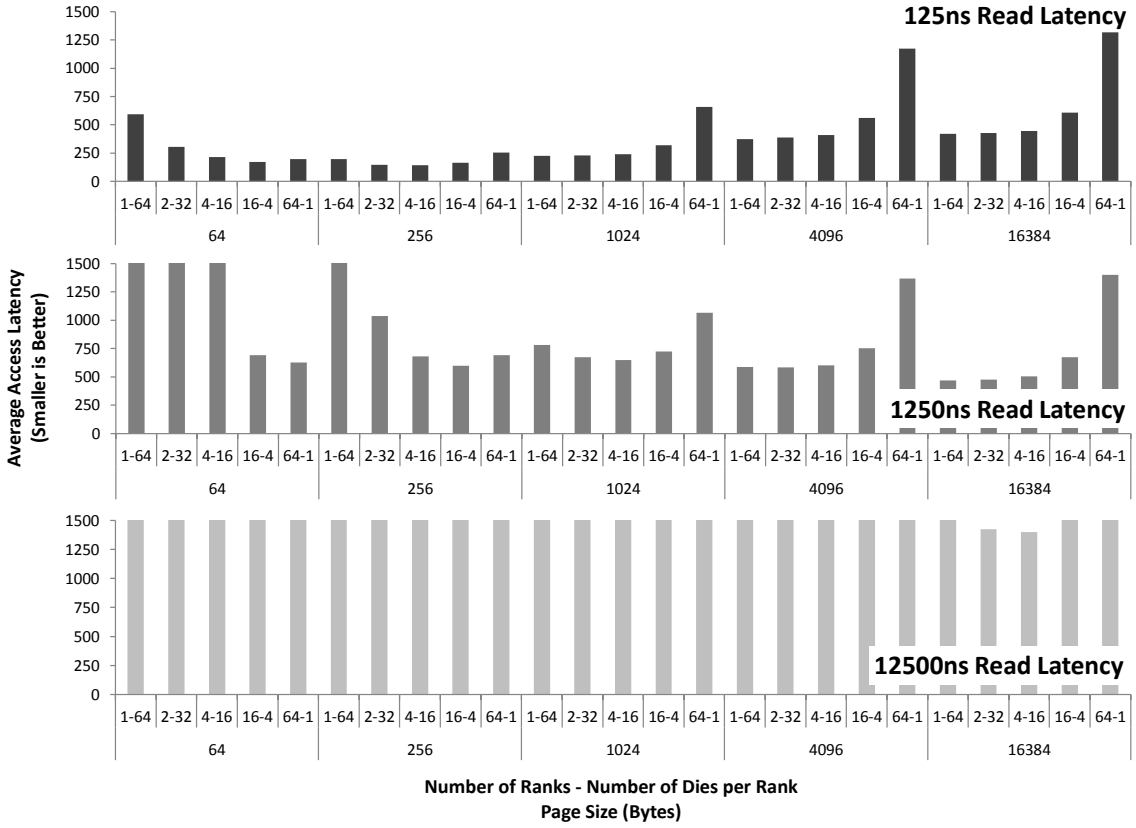

Figure 9.19: The average access latencies for *freqmine* that result from using different sized ranks and different sized pages with a 256MB cache that is significantly smaller than all of the working sets of the benchmarks used in this chapter. Several different read latencies are shown to demonstrate the impact that technology choice has on the potential organizations.

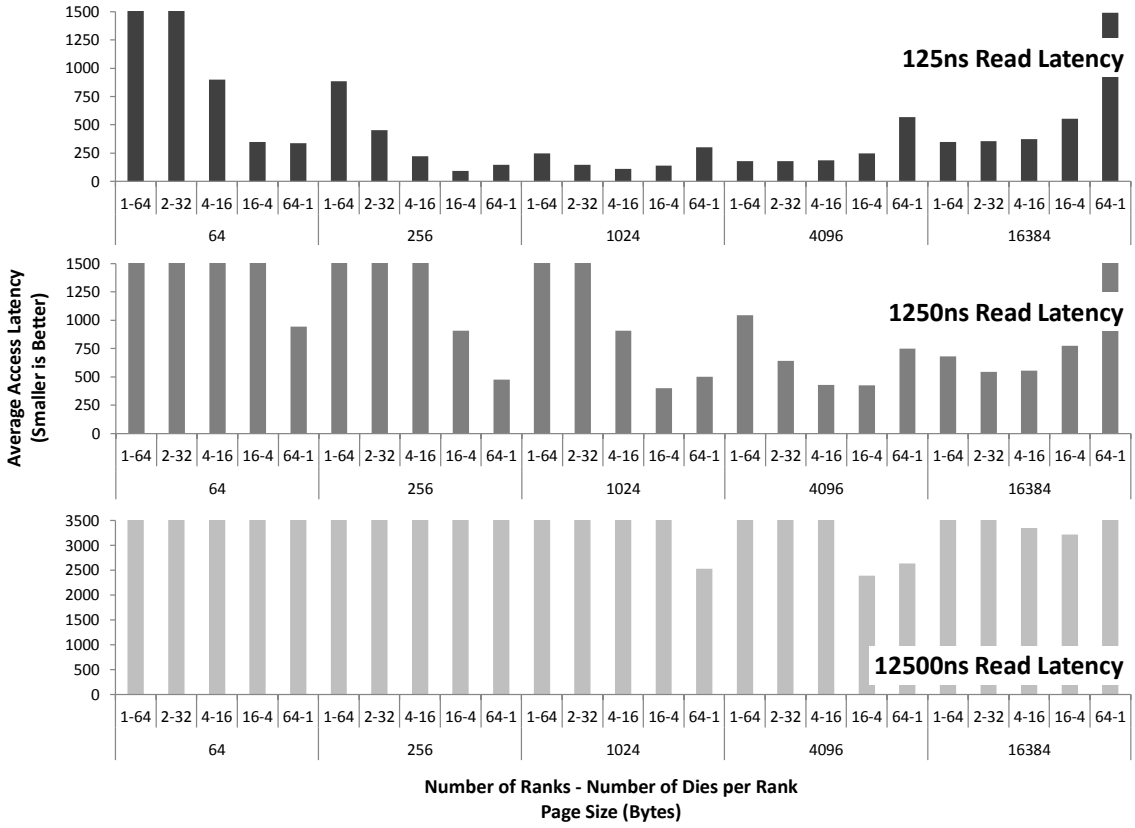

Figure 9.20: The average access latencies for *bzip2* that result from using different sized ranks and different sized pages with a 256MB cache that is significantly smaller than all of the working sets of the benchmarks used in this chapter. Several different read latencies are shown to demonstrate the impact that technology choice has on the potential organizations.

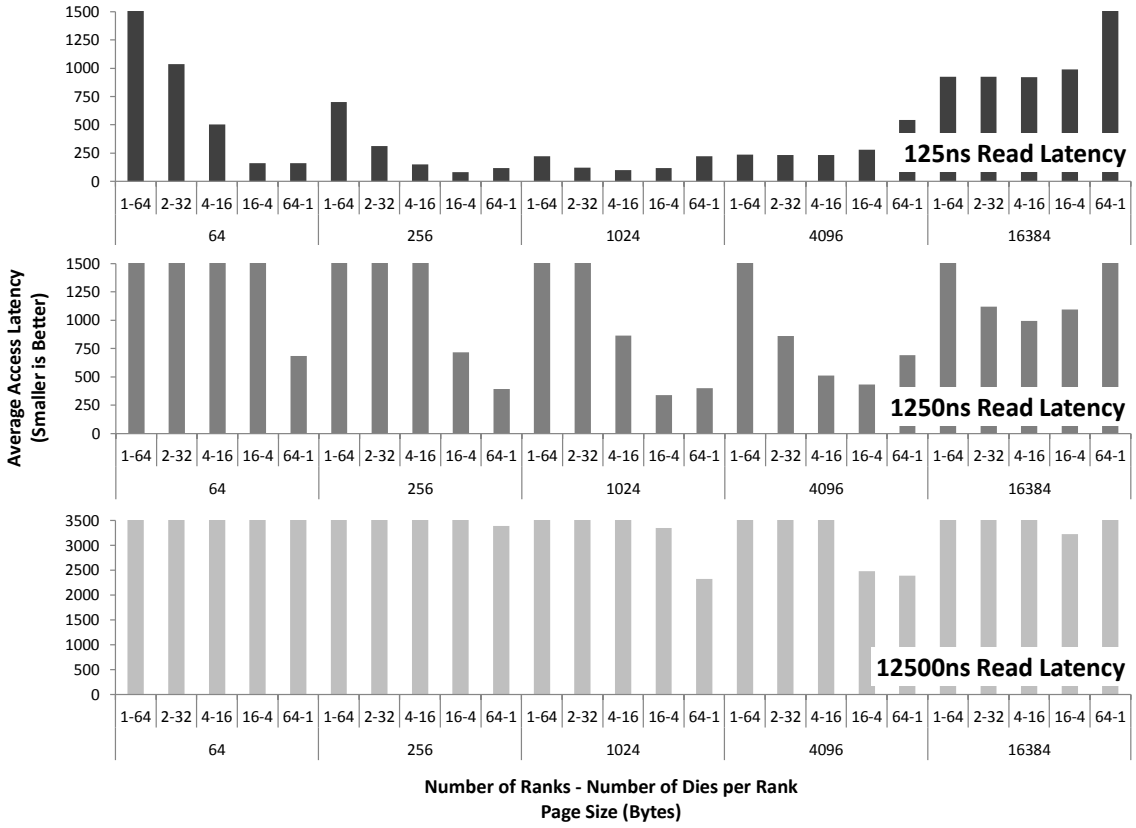

Figure 9.21: The average access latencies for *gcc* that result from using different sized ranks and different sized pages with a 256MB cache that is significantly smaller than all of the working sets of the benchmarks used in this chapter. Several different read latencies are shown to demonstrate the impact that technology choice has on the potential organizations.

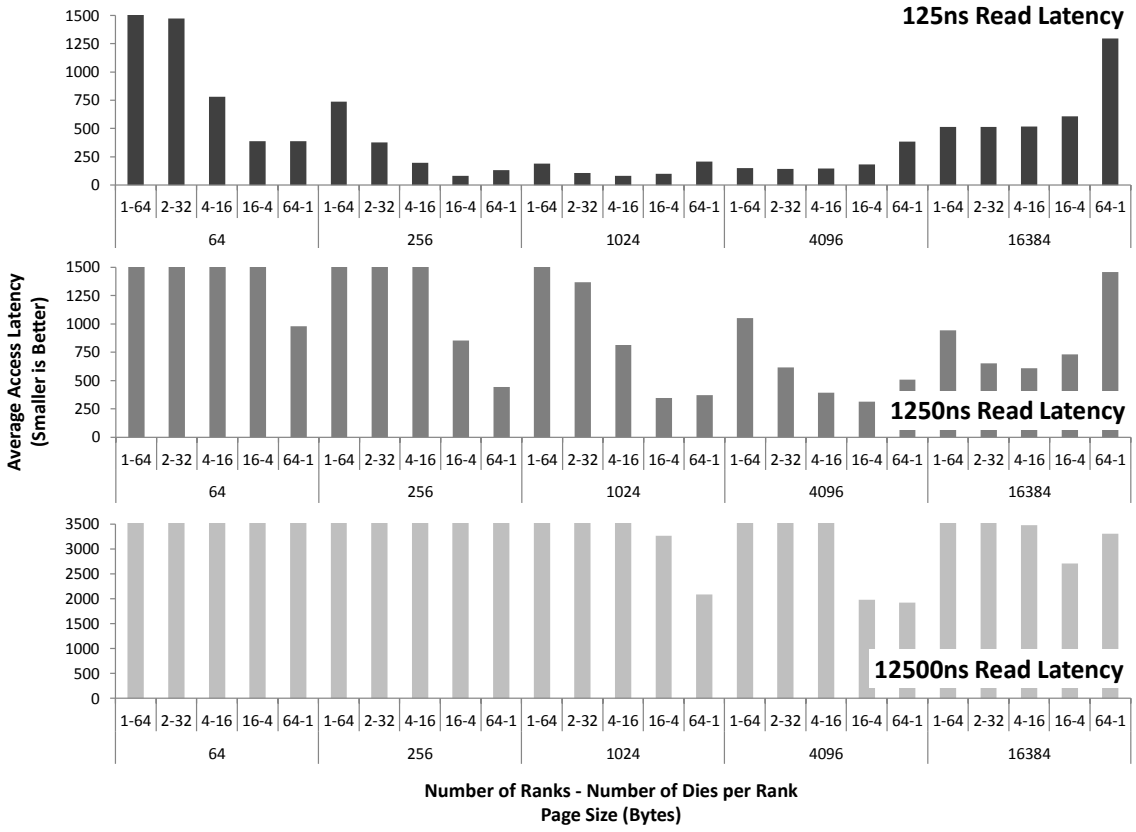

Figure 9.22: The average access latencies for *leslie3d* that result from using different sized ranks and different sized pages with a 256MB cache that is significantly smaller than all of the working sets of the benchmarks used in this chapter. Several different read latencies are shown to demonstrate the impact that technology choice has on the potential organizations.

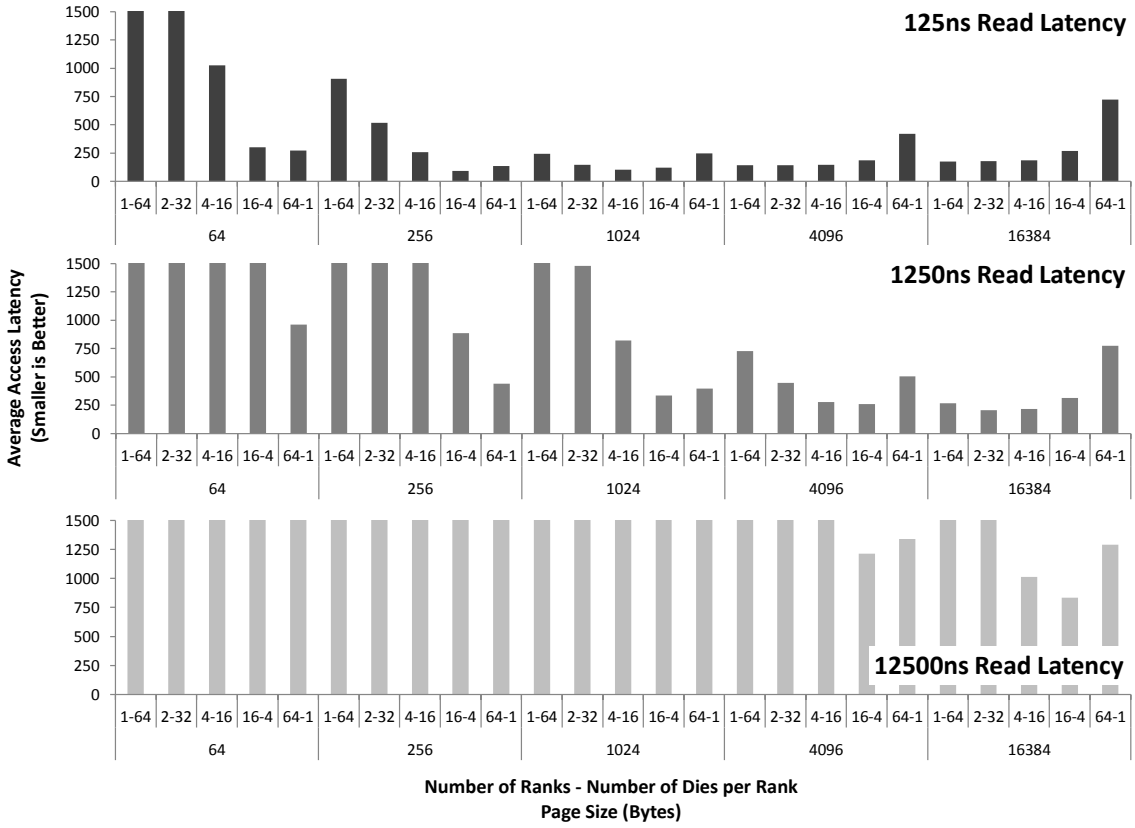

Figure 9.23: The average access latencies for *milc* that result from using different sized ranks and different sized pages with a 256MB cache that is significantly smaller than all of the working sets of the benchmarks used in this chapter. Several different read latencies are shown to demonstrate the impact that technology choice has on the potential organizations.

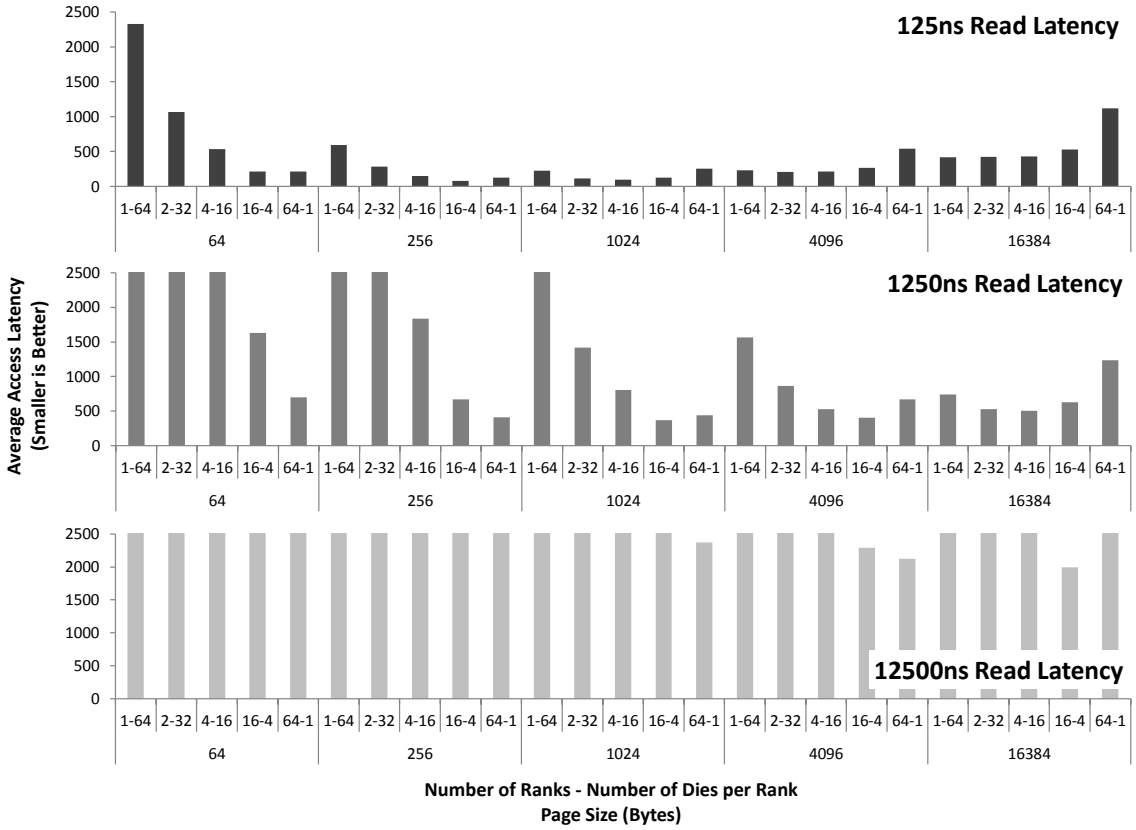

Figure 9.24: The average access latencies that result from using different sized ranks and different sized pages with a 256MB cache that is only slightly smaller than most of the working sets of the benchmarks used in this chapter. Several different read latencies are shown to demonstrate the impact that technology choice has on the potential organizations.

In the next study, pictured in Figure 9.24, we double the size of the cache to 256MB so that it is only slightly smaller than average workload size. The first thing to note in Figure 9.24 is that the larger cache size has significantly reduced the pressure on the backing store and the access latencies are nearly a third shorter as a result. This study shows many of the same trends as the 256MB study pictured

in Figure 9.12 for the 125ns and 1250ns memories. However, the 12500ns memory shows some slightly different effects than what we previously saw with the smaller cache. Specifically, the fastest organization for this type of memory has shifted from the 64 rank, 4KB page size organization in the previous experiment to the 16 rank, 16KB page size organization in this study. We believe that this indicates a dynamic between the pressure on the backing store, the amount of concurrency available, and the page size. Larger pages take longer to move off of a die and so tend to use up more of the available concurrency. When there is pressure on the backing store, that concurrency becomes more important to the overall performance of the system than the reduction in misses that can result from larger page sizes. As a result, when the backing store is under pressure it tends to perform better with smaller pages because the same degree of concurrency can serve a greater volume of accesses.

9.2.3 512MB Cache

In the last rank and page size experiments, we double the size of the cache again to 512MB so that it is now larger than most of the workloads. The results from the experiments using this cache can be seen in Figure 9.36. The most interesting aspect of this study is that the differences between the different organizations appear to be much less significant than they were in the previous tests. This suggests that as the pressure on the backing store is relaxed, the impact of both concurrency and bandwidth are reduced.

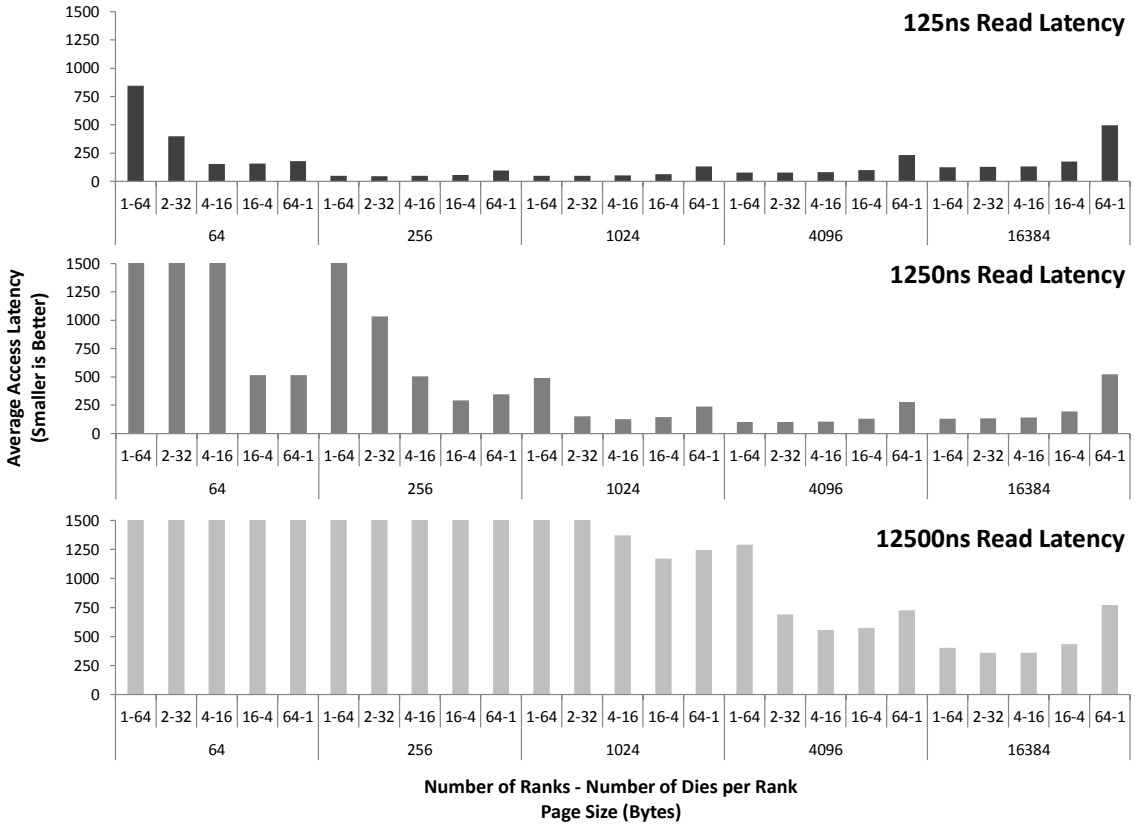

Figure 9.25: The average access latencies for *ft* that result from using different sized ranks and different sized pages with a 512MB cache that is significantly smaller than all of the working sets of the benchmarks used in this chapter. Several different read latencies are shown to demonstrate the impact that technology choice has on the potential organizations.

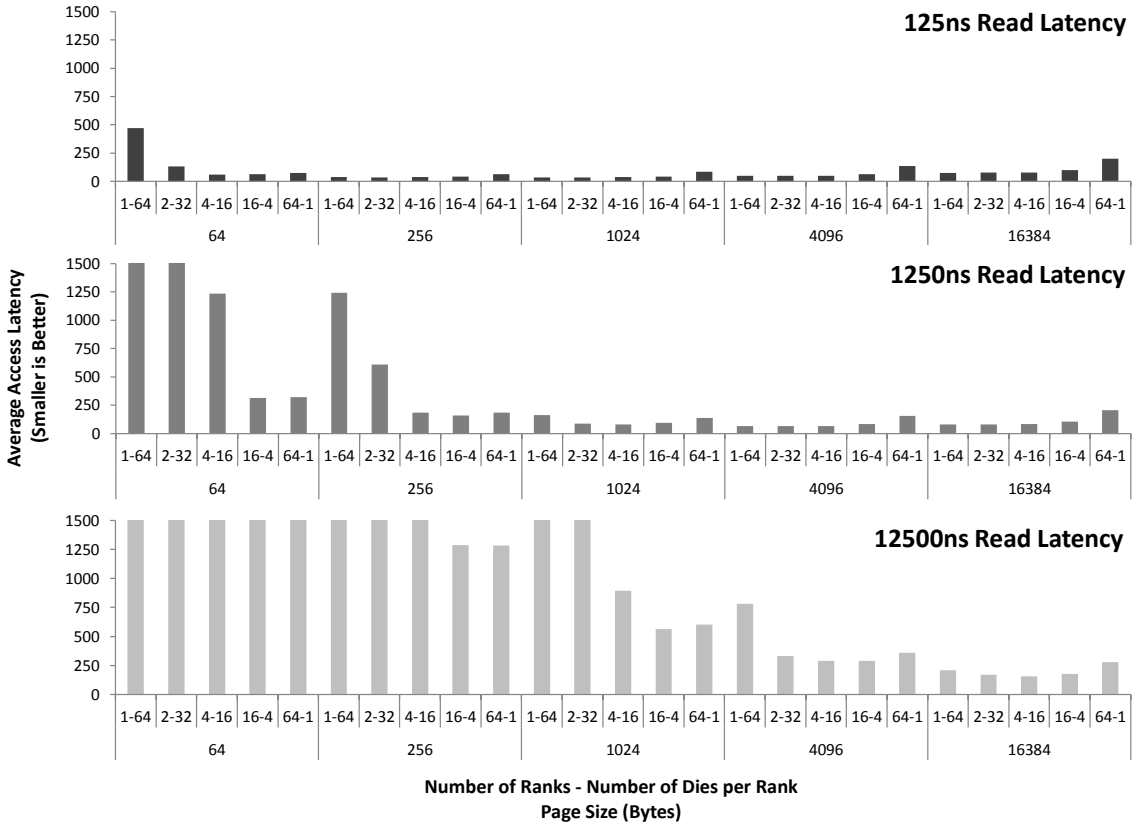

Figure 9.26: The average access latencies for *is* that result from using different sized ranks and different sized pages with a 512MB cache that is significantly smaller than all of the working sets of the benchmarks used in this chapter. Several different read latencies are shown to demonstrate the impact that technology choice has on the potential organizations.

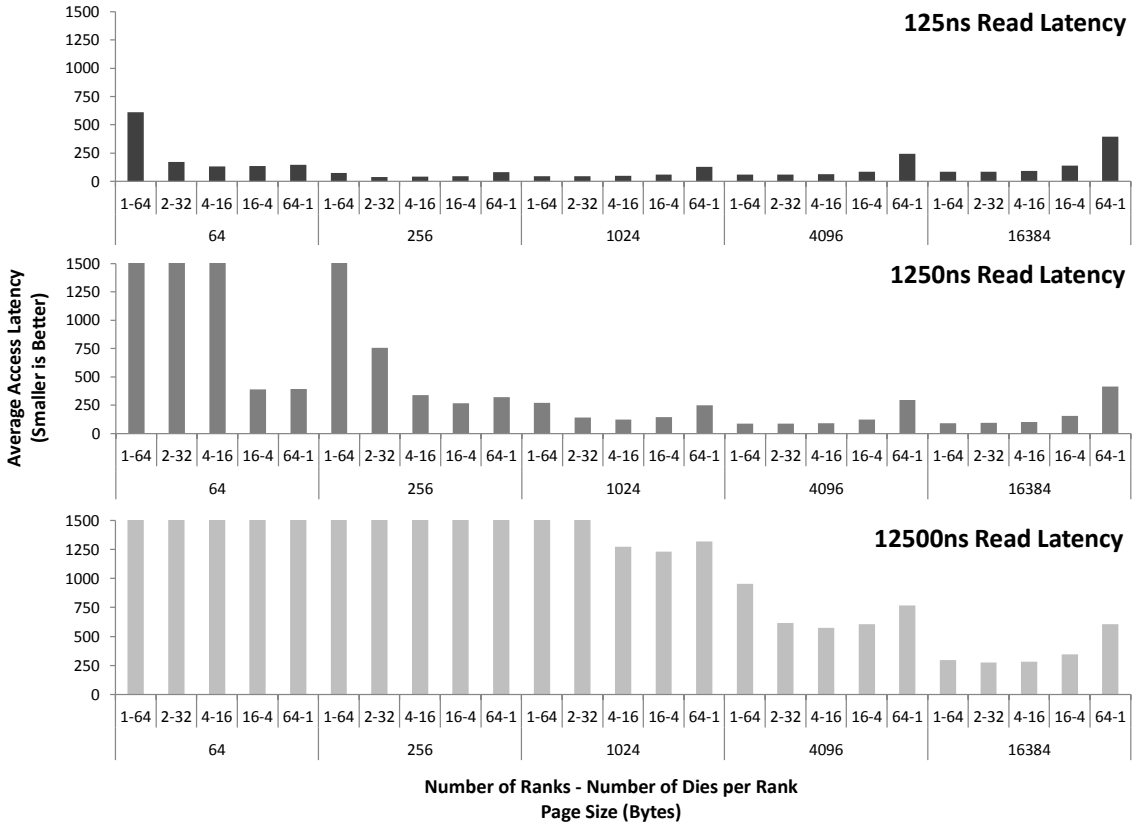

Figure 9.27: The average access latencies for *mg* that result from using different sized ranks and different sized pages with a 512MB cache that is significantly smaller than all of the working sets of the benchmarks used in this chapter. Several different read latencies are shown to demonstrate the impact that technology choice has on the potential organizations.

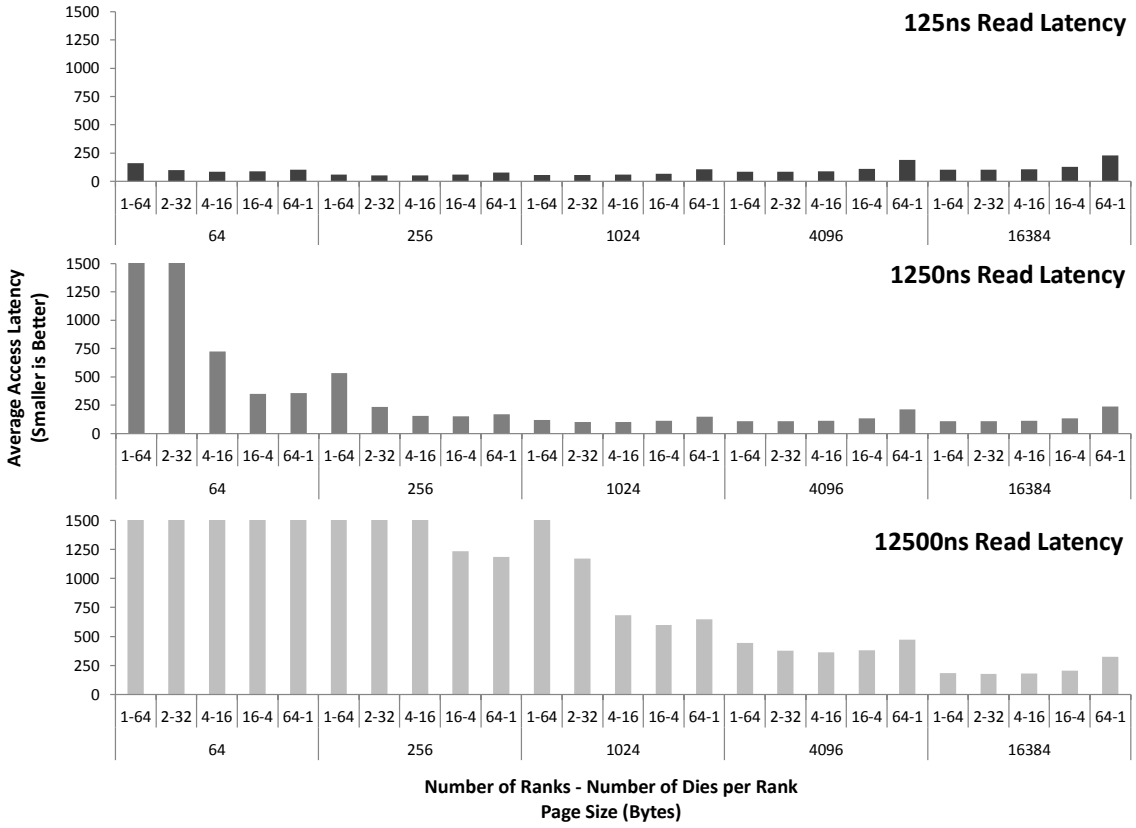

Figure 9.28: The average access latencies for *blackscholes* that result from using different sized ranks and different sized pages with a 512MB cache that is significantly smaller than all of the working sets of the benchmarks used in this chapter. Several different read latencies are shown to demonstrate the impact that technology choice has on the potential organizations.

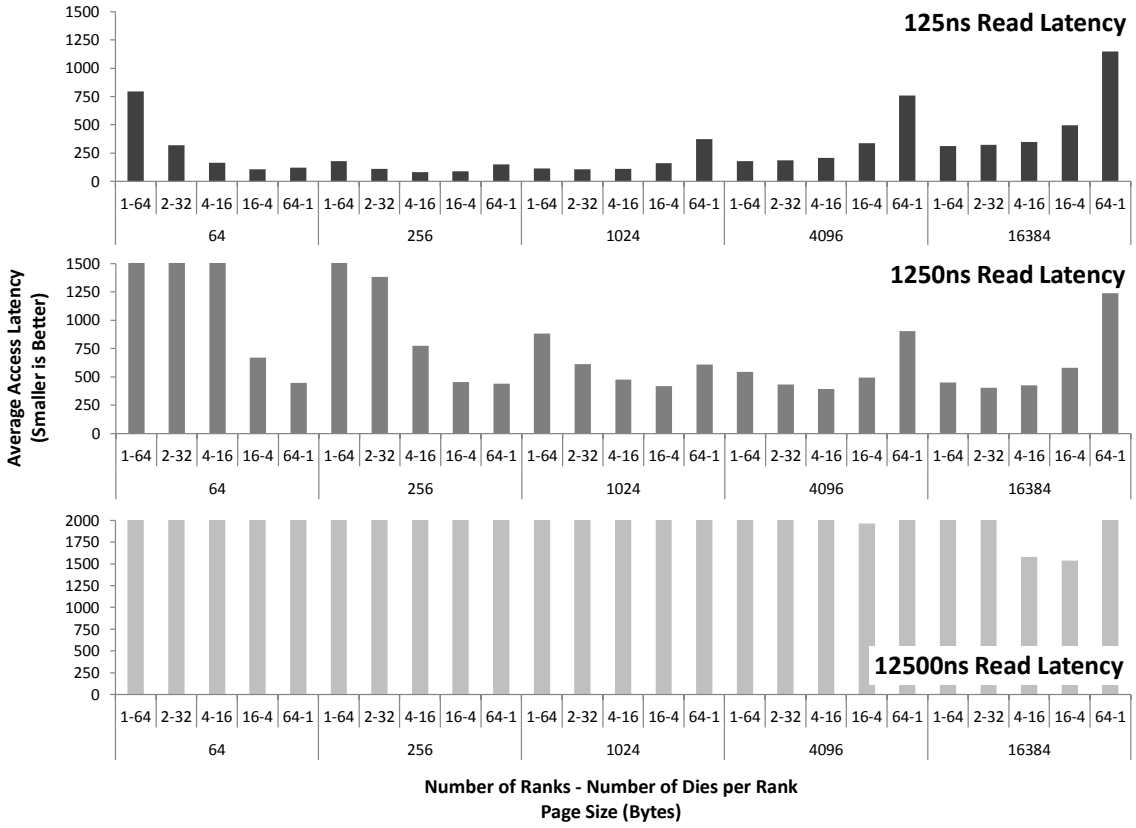

Figure 9.29: The average access latencies for *bodytrack* that result from using different sized ranks and different sized pages with a 512MB cache that is significantly smaller than all of the working sets of the benchmarks used in this chapter. Several different read latencies are shown to demonstrate the impact that technology choice has on the potential organizations.

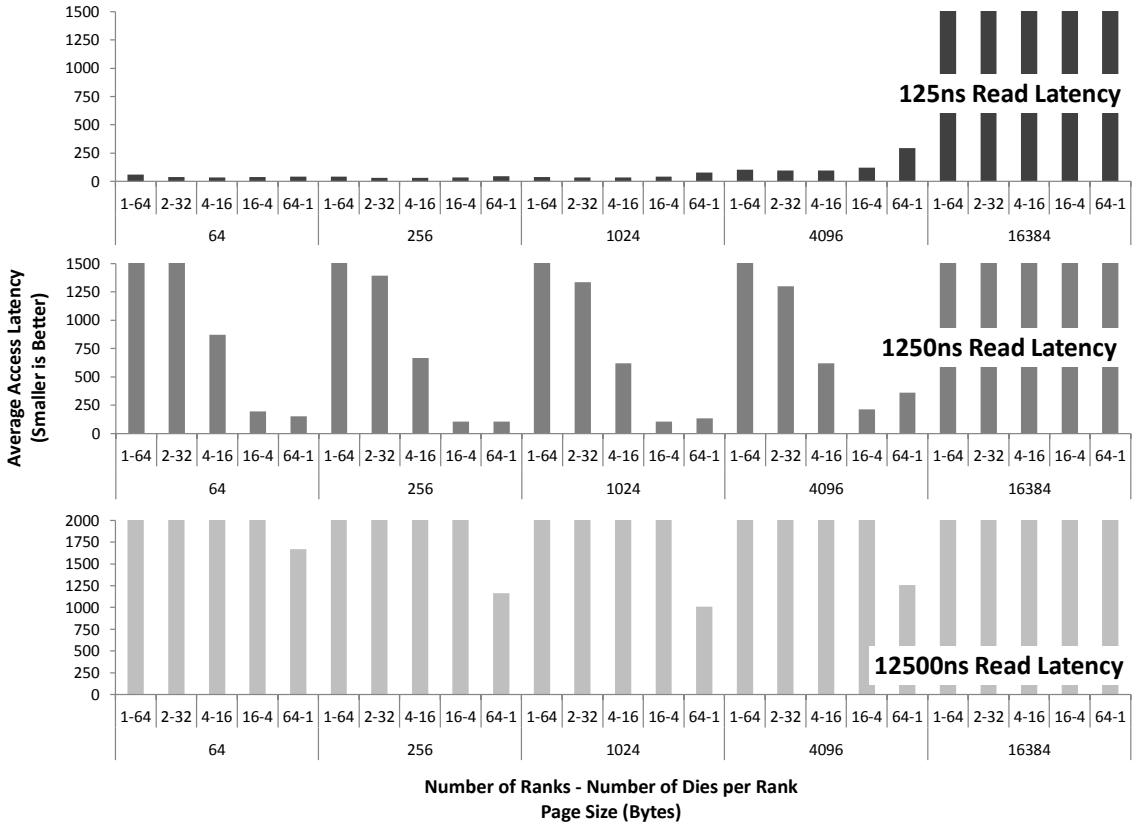

Figure 9.30: The average access latencies for *canneal* that result from using different sized ranks and different sized pages with a 512MB cache that is significantly smaller than all of the working sets of the benchmarks used in this chapter. Several different read latencies are shown to demonstrate the impact that technology choice has on the potential organizations.

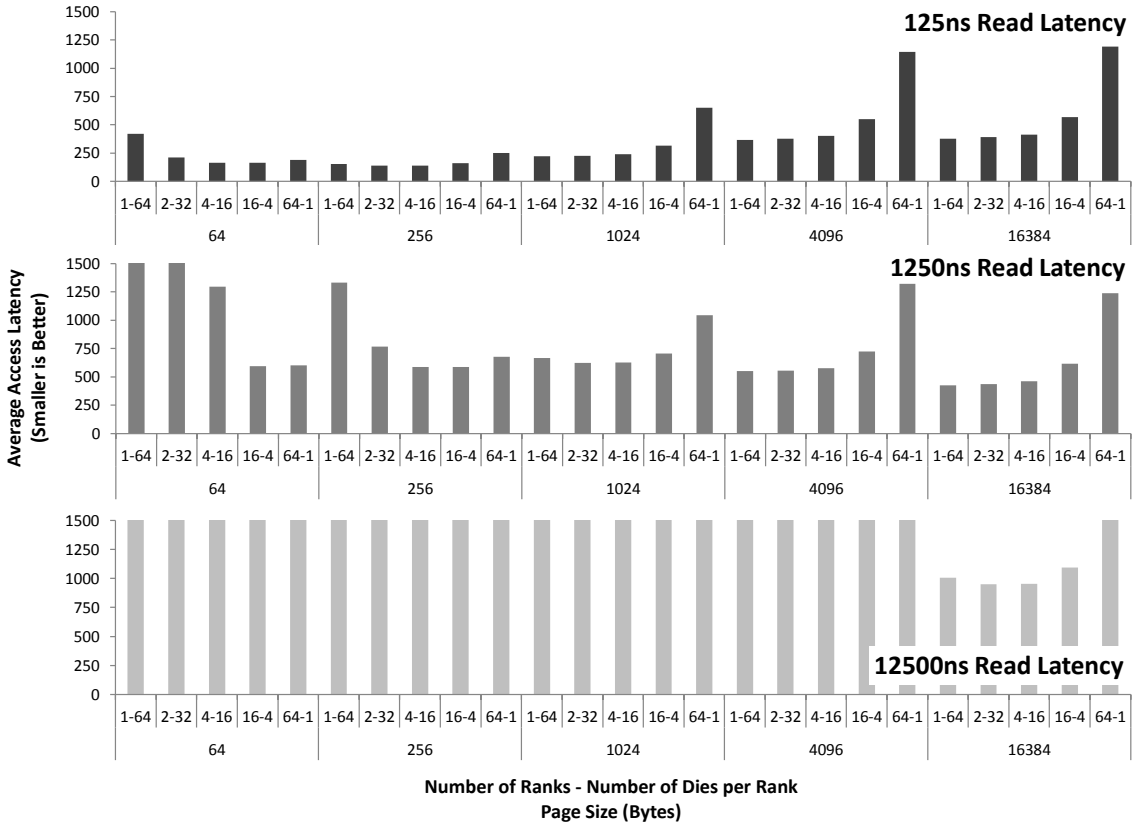

Figure 9.31: The average access latencies for *freqmine* that result from using different sized ranks and different sized pages with a 512MB cache that is significantly smaller than all of the working sets of the benchmarks used in this chapter. Several different read latencies are shown to demonstrate the impact that technology choice has on the potential organizations.

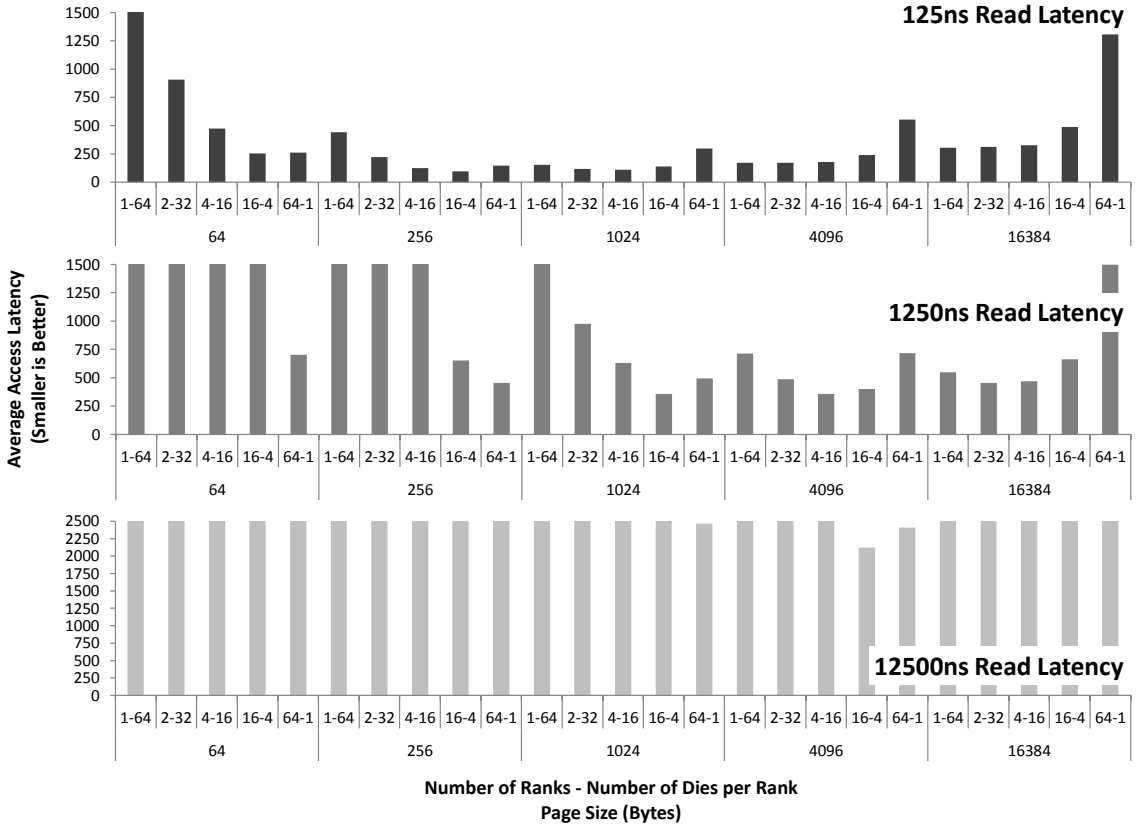

Figure 9.32: The average access latencies for *bzip2* that result from using different sized ranks and different sized pages with a 512MB cache that is significantly smaller than all of the working sets of the benchmarks used in this chapter. Several different read latencies are shown to demonstrate the impact that technology choice has on the potential organizations.

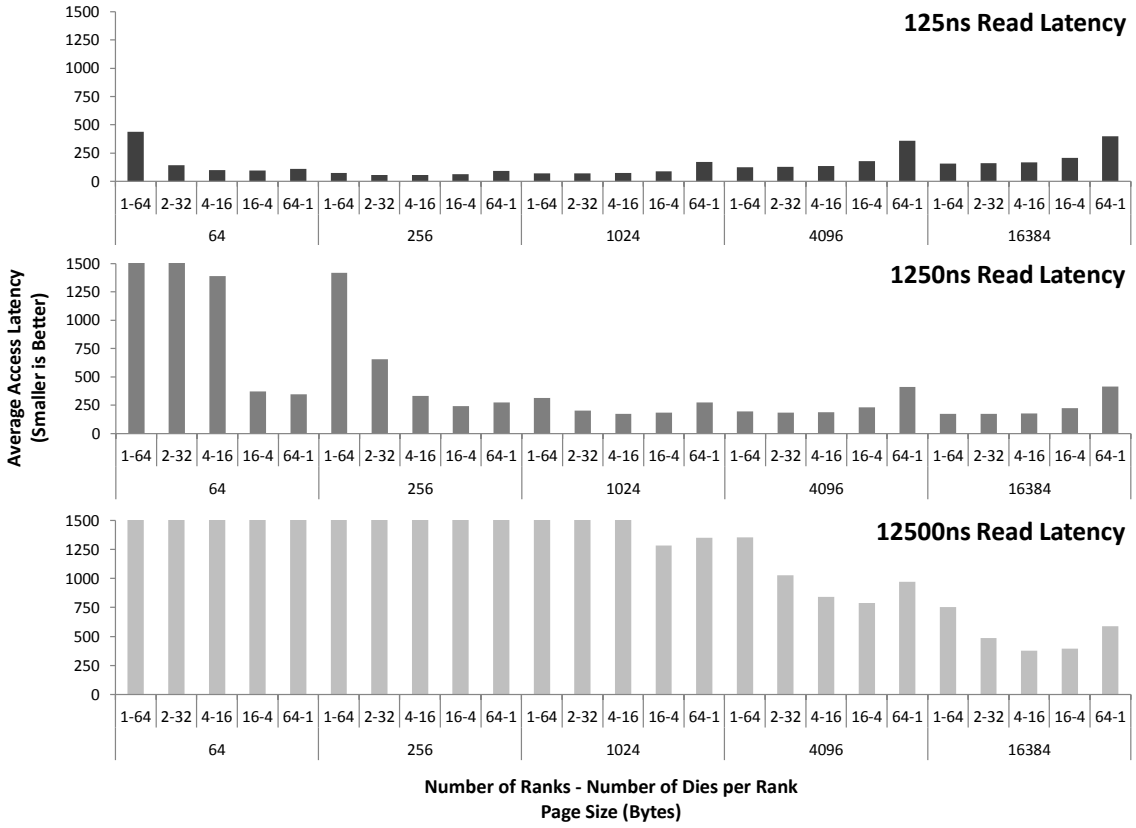

Figure 9.33: The average access latencies for *gcc* that result from using different sized ranks and different sized pages with a 512MB cache that is significantly smaller than all of the working sets of the benchmarks used in this chapter. Several different read latencies are shown to demonstrate the impact that technology choice has on the potential organizations.

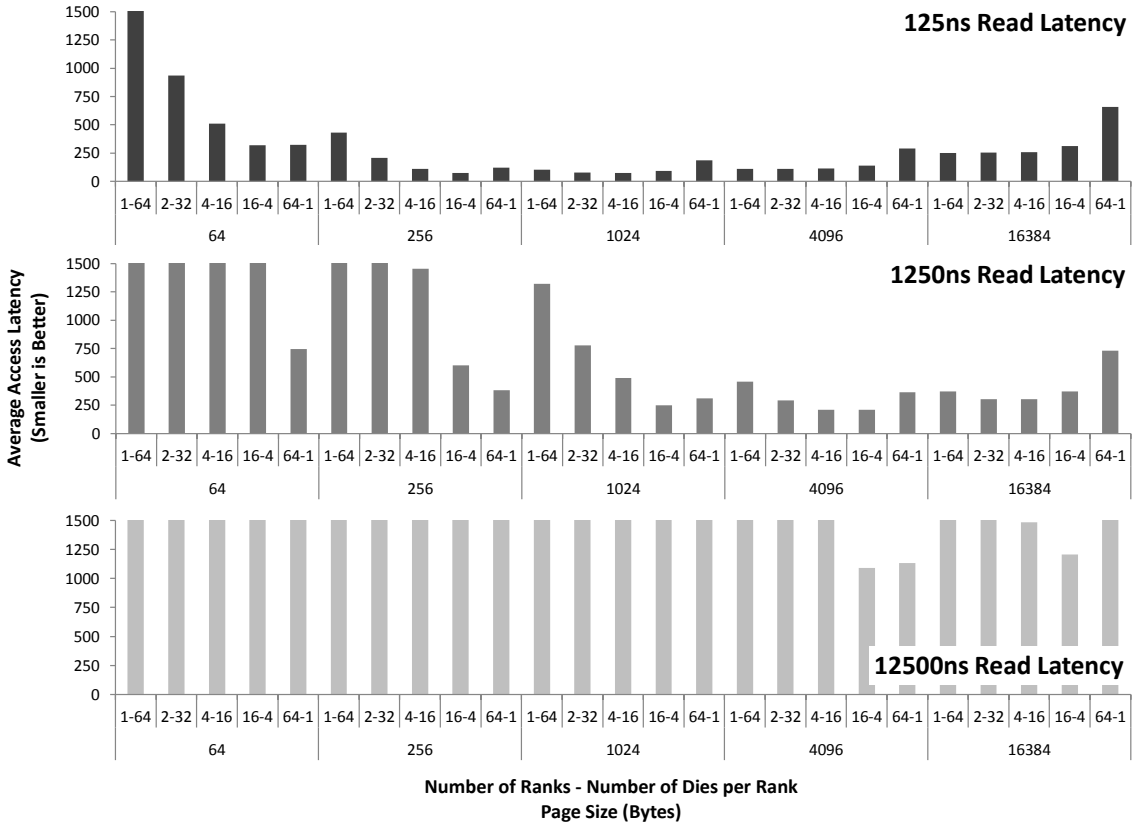

Figure 9.34: The average access latencies for *leslie3d* that result from using different sized ranks and different sized pages with a 512MB cache that is significantly smaller than all of the working sets of the benchmarks used in this chapter. Several different read latencies are shown to demonstrate the impact that technology choice has on the potential organizations.

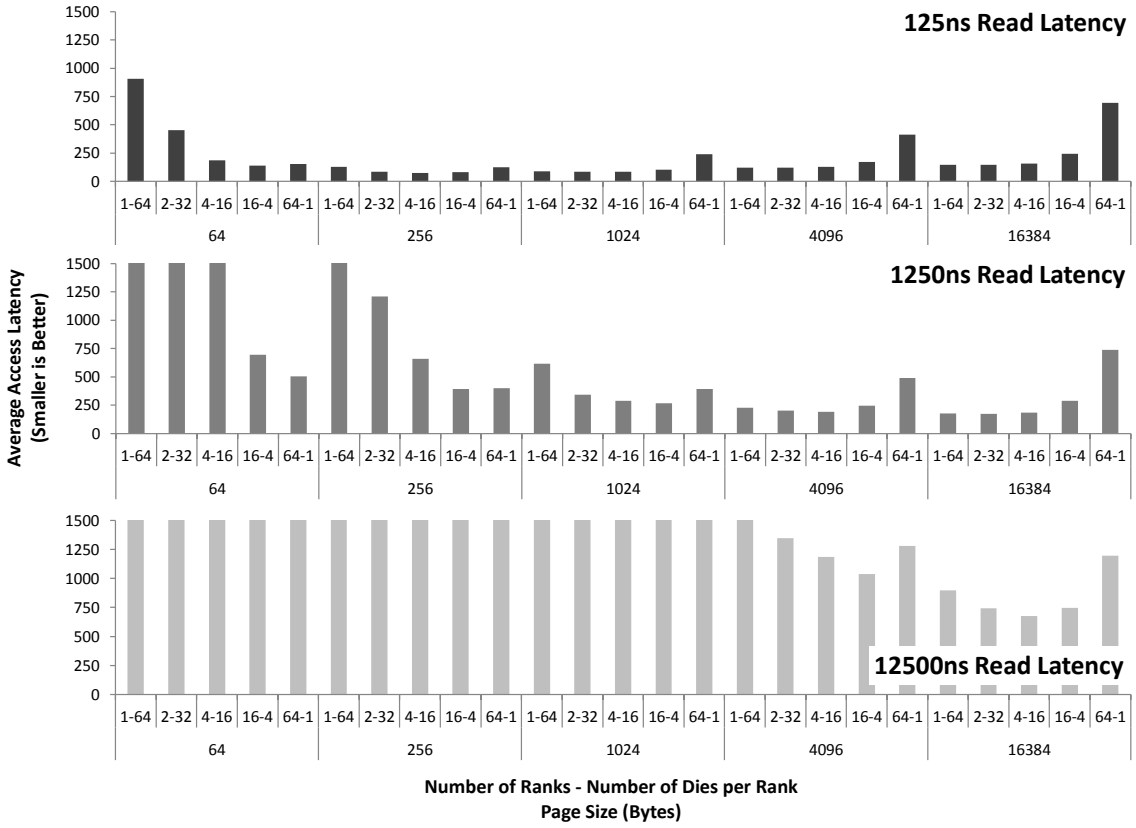

Figure 9.35: The average access latencies for *milc* that result from using different sized ranks and different sized pages with a 512MB cache that is significantly smaller than all of the working sets of the benchmarks used in this chapter. Several different read latencies are shown to demonstrate the impact that technology choice has on the potential organizations.

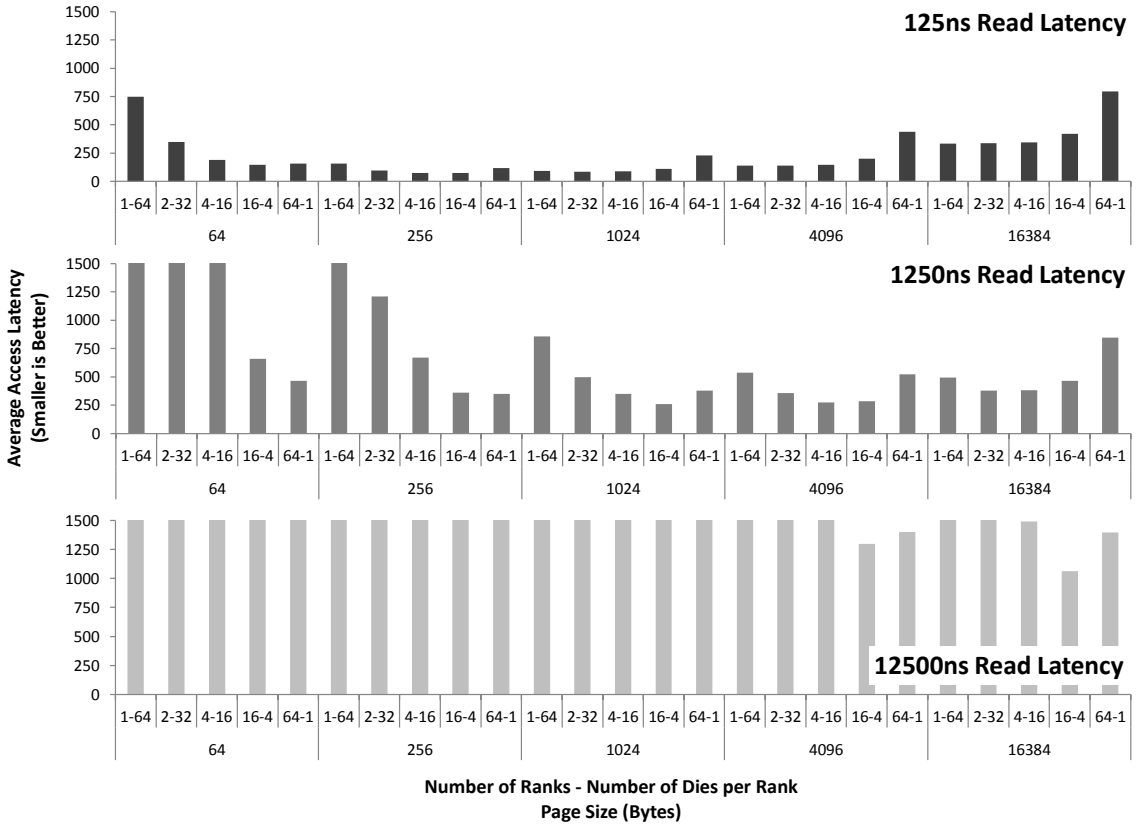

Figure 9.36: The average access latencies that result from using different sized ranks and different sized pages with a 512MB cache that is larger than most of the working sets of the benchmarks used in this chapter. Several different read latencies are shown to demonstrate the impact that technology choice has on the potential organizations.

9.3 Prefetching

One way to reduce the miss ratio of a cache and thereby improve overall performance is to prefetch data from the backing store in order to bring it into the cache before it is explicitly requested. By prefetching data based on previous access patterns, it is possible to leverage spatial locality and frequently bring useful data into the cache before it is needed. However, prefetching does not always bring in useful data and can use up bandwidth and concurrency that would have been put to better use serving actual requests. Furthermore, prefetching can also lead to cache pollution which is caused by filling the cache with useless data while evicting potentially useful data.

In the following experiments we use a simple sequential prefetching algorithm to evaluate the potential benefits that can result from reducing the miss rate of the cache. This algorithm simply prefetches the next n addresses worth of data after the requested address every time a miss occurs. We use a 16 rank, 4 die organization for all of these experiments because that organization appeared to most consistently outperform the other organizations regardless of memory speed or page size. In addition, we utilize a 256MB cache for these experiments because this size generates sufficient miss pressure on the backing store while being less susceptible to cache pollution than a 128MB cache.

Figure 9.40 shows the latency results of our prefetching experiments. We normalize these results to the fastest observed latency for each of the three memory technologies in order to gauge the speedup that was achieved by introducing

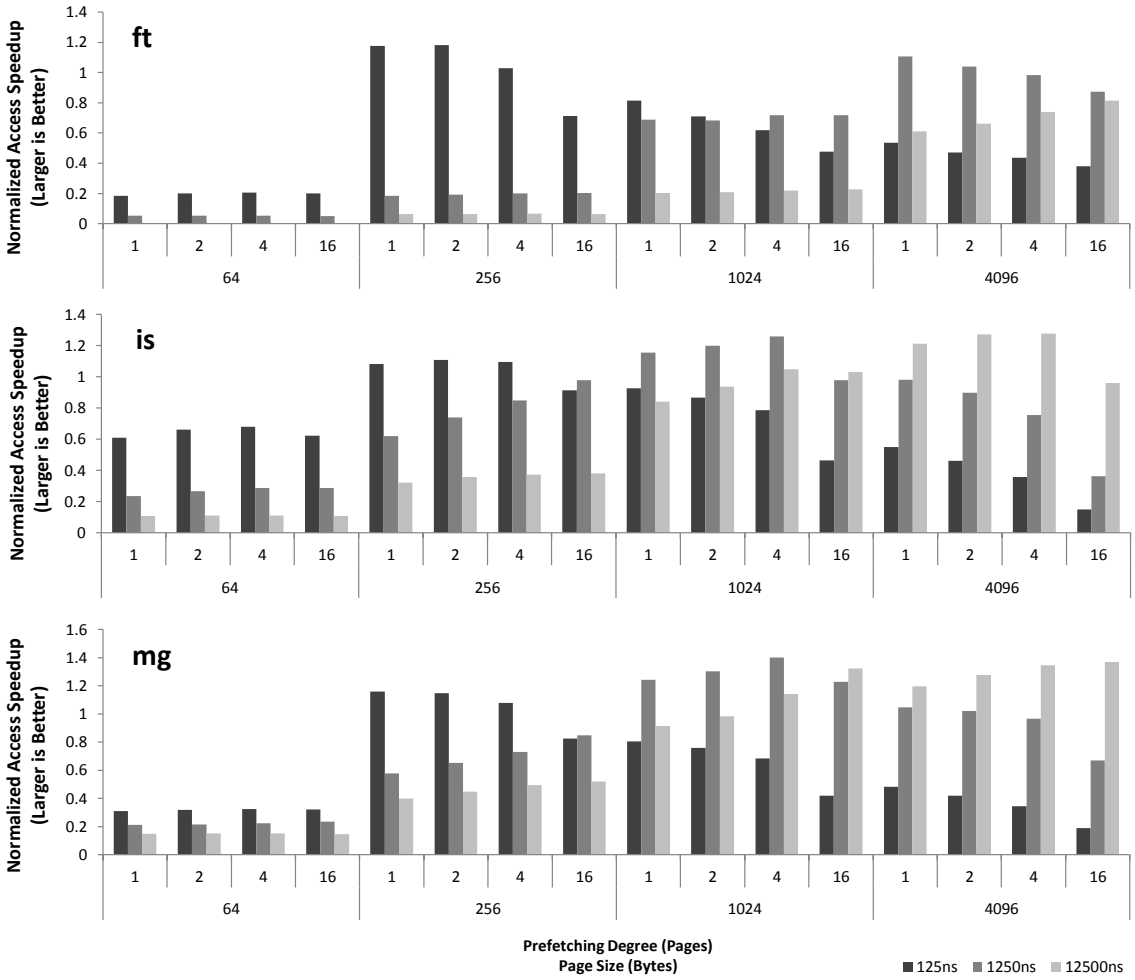

Figure 9.37: The average access latencies for the NPB workloads that result from different degrees of prefetching normalized to the base average access latency that was observed without prefetching for each workload. The ranks organizations used in these experiments are the best values for each read latency as observed in the ranks versus page size experiments.

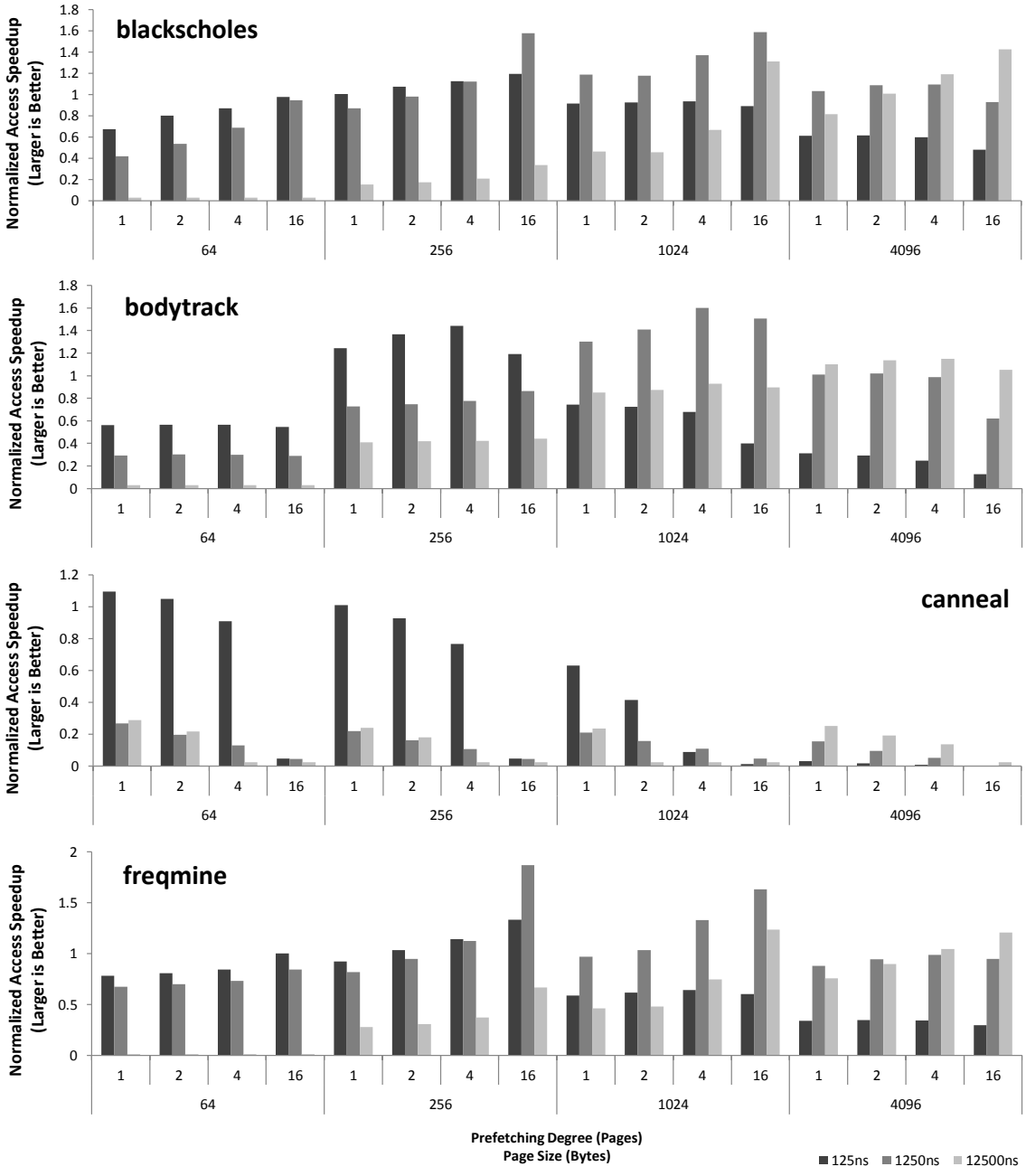

Figure 9.38: The average access latencies for the PARSEC workloads that result from different degrees of prefetching normalized to the base average access latency that was observed without prefetching for each workload. The ranks organizations used in these experiments are the best values for each read latency as observed in the ranks versus page size experiments.

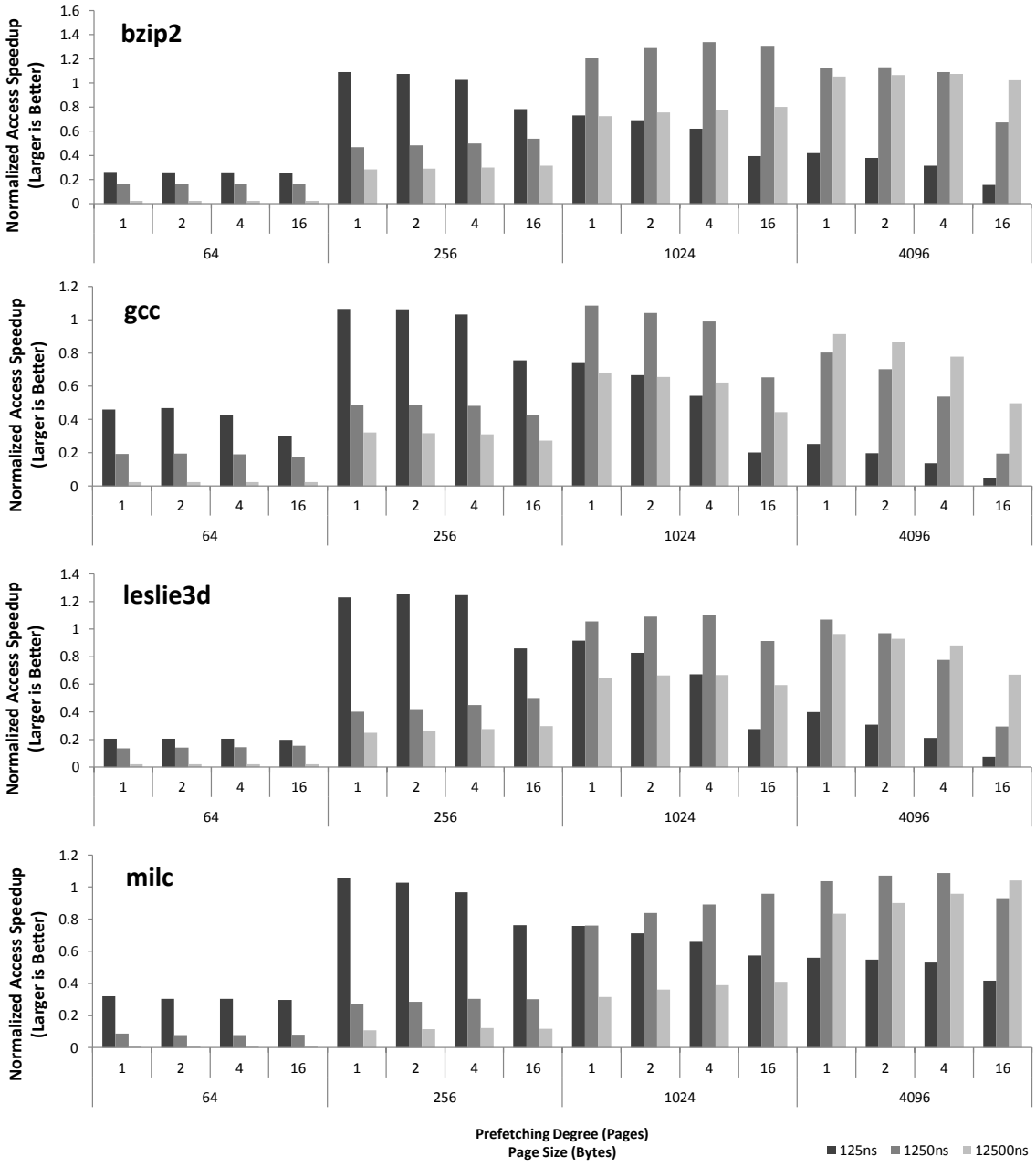

Figure 9.39: The average access latencies for the SPEC workloads that result from different degrees of prefetching normalized to the base average access latency that was observed without prefetching for each workload. The ranks organizations used in these experiments are the best values for each read latency as observed in the ranks versus page size experiments.

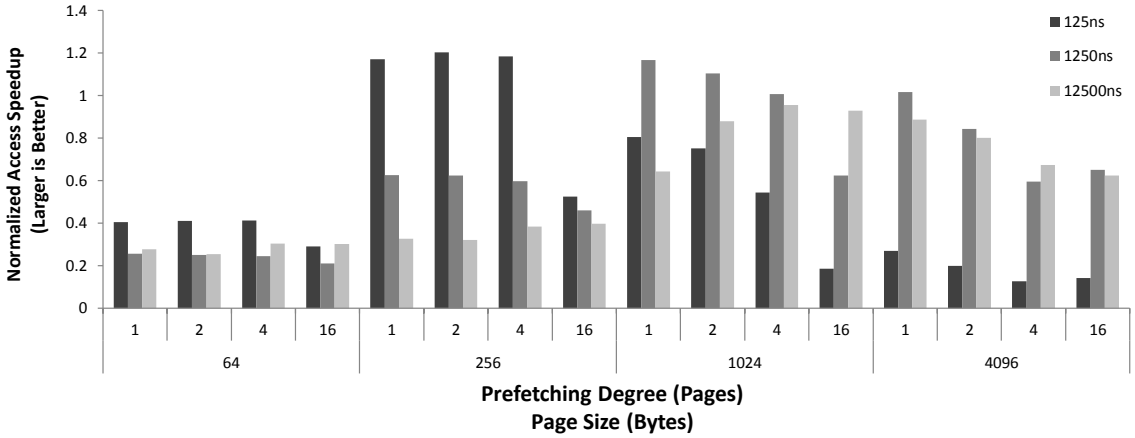

Figure 9.40: The average access latencies that result from different degrees of prefetching normalized to the base average access latency that was observed without prefetching. The ranks organizations used in these experiments are the best values for each read latency as observed in the ranks versus page size experiments.

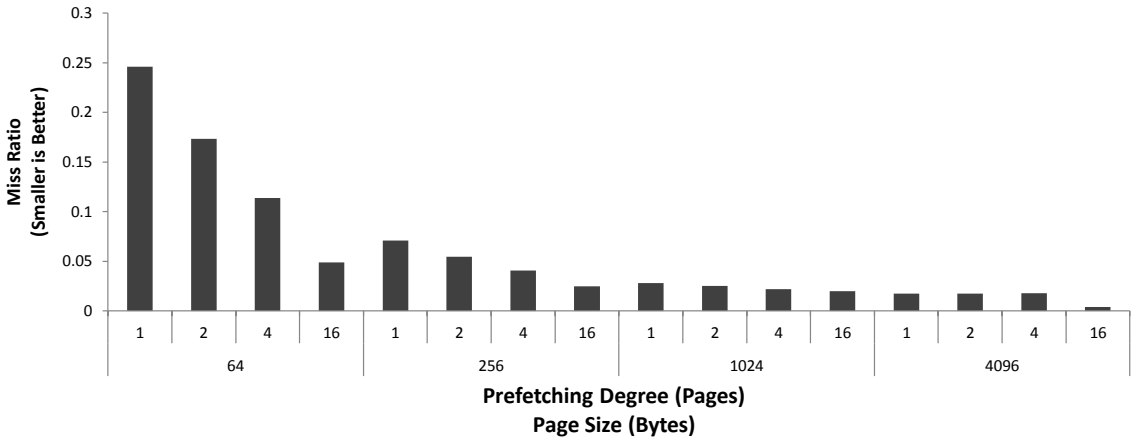

Figure 9.41: The average miss ratios that result from different degrees of prefetching. The ranks organizations used in these experiments are the best values for each read latency as observed in the ranks versus page size experiments.

prefetching. One hypothesis that we had going into this experiment was that by prefetching smaller pages, it would be possible to surpass the benefits that were seen when large pages were used in the previous study. The thought here is that prefetching would allow the same amount of data to be acquired on each access but the smaller pages would allow for better utilization of the available cache capacity. Large pages can often lead to wasted cache capacity because significant portions of a large page can go unused but the page cannot be evicted to make space for useful data because other parts of the large page are being used. We see that our hypothesis was true for the slowest memory which achieved a performance that was roughly equivalent to its 16KB page performance with 1KB pages and a 4 page prefetching degree. This is interesting because the total amount of data being transfered in both cases is not equivalent. We would have expected the 4KB page, 4 page prefetching degree setup to do as well as the 16KB page because in both cases 16KB are fetched each time. We believe that this is due to a trade off between hit rate and bandwidth utilization that tends to favor hit rate when the backing store latency is long and the miss penalty is severe. Prefetching spreads the accesses out across ranks though and so it winds up utilizing less bandwidth per individual device to access the data. In the 12500ns case, this appears to result in a better utilization of the available concurrency that, in turn, improves performance.

However, from the other results in Figure 9.40 we can see that our hypothesis does not hold true for the faster memories. In the case of the 125ns memory, the optimal page size based on the previous studies was 256B. In this test, the 256B page once again provided the best performance but also did benefit somewhat from

limited prefetching. The same trend can also be observed for the 1250ns memory which did best with the 1KB page size in both this study and the previous one. This suggests that there was some available bandwidth and concurrency in the system that could be used by prefetching to improve performance. However, the drop in performance at the 16 page prefetching degree shows that there is a definite limit to the available bandwidth and concurrency.

We also include the miss ratio results for these experiments in Figure 9.41. These results show that the improved performance in the previous experiments can be attributed to the reduction in misses provided by prefetching. Increasing the prefetching degree reduced the miss ratio in all cases. This means that the cases where prefetching does not provide a performance benefit are due to a bandwidth and concurrency over utilization and not cache pollution.

9.4 Channel Organization

In all of the studies that we have done thus far, we have assumed that the memory devices were directly connected to the memory controller. This allowed us to focus on other important aspects of the system without introducing system bandwidth as a potential bottleneck. However, in a real system attaching all of the memory devices directly to the memory controller is simply not an option. The number of pins required to accomplish this would be too great and the resulting cost of the system would be too high. Instead, narrow host channels are typically shared by multiple devices.

For these experiments we investigate the effect of adding a host channel on the performance of our backing store. Sharing high speed buses between many devices ultimately results in less bandwidth being provided per device if the backing store memory traffic is particularly severe. However, if the traffic is more moderate, it may be possible to transparently share the channel between the devices provided the channel bandwidth is sufficiently higher than the device bandwidth. The goal of these experiments is to determine if it is possible to effectively share a host channel in this way. In addition, we also investigate the benefits of different channel organizations to see if separate request/response buses or separate address buses can make better use of the available pin count.

The architecture that we utilize in these experiments places a buffer between the host interface and the device interfaces. This is similar to several research proposals that have attempted to utilize a buffer to decouple the host channel architecture from the device channel architecture [40–42]. It also bears some resemblance to the FBDIMM and LRDIMM designs. We utilize this channel architecture because it allows us to investigate a wide range of host channel organizations while not having to modify the device channel.

We look at the effects of three different channel organizations on the total system performance in these tests. First, we look at the effect of adding a separate command/address bus to the devices like DRAM devices have. This is not a common feature in Flash devices and so we were interested to see if it would provide any noticeable benefits for the slower memory technologies. We also investigate the benefits of full-duplex versus half-duplex bus organizations when the pin count is

held constant. As a result, the half-duplex channel has twice as much bandwidth going in one direction compared to the full duplex. The question is whether that bandwidth will be more useful than simultaneous bi-directional communication. Finally, we also consider the effects of splitting off the addresses and commands from the data and putting them on their own bus. This could help improve access times by ensuring that commands and data do not block one another. Unlike the other bus organizations, we do not split off the command and address pins from the data pins, instead we simply add a small number of additional pins to the system for this separate command/address bus. As the results will show, the bandwidth impact of these additional pins is minimal.

We perform these experiments for two different speeds of host channel to determine the impact that the overall host channel bandwidth has on the effectiveness of the different organizations.

The results of our channel organization experiments are presented in Figures 9.48 and 9.49. In these graphs we can see clearly that the fastest memory technology is the most sensitive to variations in the host channel architecture. This is because the transfer time makes up a much more significant portion of the overall delay for that memory. Also, the greatest impact seems to result from adding a host channel to the device. This provided a reasonable speedup in both the 125ns and 1250ns cases suggesting that all but the slower memory technologies can benefit from a slightly more complex device interface. Interestingly, the other parameters, host channel organization and host channel frequency, don't appear to have much of an effect on the performance of the system in most cases. The only exception to this

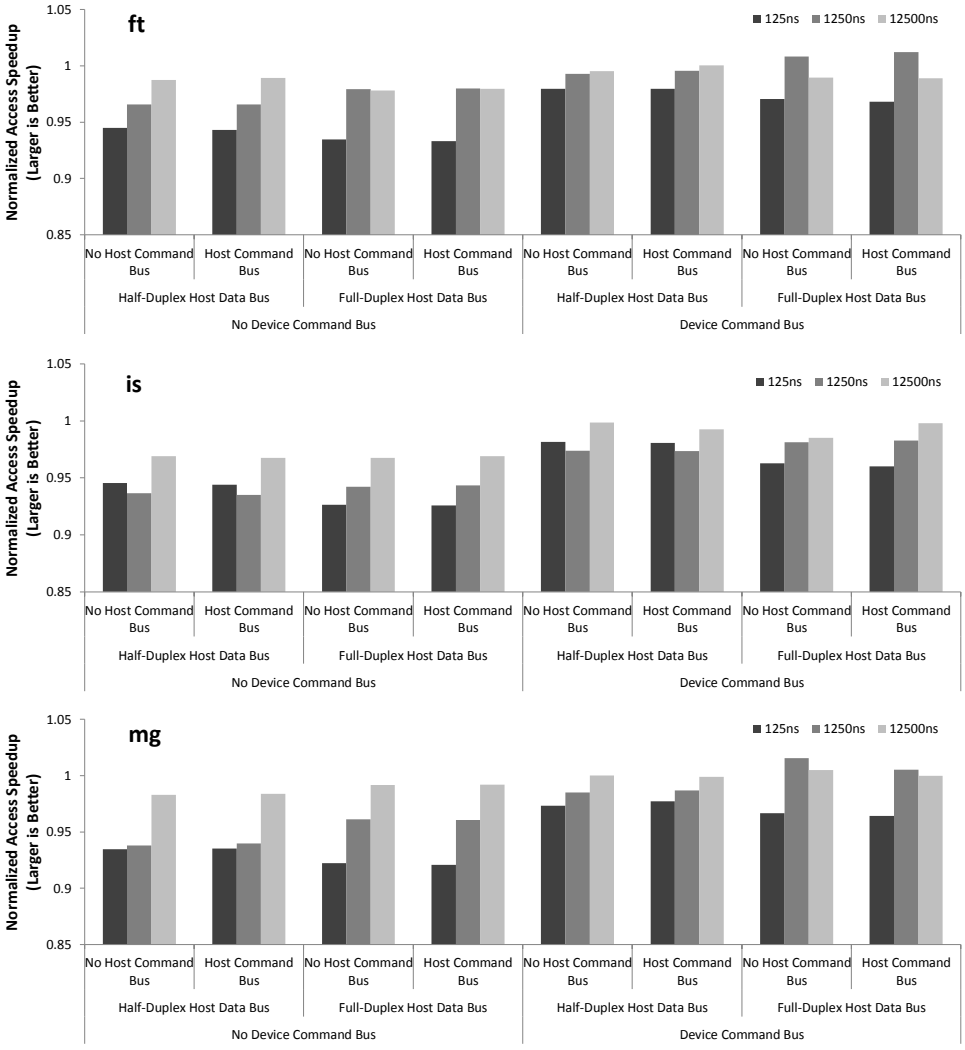

Figure 9.42: The average access latencies for the NPB workloads that result from different host and device channel organizations normalized to the average access latency that was observed with all ranks directly connected to the host and prefetching enabled. The ranks organizations, page sizes, and prefetching degree used in these experiments are the best values for each read latency as observed in the ranks versus page size and prefetching experiments. The host channel in this experiment is operating at a frequency of 400MHz (DDR-800).

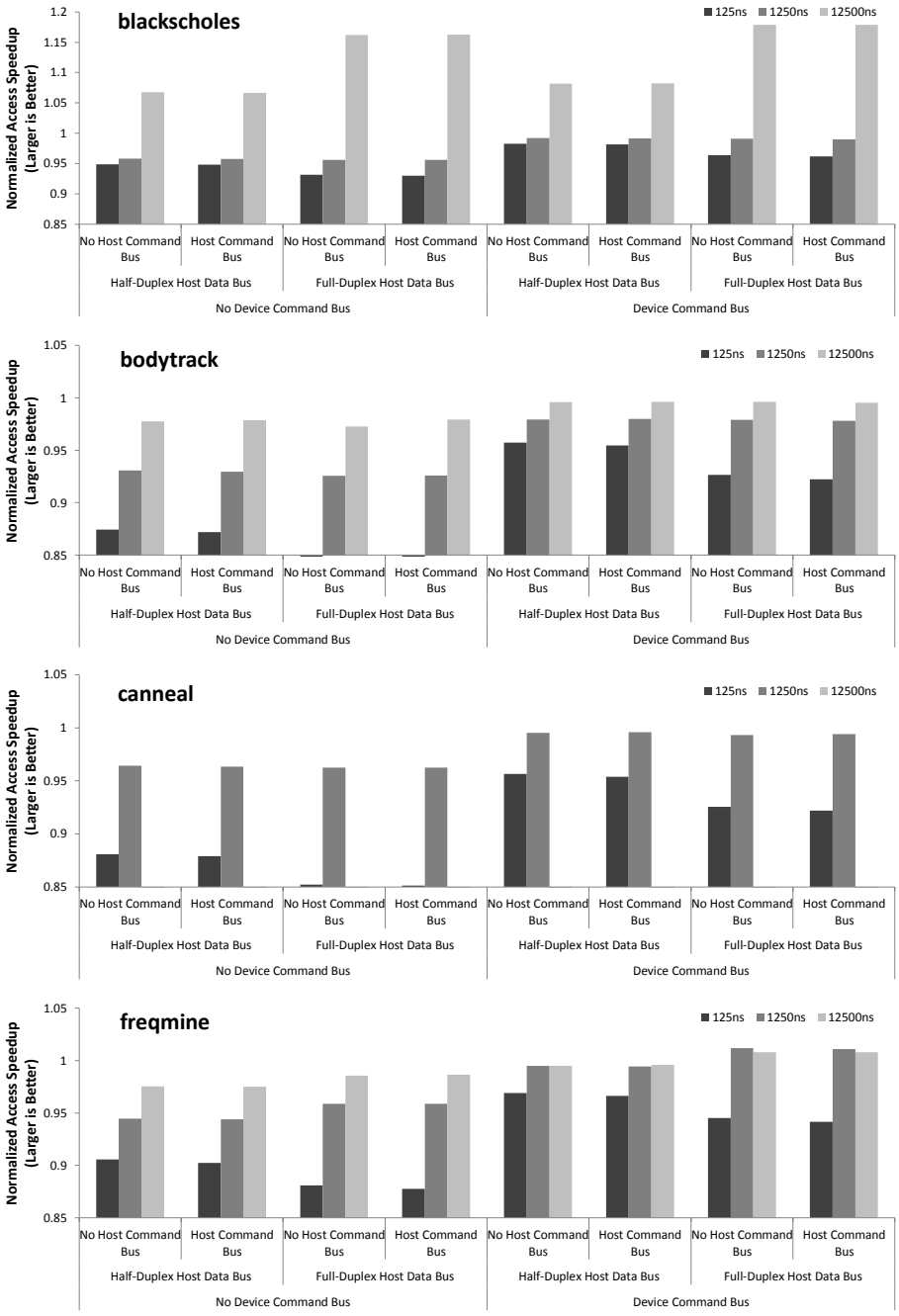

Figure 9.43: The average access latencies for the PARSEC workloads that result from different host and device channel organizations normalized to the average access latency that was observed with all ranks directly connected to the host and prefetching enabled. The host channel in this experiment is operating at a frequency of 400MHz (DDR-800).

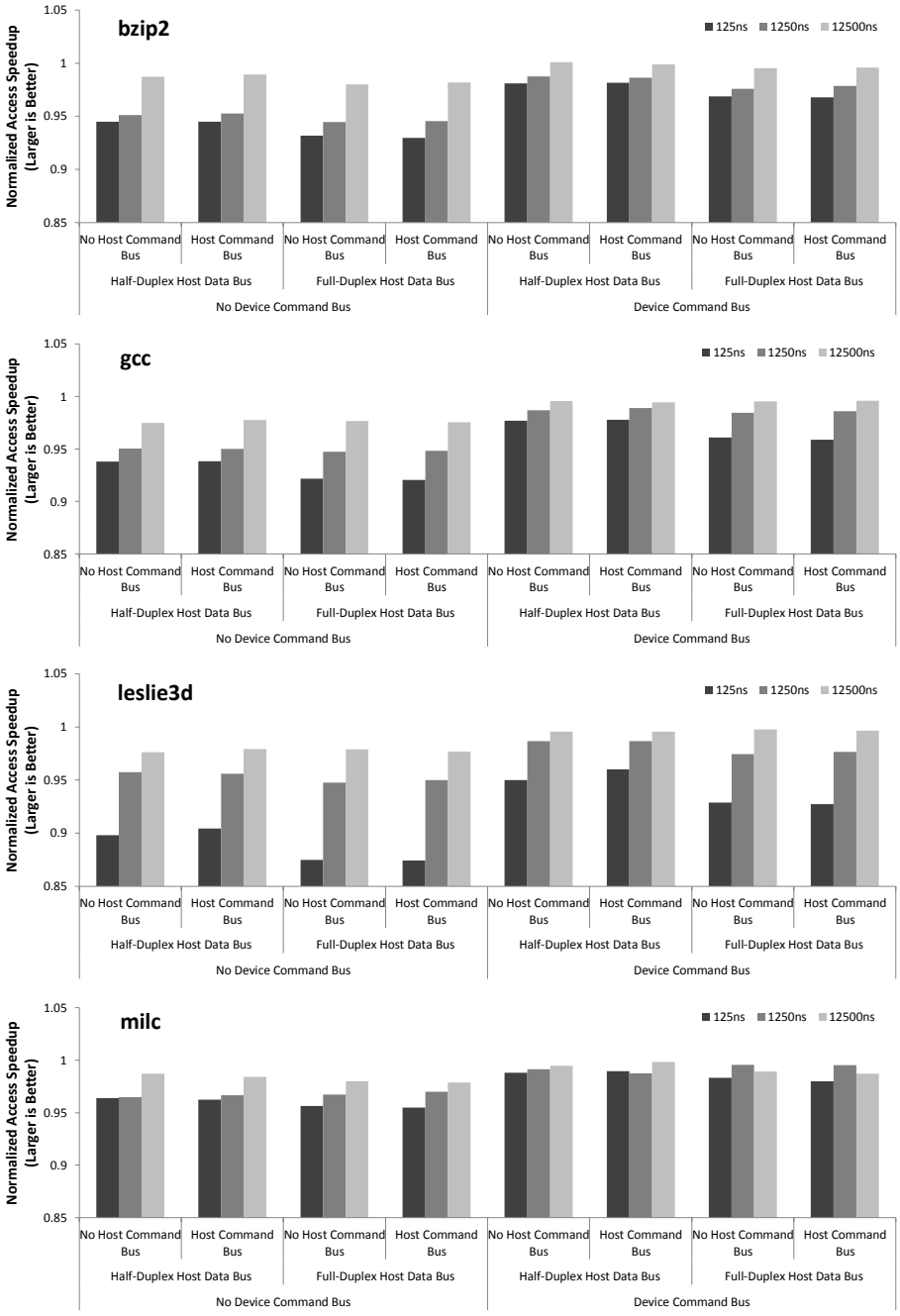

Figure 9.44: The average access latencies for the SPEC workloads that result from different host and device channel organizations normalized to the average access latency that was observed with all ranks directly connected to the host and prefetching enabled. The host channel in this experiment is operating at a frequency of 400MHz (DDR-800).

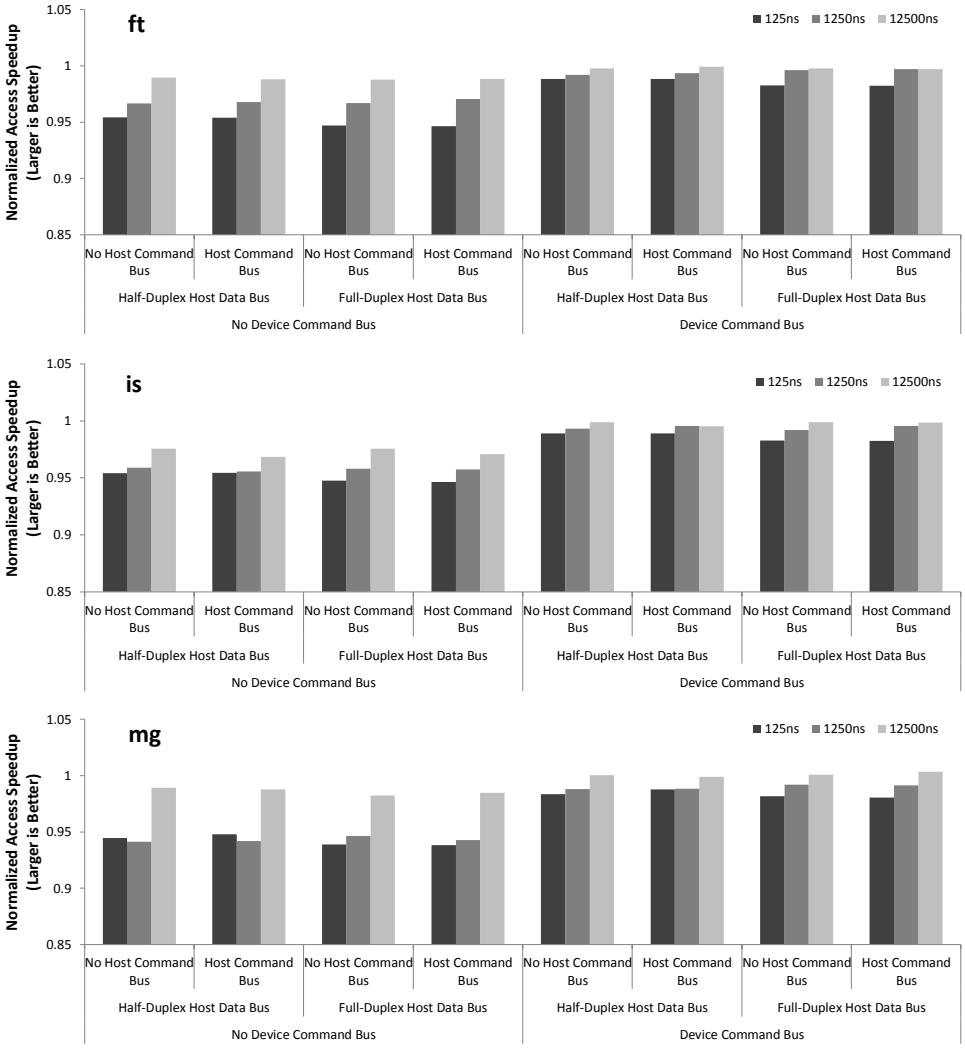

Figure 9.45: The average access latencies for the NPB workloads that result from different host and device channel organizations normalized to the average access latency that was observed with all ranks directly connected to the host and prefetching enabled. The ranks organizations, page sizes, and prefetching degree used in these experiments are the best values for each read latency as observed in the ranks versus page size and prefetching experiments. The host channel in this experiment is operating at a frequency of 800MHz (DDR-1600).

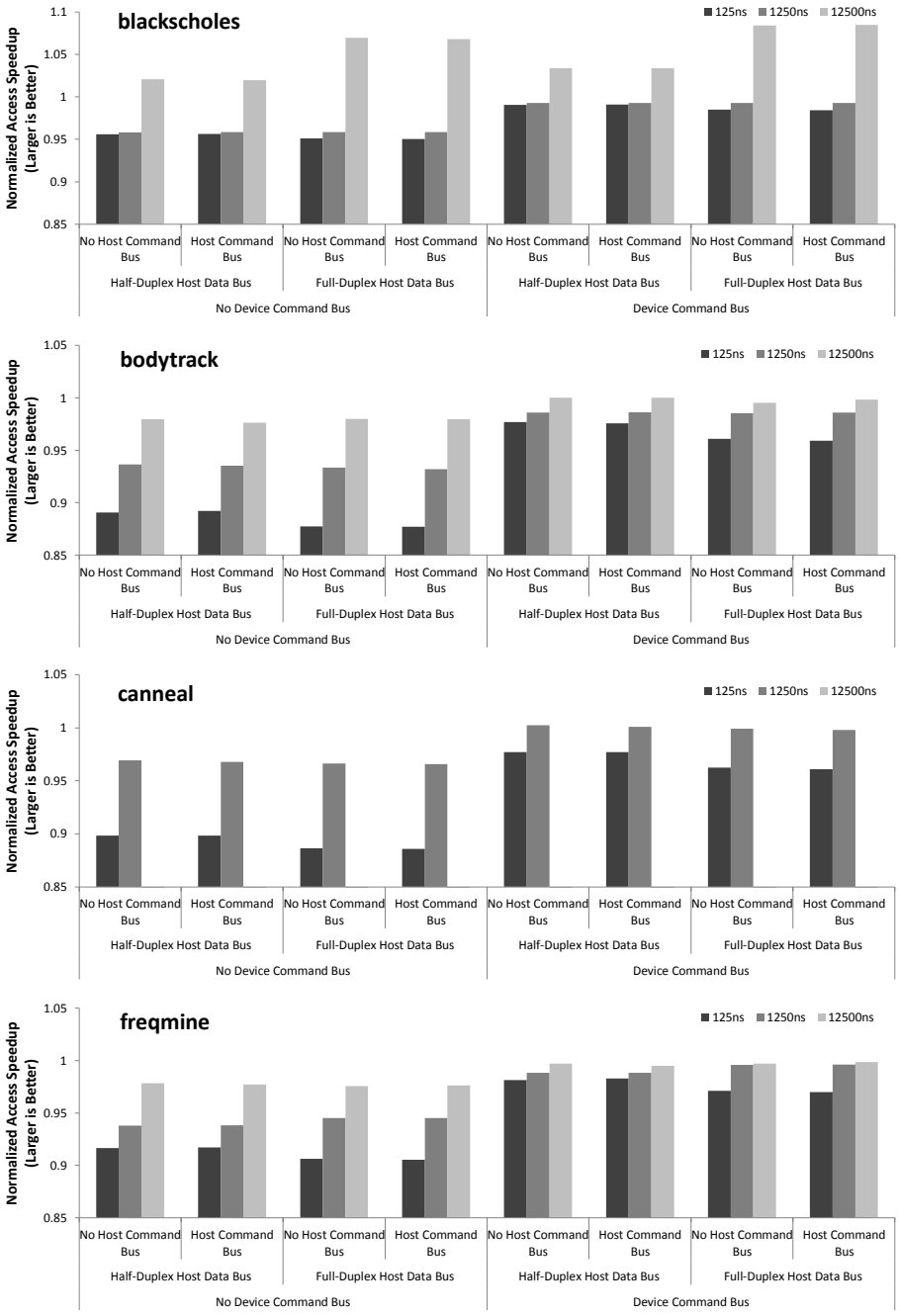

Figure 9.46: The average access latencies for the PARSEC workloads that result from different host and device channel organizations normalized to the average access latency that was observed with all ranks directly connected to the host and prefetching enabled. The host channel in this experiment is operating at a frequency of 800MHz (DDR-1600).

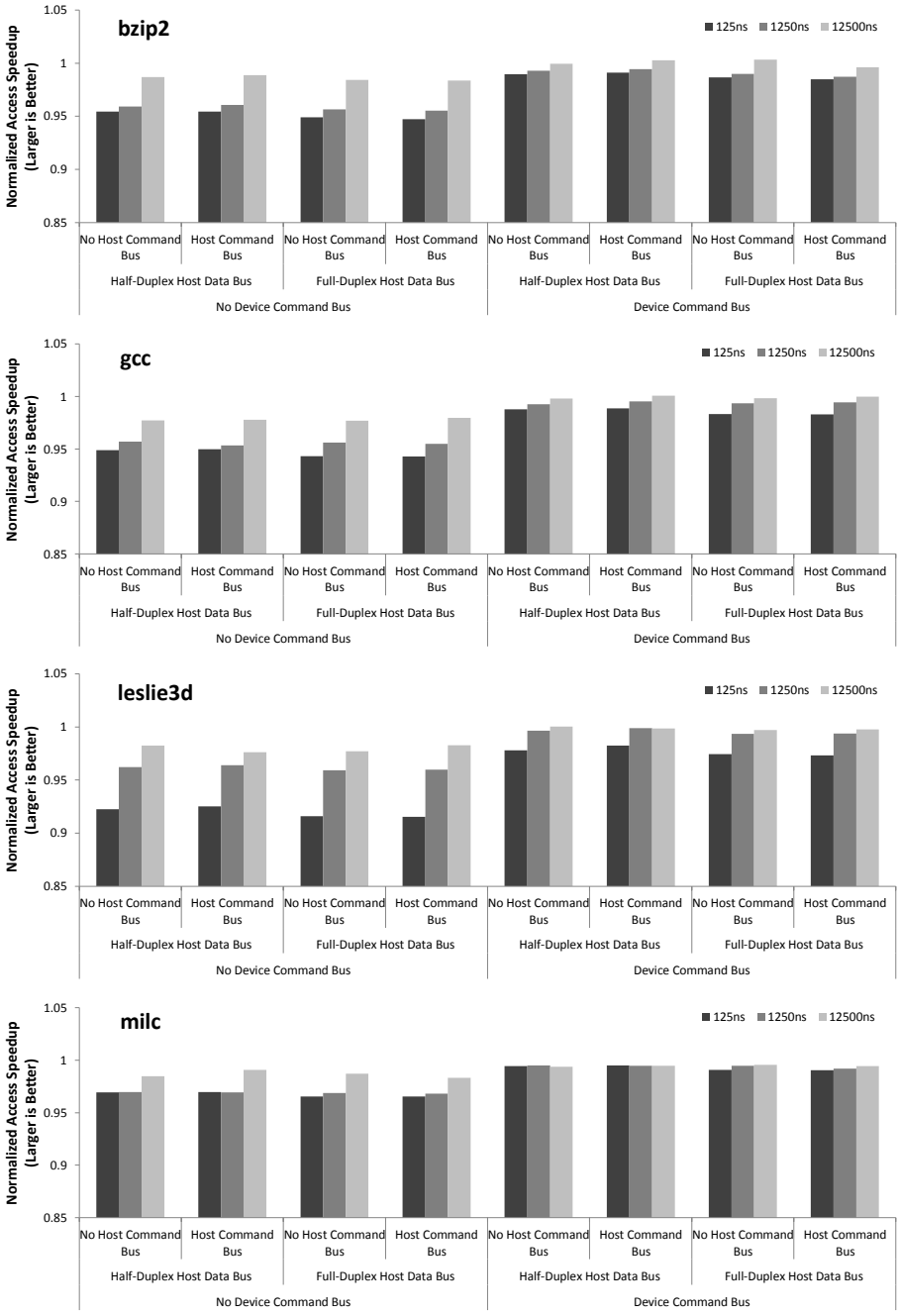

Figure 9.47: The average access latencies for the SPEC workloads that result from different host and device channel organizations normalized to the average access latency that was observed with all ranks directly connected to the host and prefetching enabled. The host channel in this experiment is operating at a frequency of 800MHz (DDR-1600).

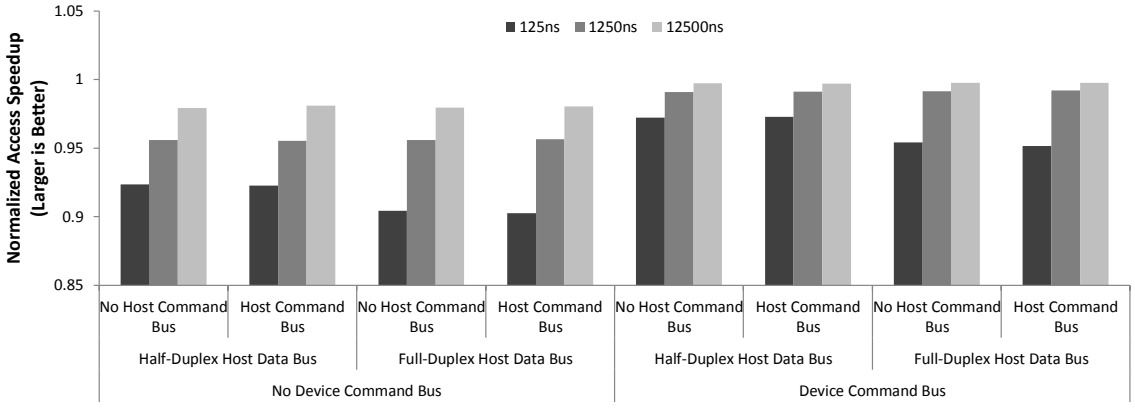

Figure 9.48: The average access latencies averaged across all workloads that result from different host and device channel organizations normalized to the average access latency that was observed with all ranks directly connected to the host and prefetching enabled. The ranks organizations, page sizes, and prefetching degree used in these experiments are the best values for each read latency as observed in the ranks versus page size and prefetching experiments. The host channel in this experiment is operating at a frequency of 400MHz (DDR-800).

is that the full-duplex host channel provides some speedup for the 125ns memory at the DDR1600 frequency. This would seem to indicate that the host channel's bandwidth is generally not a bottleneck in this system. Overall, the addition of the host channel does not greatly reduce the performance of the backing store even when it is a relatively slow DDR-800 channel. Therefore, it should be possible to implement the backing stores of these new multi-level main memories with relatively few CPU pins.

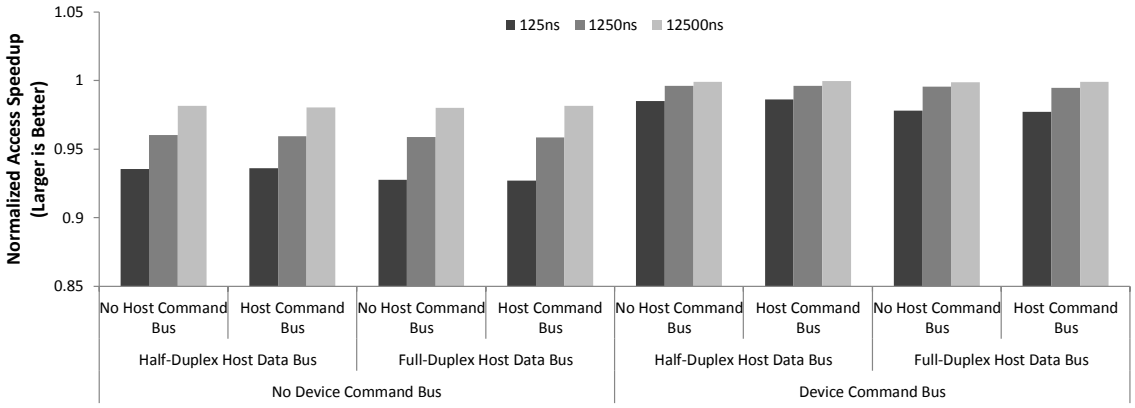

Figure 9.49: The average access latencies averaged across all workloads that result from different host and device channel organizations normalized to the average access latency that was observed with all ranks directly connected to the host and prefetching enabled. The ranks organizations, page sizes, and prefetching degree used in these experiments are the best values for each read latency as observed in the ranks versus page size and prefetching experiments. The host channel in this experiment is operating at a frequency of 800MHz (DDR-1600).

9.5 Summary

In this chapter we have shown that the organization and optimizations used when designing the backing store architecture can greatly impact its overall effectiveness. Selecting the wrong page size or providing too little concurrency can result in massive increases in access latencies. Similarly, prefetching can provide a reasonable performance benefit but prefetching too aggressively can result in a drastic loss in performance. Therefore, while the cache is the primary bottleneck in the multi-level memory system, properly designing the backing store still plays an important

role in reducing the miss penalty of the system.

Chapter 10: Conclusions

The slowdown in DRAM scaling and increase in application working set size has resulted in the development of new multi-level main memory architectures. These architectures leverage the benefits of different technologies and organizations to enable the continued development of larger, faster, more energy effiecient main memory systems. In this dissertation we have explored the expansive design space of this new class of memory architecture in order to better understand the dynamics that exist between its various components. As a result of this work, we have arrived at the following conclusions:

- The cache is one of the most important bottlenecks in the multi-level main memory system because preventing misses is often the key to providing acceptable performance.

- Multi-level main memory architectures can provide a considerable performance boost over existing memory system architectures even when employing backing store technologies that approach SLC NAND flash latencies.

- Associativity can be a critical aspect of DRAM cache design due to the importance of minimizing misses in the multi-level architecture, especially when

backing store latencies are long. However, a careful balance has to be negotiated when building associative DRAM caches so that hit latency is not increased by more than the latency savings introduced by associativity.

- Selecting the correct page size and concurrency for a particular backing store access latency is critical to ensuring adequate overall performance.

- Prefetching can improve the overall system performance by allowing the cache to avoid some misses. However, prefetching too aggressively can consume too much of the available system bandwidth and result in a reduction in system performance rather than an increase.

10.1 Summary of Contributions

This dissertation makes the following noteworthy contributions:

- We present an overview of multi-level main memory system architectures and memory technologies.

- We develop a suite of detailed simulators that enable the investigation of multi-level main memories utilizing generalized memory technologies that currently lack specific access protocols.

- We evaluate the impact of hardware management on the performance of multi-level main memory systems compared to the current software managed SSD architecture.

- We propose a novel associative DRAM cache design that reduces miss rate while not substantially increasing the hit latency or requiring the addition of large amounts of on chip tag storage.

- We quantify the effects of miss rate, miss latency, and cache size on different DRAM cache implementations.

- We demonstrate the effects of page size and concurrency on the performance of backing stores that utilize technologies with different access latencies.

- We show the potential performance advantages of utilizing prefetching to reduce the misses in the DRAM cache.

10.2 Future Work

- *Adaptive Prefetching.* In this work we presented some initial studies on prefetching that used a sequential prefetching algorithm to improve the performance of the system. However, other more complex prefetching algorithms are capable of delivering improved performance by detecting and adjusting to changes in the degree of spatial locality present in the access stream. In particular, the adaptive sequential prefetching, stream buffer prefetching, or sandbox prefetching could all potentially be used to further reduce the number of misses in the DRAM cache [115, 116]. It would be interesting to implement these other prefetching algorithms to see if they could provide significantly better speedups than the improvement achieved by implementing sequential

prefetching.

- *Higher Associativity Combo-Tag Organization.* The current implementation of Combo-Tag is capable of a maximum of 4-way set associativity. Increasing the associativity beyond that point results in very inefficeint DRAM row utilization for a standard 2KB DRAM row buffer. However, increasing the row buffer size to 4KB allows for a 6-way implementation of Combo-Tag that could potentially provide some additional perfromance for some workloads. In addition, increasing the row buffer size to 4KB also improves the efficiency of Combo-Tag in terms of DRAM row utilization. In the future, we would like to investigate the possible benefits of implementing Combo-Tag with a 4KB DRAM row buffer to see if can provide an even greater performance advantage over the completing designs.

- *Variable-Way DRAM Caches.* Many of the workloads that we used to evaluate our DRAM cache design responded to associativity in different ways. Some of them, like milc, seemed to benefit significantly from high degress of associaitivty while others, like the NAS parallel benchmakes, achieved their best performance with just 2-way associativity. This suggests that as the programs running on a system change, the optimal degree of associativity can change with them. Therefore, we feel that it could be valuable to modify Combo-Tag to enable it to change associativity levels dynamically in response to changes in workload miss rates.

Appendix A: Workload Characterization

In this appendix we provide our characterization of the various suites of workloads that we have used in this dissertation. We preformed this characterization in order to determine which benchmarks in each suite were best suited for use in our studies. MARSSx86 was used to run the different benchmarks, the baseline configuration of the simulator can be found in Table A.1. This characterization includes the following statistics as recoded by our HybridSim simulator: observed footprint at the last level cache, the L3 MPKI, the ratio of read accesses to write accesses at the DRAM cache level, and the miss ratio of a 128MB direct mapped DRAM cache. These aspects of the workloads were of particular interest to us because they provide a good indication of how much pressure a particular workload will place on the memory system.

Table A.1: Evaulation Simulator Configuration

Processor	
Number of cores	8-core
Issue Width	4
Frequency	3.2GHz
On Chip Caches	
L1I (private)	128 KB, 8-way, 64 B block size
L1D (private)	128 KB, 8-way, 64 B block size
L2 (private)	256 KB, 8-way, 64 B block size
L3 (shared)	32 MB, 20-way, 64 B block size

Table A.2: Characterization of the NAS Parallel Benchmarks for 10 billion instructions

Benchmark	LLC Footprint (MB)	L3 MPKI	R/W Ratio	DRAM Direct Miss Ratio
bt	173.184082	3.913029	2.880846541	0.408046
cg	160.6359253	19.1633738	87.25277294	0.396512
ft	1287.27301	7.115501	1.109049952	0.550024
is	264.866333	10.424386	3.815548147	0.623369
lu	155.4019775	4.3579122	2.17063395	0.245093
mg	430.8671875	13.5972049	3.010531733	0.604031
sp	190.3614502	10.6178024	2.209388291	0.15951

Table A.3: Characterization of the PARSEC Benchmarks for 50 billion instructions

Benchmark	LLC Footprint (MB)	L3 MPKI	R/W Ratio	DRAM Direct Miss Ratio
blackscholes	269.5270996	0.6400292	7.070654346	0.876011
bodytrack	534.282959	0.4995586	1.866059588	0.521681
canneal	497.6022339	6.50089792	2.10668014	0.35405
dedup	251.3790283	0.20152436	1.736885993	0.531703
facesim	175.843811	4.10726822	7.222585035	0.0541854
ferret	92.02905273	0.87002412	9.661749326	0.05025
fluidanimate	116.5549927	1.1199417	2.019722022	0.123372
freqmine	894.0386963	1.31842398	2.795954077	0.467288
raytrace	206.8879395	0.51020034	16.78099282	0.64883
vips	168.6609497	0.13417034	2.15690758	0.435417

Table A.4: Characterization of a selection of benchmarks from the SPEC CPU2006 suite for 5 billion instructions

Benchmark	LLC Footprint (MB)	L3 MPKI	R/W Ratio	DRAM Direct Miss Ratio
bzip2	2559.927063	61.0071412	1.208938092	0.578204
gcc	441.3080444	6.0996348	3.680787119	0.527504
leslie3d	601.1542358	18.724911	3.07179435	0.593654
mcf	181.4937134	67.6635826	3.939492989	0.264314
milc	1909.656738	22.4704558	1.816159333	0.66328
soplex	130.819458	1.176362	1.588369125	0.386595
wrf	458.9356079	3.3217486	1.692260422	0.605523

Appendix B: Open Memory Simulator Verification

This appendix contains the average latency values for a range of SPEC benchmarks that were produced by the Open Memory Simulator and DRAMSim2. These values were gathered in order to establish that OMS generated latency values that were reasonably similar to the hardware verified DRAMSim2. The tests used a similar configuration for both OMS and DRAMSim2 with 1 channel, 1 rank, and 8 banks. This organization was selected because it was simple yet still tested the similarity of the scheduling algorithms in both simulators. In addition, the DRAMSim2 simulator did not use tFAW or tRRD timings in this test as those limitations are not currently implemented in OMS. The results of the test show that OMS is capable of generating average access latency values that are within 7% of DRAMSim2's values on average and no more than 10% in the worst case.

Table B.1: A comparison of average access latencies for the SPEC traces used in this dissertation measured using DRAMSim and OMS on a system with the same 1Channel - 1Rank - 8Bank configuration in both cases

	DRAMSim	NVDIMM	Difference	Absolute Value
bzip2	280.63	289.63	0.03156455	0.03156455
gcc	177.25	192.343	0.081673625	0.081673625
leslie3d	68.768	62.619	-0.093601346	0.093601346
mcf	391.969	406.794	0.037119897	0.037119897
milc	71.02	64.45	-0.096995645	0.096995645
soplex	67.675	60.746	-0.1079107	0.1079107
wrf	73.685	68.1156	-0.078552559	0.078552559
		Average	-0.032386026	
		Abs Average	0.075345474	

Bibliography

[1] R. J. Baker, *CMOS Circuit Design, Layout, and Simulation*, 3rd ed. Wiley-IEEE Press, 2010.

[2] "Micron technologies, inc." http://www.micron.com, 2015.

[3] "Samsung first to begin shipping 40nm-class, 32-gigabyte memory module for server applications," http://www.samsung.com/global/business/semiconductor/newsView.do?news_id=1139, *Samsung*, Mar 2010.

[4] "Micron introduces industry's highest density ddr3 components and modules," http://news.micron.com/releasedetail.cfm?ReleaseID=440712, *Micron*, Oct 2007.

[5] "Hynix introduces dram industry's first jedec standard 8gb ddr2 r-dimms," http://www.thefreelibrary.com/HynixIntroducesDRAMIndustry'sFirstJEDECStandard8GBDDR2...-a0138702474, *The Free Library*, Nov 2005.

[6] "Micron touts first 4gb ddr dimm," http://www.theregister.co.uk/2003/03/20/micron_touts_first_4gb_ddr/, *The Register*, Mar 2003.

[7] "Samsung announces industry's first 2gb ddr dimm modules; new modules power industry standard hp proliant servers." http://www.thefreelibrary.com/SamsungAnnouncesIndustry'sFirst2GBDDRDIMMModules;NewModules...-a092809153, *The Free Library*, Sept 2002.

[8] "Micron technology, inc., announces industry samples of 2.5v 266mhz 256 meg ddr sdrams," http://www.thefreelibrary.com/MicronTechnology,Inc.,AnnouncesIndustrySamplesof2.5V266MHz...-a067130444, *The Free Library*, Nov 2000.

[9] "Micron announces industry's first monolithic 8gb ddr3 sdram," http://investors.micron.com/releasedetail.cfm?releaseid=859298, *Micron*, July 2014.

[10] "Samsung makes highest density dram chip," http://techcrunch.com/2009/01/29/samsung-makes-highest-density-dram-chip/, *Samsung*, Jan 2009.

[11] "Samsung shows off 2gb ddr2 drams." http://www.xbitlabs.com/news/memory/display/20040920150146.html, *Samsung*, Sept 2004.

[12] "About us: History," http://www.samsung.com/global/business/semiconductor/aboutus/AboutUs_History.html, *Samsung*, 2011.

[13] P. Clarke, "Toshiba rolls 24-nm nand flash," http://www.eetimes.com/electronics-news/4207194/Toshiba-rolls-24-nm-NAND-flash, *EE Times*, Aug 2010.

[14] "Toshiba to launch 43nm slc nand flash memory," http://www.reuters.com/article/2008/10/28/idUS32460+28-Oct-2008+PRN20081028, *Toshiba*, Oct 2008.

[15] "Micron offers industry's first high speed nand product," http://www.eetimes.com/electronics-news/4207194/Toshiba-rolls-24-nm-NAND-flash, *Micron*, Feb 2008.

[16] "Samsung produces first 4-gigabit nand flash memory using 70-nanometer technology; nand flash-dedicated 300mm line delivers first wafers." http://www.thefreelibrary.com/SamsungProducesFirst4-GigabitNANDFlashMemoryUsing70-nanometer...-a0132828739, *The Free Library*, May 2005.

[17] "Micron ships first production 2 gigabit 90 nanometer nand flash memory products today," http://www.thefreelibrary.com/MicronShipsFirstProduction2Gigabit90NanometerNANDFlash...-a0126307978, *The Free Library*, Dec 2004.

[18] "Samsung mass-produces industry's first 1gb nand flash memory device utilizing 0.12-micron process technology," http://www.thefreelibrary.com/SamsungMass-ProducesIndustry'sFirst1GbNANDFlashMemoryDevice...-a090351934, *The Free Library*, Aug 2002.

[19] "Toshiba introduces new high-density nand flash memory products," http://www.thefreelibrary.com/ToshibaIntroducesNewHigh-DensityNANDFlashMemoryProducts.-a065469073, *The Free Library*, Sept 2000.

[20] "Samsung ships world's first 128mb flash memory chips – company gets head start on next-generation flash memory market – stores up to 120 vga-quality pictures," http://www.thefreelibrary.com/SamsungShipsWorld'sFirst128MbFlashMemoryChips--Companygets...-a020460773, *The Free Library*, Sept 2002.

[21] *Fusion-IO*, "iodrive octal angle shot," http://www.fusionio.com/photo/iodrive-octal-angle, 2015.

[22] "iodrive octal flat shot," http://www.fusionio.com/photo/iodrive-octal-flat, *Fusion-IO*, 2015.

[23] B. Kiyoo Itoh, "The history of dram circuit designs; at the forefront of dram development," *Solid-State Circuits Society Newsletter, IEEE*, vol. 13, no. 1, pp. 27–31, Winter 2008.

[24] J. Gantz and D. Reinsel, "The digital universe decade-are you ready," *External Publication of IDC (Analyse the Future) Information and Data*, pp. 1–16, 2010.

[25] M. K. Qureshi, V. Srinivasan, and J. A. Rivers, "Scalable High Performance Main Memory System Using Phase-Change Memory Technology," in *Proceedings of the 36th Annual International Symposium on Computer Architecture*, ser. ISCA '09. New York, NY, USA: ACM, 2009, pp. 24–33.

[26] A. P. Ferreira, B. Childers, R. Melhem, D. Mosse, and M. Yousif, "Using PCM in Next-generation Embedded Space Applications," *Real-Time and Embedded Technology and Applications Symposium, IEEE*, vol. 0, pp. 153–162, 2010.

[27] N. Chatterjee, M. Shevgoor, R. Balasubramonian, A. Davis, Z. Fang, R. Illikkal, and R. Iyer, "Leveraging heterogeneity in dram main memories to accelerate critical word access," in *Microarchitecture (MICRO), 2012 45th Annual IEEE/ACM International Symposium on*, Dec 2012, pp. 13–24.

[28] B. C. Lee, E. Ipek, O. Mutlu, and D. Burger, "Architecting Phase Change Memory as a Scalable Dram Alternative," in *Proceedings of the 36th Annual International Symposium on Computer Architecture*, ser. ISCA '09. New York, NY, USA: ACM, 2009, pp. 2–13.

[29] P. Zhou, B. Zhao, J. Yang, and Y. Zhang, "A durable and energy efficient main memory using phase change memory technology," in *Proceedings of the 36th annual international symposium on Computer architecture*, ser. ISCA '09. New York, NY, USA: ACM, 2009, pp. 14–23.

[30] R. C. Johnson, "Will memristors prove irresistible," *EE Times*, Aug 2008. [Online]. Available: http://www.eetimes.com/electronics-products/analog-products/4106472/Will-memristors-prove-irresistible-

[31] B. Jacob, "The memory system: you can't avoid it, you can't ignore it, you can't fake it," *Synthesis Lectures on Computer Architecture*, vol. 4, no. 1, pp. 1–77, 2009.

[32] E. Kultursay, M. Kandemir, A. Sivasubramaniam, and O. Mutlu, "Evaluating stt-ram as an energy-efficient main memory alternative," in *Performance*

Analysis of Systems and Software (ISPASS), 2013 IEEE International Symposium on, April 2013, pp. 256–267.

[33] J. Coburn, A. M. Caulfield, A. Akel, L. M. Grupp, R. K. Gupta, R. Jhala, and S. Swanson, "Nv-heaps: making persistent objects fast and safe with next-generation, non-volatile memories," *SIGARCH Comput. Archit. News*, vol. 39, no. 1, pp. 105–118, Mar. 2011. [Online]. Available: http://doi.acm.org/10.1145/1961295.1950380

[34] A. Badam and V. S. Pai, "SSDAlloc: Hybrid SSD/RAM Memory Management Made Easy," in *In Proc. 8th USENIX Symposium on Networked Systems Design and Implementation (NSDI '11)*, 2011.

[35] J. C. Mogul, E. Argollo, M. Shah, and P. Faraboschi, "Operating System Support for NVM+DRAM Hybrid Main Memory," in *Proceedings of the 12th conference on Hot topics in operating systems*, ser. HotOS'09. Berkeley, CA, USA: USENIX Association, 2009, pp. 14–14. [Online]. Available: http://dl.acm.org/citation.cfm?id=1855568.1855582

[36] "Hybrid Memory Cube Consortium," http://hybridmemorycube.org.

[37] B. Ganesh, A. Jaleel, D. Wang, and B. Jacob, "Fully-buffered dimm memory architectures: Understanding mechanisms, overheads and scaling," in *High Performance Computer Architecture, 2007. HPCA 2007. IEEE 13th International Symposium on*, Feb 2007, pp. 109–120.

[38] E. Cooper-Balis, P. Rosenfeld, and B. Jacob, "Buffer On Board memory systems," in *Proceedings of the 39th Annual International Symposium on Computer Architecture*, ser. ISCA '12, 2012.

[39] "Open NAND Flash Interface Specification," http://onfi.org/specifications, May 2014.

[40] H. Zheng, J. Lin, Z. Zhang, and Z. Zhu, "Decoupled dimm: Building high-bandwidth memory system using low-speed dram devices," in *Proceedings of the 36th Annual International Symposium on Computer Architecture*, ser. ISCA '09. New York, NY, USA: ACM, 2009, pp. 255–266. [Online]. Available: http://doi.acm.org/10.1145/1555754.1555788

[41] H. Zheng, J. Lin, Z. Zhang, E. Gorbatov, H. David, and Z. Zhu, "Mini-rank: Adaptive dram architecture for improving memory power efficiency," in *Microarchitecture, 2008. MICRO-41. 2008 41st IEEE/ACM International Symposium on*, Nov 2008, pp. 210–221.

[42] K. Fang, H. Zheng, and Z. Zhu, "Heterogeneous mini-rank: Adaptive, power-efficient memory architecture," in *Parallel Processing (ICPP), 2010 39th International Conference on*, Sept 2010, pp. 21–29.

[43] B. Jacob, S. Ng, and D. Wang, *Memory Systems: Cache, DRAM, Disk*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2007.

[44] "Ddr3 sdram mt41jxx datasheet," *Micron*, 2006.

[45] "Samsung k4b2g0446c datasheet," http://www.samsung.com/global/business/semiconductor/products/dram/DDR3/downloads/ds_k4b2gxx46c_1_35v_rev11.pdf, *Samsung*, 2010.

[46] "Process integration, devices & structures," http://www.itrs.net/Links/2009ITRS/2009Chapters_2009Tables/2009_ExecSum.pdf, *International Technology Roadmap for Semiconductors*, 2009.

[47] K. Kim and G. Jeong, "Memory technologies for sub-40nm node," in *Electron Devices Meeting, 2007. IEDM 2007. IEEE International*, dec. 2007, pp. 27 –30.

[48] K. T. Malladi, B. C. Lee, F. A. Nothaft, C. Kozyrakis, K. Periyathambi, and M. Horowitz, "Towards energy-proportional datacenter memory with mobile dram," in *Proceedings of the 39th Annual International Symposium on Computer Architecture*, ser. ISCA '12. Washington, DC, USA: IEEE Computer Society, 2012, pp. 37–48. [Online]. Available: http://dl.acm.org/citation.cfm?id=2337159.2337164

[49] R. Micheloni, L. Crippa, and A. Marelli, *Inside NAND Flash Memories*. Springer, 2010.

[50] P. Pavan, R. Bez, P. Olivo, and E. Zanoni, "Flash memory cells-an overview," *Proceedings of the IEEE*, vol. 85, no. 8, pp. 1248 –1271, aug 1997.

[51] "Tn-29-42: Wear-leveling techniques in nand flash devices," https://www.micron.com/~/media/documents/products/technical-note/nand-flash/tn2942_nand_wear_leveling.pdf, *Mircon*, 2008.

[52] "Tn-29-19: Nand flash 101: An introduction to nand flash and how to design it in to your next product," https://www.micron.com/~/media/documents/products/technical-note/nand-flash/tn2914.pdf, *Micron*, 2006.

[53] J. Brewer and M. Gill, *Nonvolatile Memory Technologies with Emphasis on Flash: A Comprehensive Guide to Understanding and Using Flash Memory Devices*. Wiley-IEEE Press, 2008.

[54] M. K. Qureshi, M. M. Franceschini, L. A. Lastras-Montaño, and J. P. Karidis, "Morphable memory system: a robust architecture for exploiting multi-level phase change memories," in *Proceedings of the 37th annual international symposium on Computer architecture*, ser. ISCA '10. New York, NY, USA: ACM, 2010, pp. 153–162.

[55] B. Lee, P. Zhou, J. Yang, Y. Zhang, B. Zhao, E. Ipek, O. Mutlu, and D. Burger, "Phase-change technology and the future of main memory," *Micro, IEEE*, vol. 30, no. 1, p. 143, jan.-feb. 2010.

[56] S. Mittal, "A survey of power management techniques for phase change memory," *Memory*, vol. 1223, pp. 41–1, 2015.

[57] J. Li, B. Luan, and C. Lam, "Resistance drift in phase change memory," in *Reliability Physics Symposium (IRPS), 2012 IEEE International*, April 2012, pp. 6C.1.1–6C.1.6.

[58] L. Chua, "Memristor-the missing circuit element," *Circuit Theory, IEEE Transactions on*, vol. 18, no. 5, pp. 507 – 519, sep 1971.

[59] S. H. Jo, K.-H. Kim, and W. Lu, "High-density crossbar arrays based on a si memristive system," *Nano Letters*, vol. 9, no. 2, pp. 870–874, 2009.

[60] D. Niu, Y. Chen, and Y. Xie, "Low-power dual-element memristor based memory design," in *Proceedings of the 16th ACM/IEEE international symposium on Low power electronics and design*, ser. ISLPED '10. New York, NY, USA: ACM, 2010, pp. 25–30.

[61] R. Williams, "How we found the missing memristor," *Spectrum, IEEE*, vol. 45, no. 12, pp. 28 –35, dec. 2008.

[62] P. O. Vontobel, W. Robinett, P. J. Kuekes, D. R. Stewart, J. Straznicky, and R. S. Williams, "Writing to and reading from a nano-scale crossbar memory based on memristors," *Nanotechnology*, vol. 20, no. 42, p. 425204, 2009.

[63] Y. Cassuto, s. kvatinsky, and E. Yaakobi, "Sneak-path constraints in memristor crossbar arrays," in *Information Theory Proceedings (ISIT), 2013 IEEE International Symposium on*, July 2013, pp. 156–160.

[64] "Ddrx standards," http://www.jedec.org/, *JEDEC*.

[65] "512mb c-die nor flash," *Samsung*, 2010.

[66] "Numonyx strataflash cellular memory 18-90nm/65nm," *Numonyx*, 2008.

[67] K.-J. Lee, B.-H. Cho, W.-Y. Cho, S. Kang, B.-G. Choi, H.-R. Oh, C.-S. Lee, H.-J. Kim, J.-M. Park, Q. Wang, M.-H. Park, Y.-H. Ro, J.-Y. Choi, K.-S. Kim, Y.-R. Kim, I.-C. Shin, K.-W. Lim, H.-K. Cho, C.-H. Choi, W.-R. Chung, D.-E. Kim, Y.-J. Yoon, K.-S. Yu, G.-T. Jeong, H.-S. Jeong, C.-K. Kwak, C.-H. Kim, and K. Kim, "A 90 nm 1.8 v 512 mb diode-switch pram with 266 mb/s read throughput," *Solid-State Circuits, IEEE Journal of*, vol. 43, no. 1, pp. 150 –162, jan. 2008.

[68] "Omneo p8p pcm 128-mbit parallel phase change memory datasheet," *Numonyx*, 2010.

[69] D. Lewis and H.-H. Lee, "Architectural evaluation of 3d stacked rram caches," in *3D System Integration, 2009. 3DIC 2009. IEEE International Conference on*, sept. 2009, pp. 1 –4.

[70] K. Eshraghian, K. R. Cho, O. Kavehei, S.-K. Kang, D. Abbott, and S.-M. S. Kang, "Memristor mos content addressable memory (mcam): Hybrid architecture for future high performance search engines," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 19, no. 8, pp. 1407 –1417, aug. 2011.

[71] Y. Chen, C. Rettner, S. Raoux, G. Burr, S. Chen, R. Shelby, M. Salinga, W. Risk, T. Happ, G. McClelland, M. Breitwisch, A. Schrott, J. Philipp, M. Lee, R. Cheek, T. Nirschl, M. Lamorey, C. Chen, E. Joseph, S. Zaidi, B. Yee, H. Lung, R. Bergmann, and C. Lam, "Ultra-thin phase-change bridge memory device using gesb," in *Electron Devices Meeting, 2006. IEDM '06. International*, dec. 2006, pp. 1 –4.

[72] G. Servalli, "A 45nm generation phase change memory technology," in *Electron Devices Meeting (IEDM), 2009 IEEE International*, dec. 2009, pp. 1 –4.

[73] "Nand flash memory mt29f4g08aaa, mt29f8g08baa, mt29f8g08daa, mt29f16g08faa datasheet," 2006.

[74] "Samsung nand flash memory k9kag08u1m, k9f8g08u0m, k9f8g08b0m datasheet," http://www.samsung.com/global/system/business/semiconductor/product/2007/6/11/NANDFlash/SLC_LargeBlock/8Gbit/K9F8G08U0M/ds_k9f8g08x0m_rev10.pdf, 2007.

[75] "F26 32gb mlc nand flash memory h27ubg8t2btr-bc h27ucg8u5btr-bc datasheet," *Hynix*, 2011.

[76] L. M. Grupp, J. D. Davis, and S. Swanson, "The bleak future of nand flash memory," in *Proceedings of the 10th USENIX conference on File and Storage Technologies*. USENIX Association, 2012, pp. 2–2.

[77] M. Wu and W. Zwaenepoel, "envy: A non-volatile, main memory storage system," in *ASPLOS*, 1994, pp. 86–97.

[78] C. Li, C. Ding, and K. Shen, "Quantifying the cost of context switch," in *Proceedings of the 2007 workshop on Experimental computer science*, ser. ExpCS '07. New York, NY, USA: ACM, 2007. [Online]. Available: http://doi.acm.org/10.1145/1281700.1281702

[79] R. Budruk, "Pci express basics," http://www.uio.no/studier/emner/matnat/ifi/INF5063/h14/slides/03_29_pci_express_basics.pdf, *PCI-SIG*, 2007.

[80] "Pci-sig," http://pcisig.com/, 2014.

[81] M. K. Qureshi and G. H. Loh, "Fundamental latency trade-off in architecting dram caches: Outperforming impractical sram-tags with a simple and practical design," in *Proceedings of the 2012 45th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO-45. Washington, DC, USA: IEEE Computer Society, 2012, pp. 235–246. [Online]. Available: http://dx.doi.org/10.1109/MICRO.2012.30

[82] C.-C. Huang and V. Nagarajan, "Atcache: Reducing dram cache latency via a small sram tag cache," in *Proceedings of the 23rd International Conference on Parallel Architectures and Compilation*, ser. PACT '14. New York, NY, USA: ACM, 2014, pp. 51–60. [Online]. Available: http://doi.acm.org/10.1145/2628071.2628089

[83] G. H. Loh and M. D. Hill, "Efficiently enabling conventional block sizes for very large die-stacked dram caches," in *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO-44. New York, NY, USA: ACM, 2011, pp. 454–464. [Online]. Available: http://doi.acm.org/10.1145/2155620.2155673

[84] J. Yang, D. B. Minturn, and F. Hady, "When poll is better than interrupt," in *FAST'12: Proceedings of the 10th USENIX conference on File and Storage Technologies*. Berkeley, CA, USA: USENIX Association, 2012, pp. 3–3.

[85] A. Foong, B. Veal, and F. Hady, "Towards SSD-ready Enterprise Platforms," in *1st International Workshop on Accelerating Data Management Systems Using Modern Processor and Storage Architectures (ADMS)*, 2010.

[86] A. M. Caulfield, A. De, J. Coburn, T. I. Mollow, R. K. Gupta, and S. Swanson, "Moneta: A High-Performance Storage Array Architecture for Next-Generation, Non-volatile Memories," in *Proceedings of the 2010 43rd Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO '43. Washington, DC, USA: IEEE Computer Society, 2010, pp. 385–395. [Online]. Available: http://dx.doi.org/10.1109/MICRO.2010.33

[87] J. Condit, E. B. Nightingale, C. Frost, E. Ipek, B. Lee, D. Burger, and D. Coetzee, "Better i/o through byte-addressable, persistent memory," in *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*, ser. SOSP '09. New York, NY, USA: ACM, 2009, pp. 133–146. [Online]. Available: http://doi.acm.org/10.1145/1629575.1629589

[88] T. Kgil and T. Mudge, "FlashCache: a NAND Flash Memory File Cache for Low Power Web Servers," in *Proceedings of the 2006 International Conference on Compilers, Architecture and Synthesis for Embedded systems*, ser. CASES '06. New York, NY, USA: ACM, 2006, pp. 103–112.

[89] D. Roberts, T. Kgil, and T. Mudge, "Using non-volatile memory to save energy in servers," in *Proceedings of the Conference on Design, Automation*

and Test in Europe, ser. DATE '09. 3001 Leuven, Belgium, Belgium: European Design and Automation Association, 2009, pp. 743–748. [Online]. Available: http://dl.acm.org/citation.cfm?id=1874620.1874804

[90] Oracle, "Achieving new levels of datacenter performance and efficiency with software-optimized flash storage," http://www.oracle.com/us/products/servers-storage/storage/tape-storage/software-optimized-flash-192597.pdf, 2010.

[91] "PCI Express OCZ Technology," http://www.ocztechnology.com/products/solid_state_drives/pci-e_solid_state_drives, 2012.

[92] "Fusion-io," http://www.fusionio.com, 2012.

[93] Spansion, "Using spansion ecoram to improve tco and power consumption in internet data centers," http://www.spansion.com/jp/About/Documents/spansion_ecoram_whitepaper_0608.pdf, 2008.

[94] T. Hardware, "Samsung intros nand flash-friendly file system," http://www.tomshardware.com/news/NAND-Flash-Flash-Friendly-File-System-F2FS-Jaegeuk-Kim,18229.html, 2012.

[95] InsideHPC, "Spansion packs a whole lotta ram into your server," http://insidehpc.com/2009/04/24/spansion-packs-a-whole-lotta-ram-into-your-server/, 2009.

[96] P. Rosenfeld, E. Cooper-Balis, and B. Jacob, "DRAMSim2: A Cycle Accurate Memory System Simulator," *Computer Architecture Letters*, vol. 10, no. 1, pp. 16 –19, Jan.-June 2011.

[97] F. Bellard, "Qemu, a fast and portable dynamic translator." in *USENIX Annual Technical Conference, FREENIX Track*, 2005, pp. 41–46.

[98] C. Dirik and B. Jacob, "The performance of PC Solid-State Disks (SSDs) as a function of bandwidth, concurrency, device architecture, and system organization," in *Proceedings of the 36th Annual International Symposium on Computer Architecture*, ser. ISCA '09, 2009, pp. 279–289.

[99] N. Agrawal, V. Prabhakaran, T. Wobber, J. D. Davis, M. Manasse, , and R. Panigrahy, "Design Tradeoffs for SSD Performance," in *Proceedings of the 2008 USENIX Technical Conference (USENIX'08)*, ser. USENIX '08, 2008.

[100] "Gups," http://www.dgate.org/~brg/files/dis/gups.

[101] "Filebench," http://www.fsl.cs.sunysb.edu/~vass/filebench.

[102] V. Tarasov, S. Bhanage, E. Zadok, and M. Seltzer, "Benchmarking file system benchmarking: It* is* rocket science," *HotOS XIII*, 2011.

[103] J. Edler and M. D. Hill, "Dinero iv trace-driven uniprocessor cache simulator," 1998.

[104] F. Hameed, L. Bauer, and J. Henkel, "Simultaneously optimizing dram cache hit latency and miss rate via novel set mapping policies," in *Compilers, Architecture and Synthesis for Embedded Systems (CASES), 2013 International Conference on*, Sept 2013, pp. 1–10.

[105] J. Meza, J. Chang, H. Yoon, O. Mutlu, and P. Ranganathan, "Enabling efficient and scalable hybrid memories using fine-granularity dram cache management," *Computer Architecture Letters*, vol. 11, no. 2, pp. 61–64, July 2012.

[106] S. Franey and M. Lipasti, "Tag tables," in *High Performance Computer Architecture (HPCA), 2015 IEEE 21st International Symposium on*, Feb 2015, pp. 514–525.

[107] D. Jevdjic, G. Loh, C. Kaynak, and B. Falsafi, "Unison cache: A scalable and effective die-stacked dram cache," in *Microarchitecture (MICRO), 2014 47th Annual IEEE/ACM International Symposium on*, Dec 2014, pp. 25–37.

[108] D. Jevdjic, S. Volos, and B. Falsafi, "Die-stacked dram caches for servers: Hit ratio, latency, or bandwidth? have it all with footprint cache," in *Proceedings of the 40th Annual International Symposium on Computer Architecture*, ser. ISCA '13. New York, NY, USA: ACM, 2013, pp. 404–415. [Online]. Available: http://doi.acm.org/10.1145/2485922.2485957

[109] Y. Lee, J. Kim, H. Jang, H. Yang, J. Kim, J. Jeong, and J. W. Lee, "A fully associative, tagless dram cache," in *Proceedings of the 42Nd Annual International Symposium on Computer Architecture*, ser. ISCA '15. New York, NY, USA: ACM, 2015, pp. 211–222. [Online]. Available: http://doi.acm.org/10.1145/2749469.2750383

[110] A. Patel, F. Afram, S. Chen, and K. Ghose, "MARSSx86: A Full System Simulator for x86 CPUs," in *Design Automation Conference 2011 (DAC'11)*, 2011.

[111] J. L. Henning, "Spec cpu2006 benchmark descriptions," *SIGARCH Comput. Archit. News*, vol. 34, no. 4, pp. 1–17, Sep. 2006. [Online]. Available: http://doi.acm.org/10.1145/1186736.1186737

[112] D. H. Bailey, E. Barszcz, J. T. Barton, D. S. Browning, R. L. Carter, L. Dagum, R. A. Fatoohi, P. O. Frederickson, T. A. Lasinski, R. S. Schreiber, H. Simon, V. Venkatakrishnan, and S. Weeratunga, "The nas parallel benchmarks," *International Journal of High Performance Computing Applications*, vol. 5, no. 3, pp. 63–73, 1991.

[113] C. Bienia, S. Kumar, J. P. Singh, and K. Li, "The parsec benchmark suite: Characterization and architectural implications," in *Proceedings of*

the 17th International Conference on Parallel Architectures and Compilation Techniques, ser. PACT '08. New York, NY, USA: ACM, 2008, pp. 72–81. [Online]. Available: http://doi.acm.org/10.1145/1454115.1454128

[114] "Tb-29-28: Memory management in nand flash arrays," https://www.micron.com/~/media/documents/products/technical-note/nand-flash/tn2928.pdf, *Micron*, 2005.

[115] S. P. Vanderwiel and D. J. Lilja, "Data prefetch mechanisms," *ACM Comput. Surv.*, vol. 32, no. 2, pp. 174–199, Jun. 2000. [Online]. Available: http://doi.acm.org/10.1145/358923.358939

[116] S. Pugsley, Z. Chishti, C. Wilkerson, P. fei Chuang, R. Scott, A. Jaleel, S.-L. Lu, K. Chow, and R. Balasubramonian, "Sandbox prefetching: Safe run-time evaluation of aggressive prefetchers," in *High Performance Computer Architecture (HPCA), 2014 IEEE 20th International Symposium on*, Feb 2014, pp. 626–637.