ENERGY EFFICIENT BUFFER CACHE USING PHASE CHANGE MEMORY

by

Ajeya Naithani

A Thesis Submitted to the Faculty of the

DEPARTMENT OF COMPUTER SCIENCE

In Partial Fulfillment of the Requirements
For the Degree of

MASTER OF SCIENCE

In the Graduate College

THE UNIVERSITY OF ARIZONA

2 0 1 1

STATEMENT BY AUTHOR

This thesis has been submitted in partial fulfillment of requirements for an advanced degree at The University of Arizona and is deposited in the University Library to be made available to borrowers under rules of the library.

Brief quotations from this thesis are allowable without special permission, provided that accurate acknowledgment of source is made. Requests for permission for extended quotation from or reproduction of this manuscript in whole or in part may be granted by the head of the major department or the Dean of the Graduate College when in his or her judgment the proposed use of the material is in the interests of scholarship. In all other instances, however, permission must be obtained from the author.

SIGNED: _____

APPROVAL BY THESIS DIRECTOR

This thesis has been approved on the date shown below:

Dr. Chris Gniady Assistant Professor, Department of Computer Science	Date

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to my advisor, Dr. Chris Gniady. His expertise in Computer Architecture and Operating Systems honed my research skills and prepared me for future challenges. I am also deeply grateful to him for being available for help when needed and for providing extremely useful and constructive feedback.

I am thankful to Dr. John Hartman and Dr. David Lowenthal for serving on my committee and supporting me through the entire process.

I am greatly indebted to Dr. Patrick Homer as I learnt a lot about teaching skills from my interactions with him. His enthusiasm for teaching and the manner of explaining the toughest ideas in the simplest possible way are commendable.

Family and friends have been a consistent source of support and encouragement. Aniruddha Marathe and Tapasya Patki were two great companions in Tucson. I will always remember the time we shared working together on academics, cooking, and the wonderful walks around the university every evening. I am glad to have Johny Rufus John, Young-kyoon Suh and Jaya Wadhwani as my friends and I thank them for helping me in and out of the department.

Finally, I fall short of words to express my gratitude to my father, Dr. L. P. Naithani; my brothers, Vijay Naithani and Sanjeev Naithani; and my sister, Shakuntala Sharma; for everything.

DEDICATION

Dedicated to the memory of my mother, Sarojini Naithani (1953-2004)

TABLE OF CONTENTS

TABLE OF CONTENTS — *Continued*

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

Main memory consumes a significant portion of the overall energy of a modern computer system. A major part of this energy can be attributed to the necessity of keeping DRAM in ready state, even when the memory is rarely accessed. Recently, Phase Change Memory (PCM) has emerged as a competitor to DRAM offering low energy consumption in standby state with reasonable performance. However, the issues of lower performance and high write energy of PCM need to be addressed before we consider PCM as a building block in the memory hierarchy.

In our research, we leverage the advantages of energy efficiency of PCM and low read/write latency of DRAM by designing a hybrid buffer cache architecture using PCM and DRAM. We target commercial file servers, where most of the main memory is dedicated to the buffer cache to improve the file-I/O response time. A dynamic approach to enable or disable DRAM in the proposed hybrid architecture for energy-performance efficiency is presented. We explore several schemes to bring performance close to the DRAM-only system and reduce the main memory energy requirements to 5% of the DRAM-only system. At the same time, our schemes yield up to 78% memory access time improvement over the PCM-only system.

CHAPTER 1

Introduction

Growing demand for web-accessible information has led to the development of large data centers. Increased size of data centers has caused an increase in power consumption, eventually leading to an increase in the overall budget required to run these data centers. A recent study shows that 3% of the total electricity budget of the United States is required to manage these data centers [31]. To reduce this budget, it is important to invent and apply efficient power management mechanisms while maintaining performance.

Component	Energy Consumed
Processor	28%
Memory	41%
Cooling System	28%

Table 1.1: Energy breakdown of a server

Let us consider what constitutes a typical data center. A data center runs a number of commercial file servers, each comprising of four key components– high performance processors, terabytes of memory, secondary storage and the cooling system. Though the portion of the energy consumed by processor may be dominant in a personal computing environment, it is the memory that contributes most to the overall energy requirements of a server [1]. Energy breakdown of a server is given in table 1.1. A small portion of about 3% of the total energy is consumed by other components of the server. Since secondary storage is maintained remotely

by means of a high speed network, it is not included in the table. As shown in the table, memory requires up to 41% of the total energy of a commercial file server.

Since commercial file servers are optimized to improve file-I/O response time and accesses to disk are expensive compared to the memory, most of the memory is dedicated to the buffer cache. Buffer cache in a modern commercial file server can occupy up to 92% of the total size of the memory [2, 3]. Occupying a major portion of the large amounts of memory makes buffer cache the most significant contributor to the total memory energy requirements. Hence, the work in this thesis focuses on memory energy management in general, and buffer cache energy efficiency in particular for commercial file servers running in a data center.

1.1 Problem Description

Double Data Rate Synchronous Dynamic Random Access Memory (DDR SDRAM) and its successors, DDR2 SDRAM and DDR3 SDRAM are used as memory in personal computing systems as well as high-end commercial servers. For the past four decades, DRAM has been the de-facto memory technology in the market due to its high performance characteristics. When data centers were small in size and energy was not a concern, energy characteristics of DRAM were not debated much. However, with increasing size of data centers, energy requirements of data centers has become an issue of prime importance.

The "dynamic" nature of DRAM comes from the fact that data needs to be periodically refreshed even when the memory is idle. There exist various power modes for DRAM and accesses are always performed in the high power mode. If the memory is in a low power mode, it has to be switched to the high power mode to service a request and this introduces a considerable delay. In data centers, the

workload varies with time, and DRAM is always maintained in the high power mode to avoid the switching delay and minimize the file-I/O response time. However, most of the time, the buffer cache is idle since there is no file-I/O. As a significant amount of memory is dedicated to the buffer cache, it has been observed that memory is idle most of the time leading to high memory energy requirements.

High idle energy requirements of DRAM suggest that replacing DRAM with a memory technology that consumes less idle energy can significantly reduce overall memory energy requirements. Non-volatile memory technologies like flash memory and Phase Change Memory (PCM) look promising from this perspective. Flash memory has very high I/O latencies and limited endurance, thus making it a less preferred choice for memory. On the other hand, PCM has emerged as a new memory technology with read performance comparable to DRAM. However, high write energy and high write latency of PCM need to be addressed before it can replace DRAM in the current memory hierarchy.

For the scope of this work, we focus on improving buffer cache energy efficiency of commercial file servers by using PCM, and develop mechanisms to minimize the performance loss caused by the longer I/O latencies of PCM compared to DRAM. The following section describes the contributions and significance of our work.

1.2 Contributions and Significance

Following are the contributions of this thesis:

1. When idle, PCM is more energy efficient than DRAM; however, DRAM has better I/O latencies than PCM. We designed a hybrid buffer cache architecture using both PCM and DRAM to leverage the advantages of energy efficiency

of PCM and better I/O latencies of DRAM.

Initially, we replaced the original DRAM buffer cache with PCM. This change reduced the overall energy significantly but the performance was degraded due to longer I/O latencies of PCM. To minimize the performance degradation, we proposed a hybrid architecture with PCM as buffer cache with a small DRAM-cache to PCM, so that applications with good locality of reference can take advantage of better I/O latencies of DRAM.

2. Applications with a poor locality of reference did not benefit much from the DRAM-cache. The most significant contribution of this work is a dynamic switching mechanism that selects DRAM as a cache to PCM for energy-performance efficiency at runtime. For applications with a good hit ratio in the DRAM-cache, this mechanism selects the route through the DRAM-cache. On the other hand, for applications with a poor hit ratio, this mechanism powers down the DRAM-cache and allows applications to directly access PCM.

3. We noted that with the current performance characteristics of PCM, it is not possible to meet the performance achieved with DRAM-only systems. Even with a very high hit ratio in the DRAM-cache, the best performance we were able to achieve with the hybrid buffer cache architecture was 1% worse than the system that used only DRAM.

1.3 Organization of the Thesis

This thesis starts with some essential background on DRAM and buffer cache, and then provides a detailed examination of Phase Change Memory (PCM). This background is followed by the design and evaluation of the various mechanisms for

our hybrid buffer cache architecture. Chapter 1 gives an overview of the problem of high energy requirements in data centers. Since DRAM is currently used as memory in data centers and buffer cache being an integral part of it, Chapter 2 illustrates the working principles of DRAM and the buffer cache. The structure of a PCM cell and various characteristics of PCM; like scalability, cost, I/O, endurance and energy issues are discussed in Chapter 3. The design and implementation of our mechanisms for improving energy-performance efficiency of buffer cache is provided in Chapter 4. The simulation environment and evaluation of our mechanisms are presented in Chapter 5. Chapter 6 concludes the work and explores possible future directions.

CHAPTER 2

DRAM and the Buffer Cache

2.1 Introduction

Memory hierarchy consists of levels of storage elements with different capacities, access times and costs. The key components of this hierarchy are CPU registers, CPU caches, memory and disk. The fundamental principle behind the establishment of this hierarchy is *locality of reference*. Dynamic Random Access Memory (DRAM) has been used as primary memory for the past four decades. Modern operating systems maintain a portion of DRAM to cache file blocks accessed by an application, so that subsequent accesses to the same block can be satisfied from this in-memory cache. Such an in-memory cache is known as *buffer cache* and is managed by the kernel. Since disks are very slow when compared to DRAM, buffer cache improves the file-I/O response time by reducing the number of accesses to the disk. This chapter focuses on device organization, performance and energy issues related to DRAM and the working principle of the buffer cache. A brief description of CPU caches and disks is also provided.

2.2 CPU Caches

CPU caches (L1, L2 and usually off-chip L3) are the hardware caches that follow the processor registers in the memory hierarchy. Static Random Access Memory or

SRAM is generally used to build these caches as it has lower access latency when compared to other elements of the hierarchy like DRAM or disk. However, cost per bit of SRAM is higher than DRAM or disk and is also four to six times less denser than DRAM [30]. Lower density of SRAM comes from the fact that six transistors are required to permanently hold a single bit of charge on a capacitor to make it static, as opposed to the one transistor used in DRAM [30]. This permanent holding of charge on a capacitor as long as the power is on makes SRAM static. Hence, CPU caches are static and volatile.

2.3 DRAM

Unlike SRAM that is static, DRAM is dynamic; meaning that the charge gradually leaks out of the capacitor and periodic refresh is required to maintain the charge above a distinguishable level. Each storage cell comprises of one transistor and one capacitor. Less number of transistors required for one bit of charge makes DRAM much denser than SRAM for the same area of the chip [30]. DRAM is also volatile since the content of the memory cells is lost once DRAM is powered off. In this work, we consider the system where one memory controller manages the entire DRAM memory system.

2.3.1 DRAM Basics and Organization

A set of Dual Inline Memory Modules (DIMM) constitute the DRAM memory system. *Memory modules* hold a group of DRAM devices and can be inserted or removed from the system board. Their purpose is to support upgradability of the memory system. A DRAM chip or *device* is an array of storage bits; and the performance of the memory system depends on the organization of devices in the

DRAM memory system. A typical DRAM memory system is organized as channels, ranks, banks, rows and columns [29]. This DRAM terminology is briefly described below.

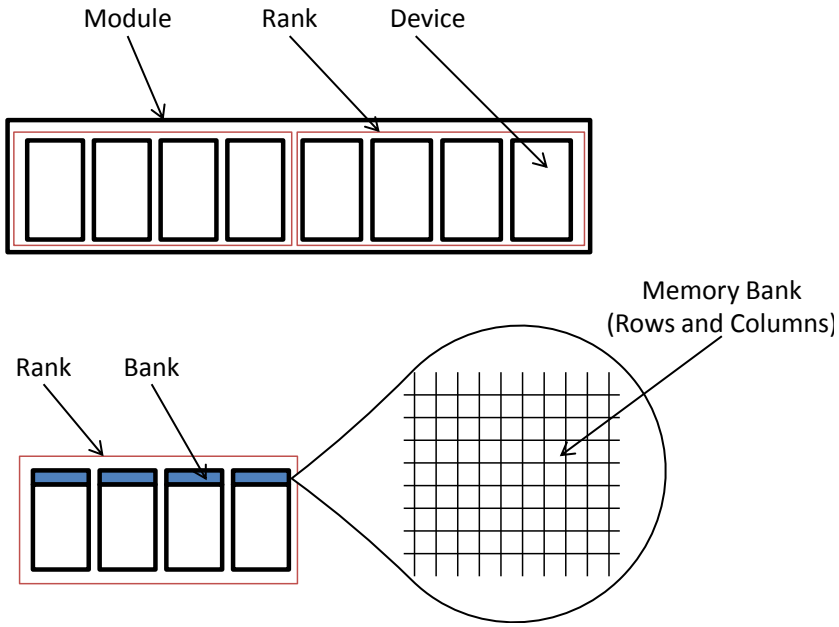

Figure 2.1: DRAM organization as module, rank, bank, rows and columns

The width of the data bus (usually 64 bits) defines how much data can be transferred at a time through physical *channel* between the memory controller and DRAM memory system. When one memory controller manages multiple channels, address mapping scheme has to select only one channel at a time. To transfer data in parallel through two channels, two memory controllers are required, each managing one channel. A *rank* is a set of DRAM devices that operate in a lockstep fashion. Exactly one rank is activated for each read or write operation, and all devices of a rank are activated or deactivated at the same time. A *bank* is a logical array of bits spanning all the devices of a rank. Each bank in a rank is associated with an array of sense amplifiers or *row buffer*. On application of a *row access command* or *Row Address Strobe (RAS)*, row buffer temporarily stores one row of data accessed from

the currently opened bank. There are multiple banks inside a rank and more than one bank can be accessed at the same time. However, all banks have to share the same data bus, and at a time, data from only one bank can be transferred through channel between the memory controller and the DRAM memory system.

A *row* spans a bank within a rank. All the memory cells corresponding to a row are activated in parallel. On receiving a RAS, data equal to the size of one row is moved from the row of memory cells to the row buffer. A *column* is equal to the size of the data bus width, and at a time, *column access command* or *Column Address Strobe (CAS)* transfers only one column size of data between the memory controller and DRAM. A DRAM module with two ranks and four devices per rank is shown in the figure 2.1.

2.3.2 DRAM I/O Operations

The communication between the memory controller and DRAM memory system is performed primarily using four commands– RAS, CAS, *precharge* and *refresh*. The first command sent by memory controller to the DRAM memory system is a RAS that moves one page size of data from a memory bank to the row buffer. Reads and writes are performed to the row buffer. Memory controller performs CAS to read one column size of data from the row buffer. For a write operation, CAS writes one column size of data to the row buffer. Data is updated in the row buffer first and then the updated row buffer is written to the original memory bank. Once a read or a write operation is complete, memory controller send a precharge command to reset the row buffers for the next row access command.

The charge stored in the capacitors of DRAM gradually leaks out and it has to be periodically restored to the full value. Periodic refresh commands sent

by memory controller read data from memory banks to the row buffer, restore it to the full values at the row buffer, and then finally write the refreshed row buffer back to the original memory bank. The gradual leakage of charge is the reason for the *dynamic* nature of the DRAM.

2.3.3 DRAM Memory Controller

The memory controller or DRAM controller is responsible for managing flow of data in and out of the DRAM memory system. Apart from performing scheduling, buffering and pipelining of read and write requests, it performs *address mapping*. Address mapping first breaks the given request into channel number, rank number, bank number, row number and column number, and then the corresponding commands are sent to the DRAM memory system. Memory controller schedules and maps requests to minimize the bank conflict and access latency. Since opening and closing a row buffer takes time, one approach to minimize the access latency is the *row buffer management policy* of the memory controller. *Open page* and *closed page* are the two types of row buffer management policies. In an open page policy, once a request has been processed, memory controller leaves a row buffer open for the subsequent requests. On the other hand, in a closed page policy, a row buffer is closed immediately after a request has been processed. Open page policy favors the applications with significant level of locality while the closed page policy is good for the applications accessing different banks within the DRAM memory system.

2.3.4 DRAM Energy Management

A rank is the basic unit of power management in DRAM. All devices in a rank are activated and deactivated at the same time. *Precharge*, *self-refresh* and *power-down*

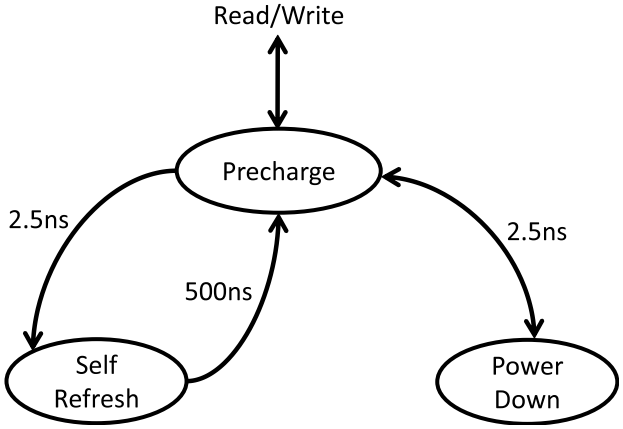

Figure 2.2: Various power states of a DRAM rank

are the three power states in which a rank can operate. Reads and writes can be performed only in the precharge state [2, 4]. Also, as shown in table 2.1, power consumption in the precharge state is significantly higher than other states. When memory is idle, memory controller can switch a rank from one state to another. Figure 2.2 shows the power states and switching delay of a rank. If a rank is in power-down or self-refresh state and a request needs access to the rank, it has to be switched to the precharge state. The application may experience a delay before the actual memory request is processed; hence, DRAM in commercial servers is always maintained in the precharge state to minimize the time to service a request. For work in this thesis also, we assume that a rank is always in the precharge state.

State	Power Consumed (mW)
Precharge	1877.1
Power-down	963.5
Self-refresh	80.075

Table 2.1: Power consumed by a 1GB DDR2 SDRAM rank in different states

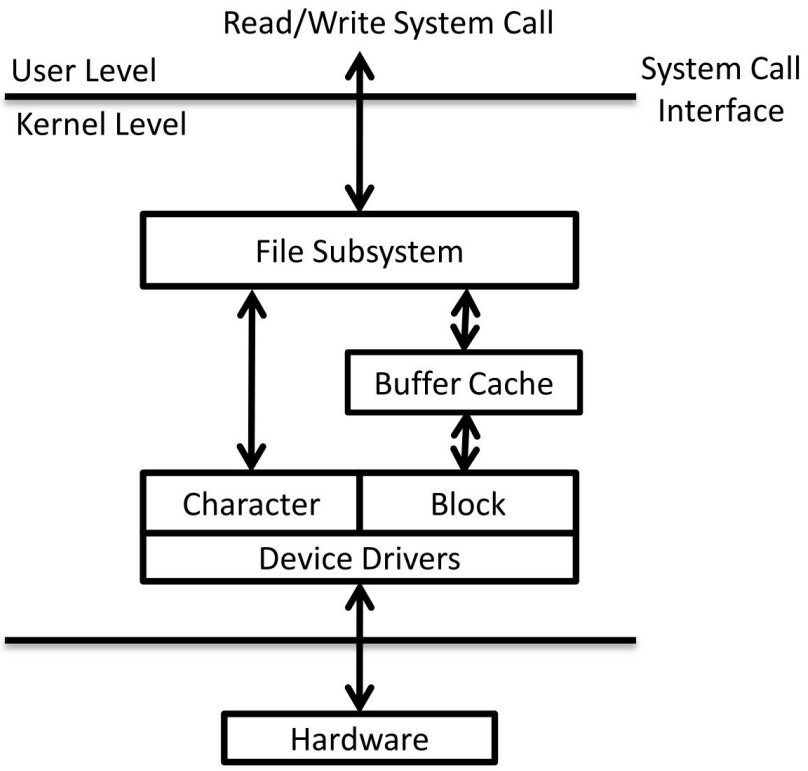

Figure 2.3: Position of the buffer cache during file-I/O handling

2.4 The Buffer Cache

Operating systems can read and write data directly from disks but this affects the file system response time and throughput as disks are very slow. To improve file system performance, the kernel maintains a pool of buffers in memory to hold data of the recently accessed file blocks from the disk. This pool of buffers in memory is known as *buffer cache*. Figure 2.3 shows the position of the buffer cache for a file-I/O request from an application. As depicted in the figure, buffer cache stores data blocks from block devices only. Each entry in the buffer cache stores data from a unique block, identified by its block number within a unique device. For the scope

of this work, the device is the *disk* and each block has a unique number within the disk.

Each in-memory buffer has two parts– a buffer header and a physical location in memory where data of the disk block is stored. For every data block, there is a unique buffer header. The list of all the buffer headers is managed as a hash table by the kernel and within each bucket of the hash table, buffer headers are arranged in a doubly linked list. The size of the buffer cache is decided during system initialization based on the amount of memory available. In case the buffer cache is full and a request comes for a block that is not present in the buffer cache, the kernel replaces an existing block from the buffer cache using one of the well-known block replacement policies. Following sections briefly describe how reads and writes are performed in presence of an in-memory buffer cache.

2.4.1 Reads

When a read request for a block from a process comes, the kernel first searches through the hash table to find if the block is present in the buffer cache. If there is a match in the hash table, the block is in the buffer cache and the read request is sent to the memory controller. Memory controller returns the block to the process that had issued the read request. When there is a miss in the hash table, the kernel retrieves the block from the disk and returns the block to the process. This involves updating the buffer cache and the hash table with information of the new block.

2.4.2 Writes

A process is usually suspended for reads but not for writes and hence, the kernel delays writing dirty blocks from the buffer cache to the disk. This improves file

system performance because a number of writes to a block can be flushed to the disk in a single disk write request. When buffer cache is full or the number of dirty blocks in the buffer cache is too large, the `bdflush` kernel thread wakes up and flushes the dirty blocks to the disk. In Linux, minimum number of dirty blocks to wake up `bdflush` is 40 [6]. It is also possible that the number of dirty blocks is less than 40 and the dirty blocks stay in the buffer cache for a long time. In such cases, the `kupdate` kernel thread flushes the dirty blocks every 30 seconds to the disk [6]. A process might also require to flush its dirty blocks to the disk during its execution. This can be accomplished by using `sync()`, `fsync()` and `fdatasync()` system calls [6].

2.5 Disk

Magnetic disks provide a permanent backup to all the data accessible within a system and constitute the lowest level of the memory hierarchy. Disks are non-volatile with very high access latency (in milliseconds) and ultralow cost per bit [25]. Magnetic disks are preferred over their electronnic counterparts for permanent backup because of their low cost. Recently, a common configuration for permanent backup is the combination of a small-sized high speed electronic disk and a large volume of magnetic disk. Electronic disks, being expensive are used to store performance critical programs like operating systems and magnetic disks provides backup for other less frequently accessed data.

CHAPTER 3

Phase Change Memory

3.1 History of PCM

The notion of how external energy events can be transformed into detectable information was first described in late 1960s by Stanford Ovshinsky [17]. Gordon Moore explained a prototype of a 128-bit memory based upon phase-change material in 1970 [18]. Using phase-change material for storage took about four decades before being well-conceived by the semiconductor industry. An obvious reason for this was the high cost of wafer processing for phase-change material than other competing technologies in the market. Another reason was the high voltage required to conduct a phase change during a write operation. Traditionally, phase-change material like chalcogenide glass is used in the manufacturing of compact disks, providing an impetus to the idea of using it as a candidate for memory hierarchy. In addition, over the past four decades, the wafer processing costs and energy requirements for phase change have reduced significantly. This has opened new avenues for phase-change based storage technologies, thus introducing a competition with existing memory hierarchy components like magnetic disk, flash and DRAM.

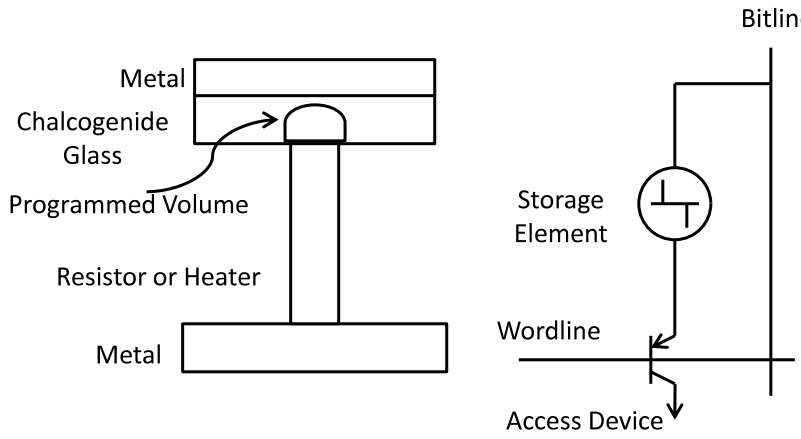

Figure 3.1: Structure of a PCM storage element.

3.2 PCM Cells

Presently, PCM uses a particular type of chalcogenide glass, known as GST or $Ge_2Sb_2Te_5$, which is an alloy of Germanium, Antimony and Tellurium. This alloy has the property that it can be switched back and forth between an amorphous and a crystalline state. The crystalline state has a very low resistivity which allows current to flow. On the other hand, the amorphous state has a very high resistivity which prevents the flow of current. This difference in resistivity, which typically is five orders of magnitude, is used to differentiate between a logical 1 and a logical 0 [19]. The crystalline state is represented as a logical 1 and the amorphous state is represented as a logical 0. Each PCM cell that holds one bit of charge also has a tiny heater or a resistor to heat the material, and a Bipolar Junction Transistor (BJT) or a Field-Effect Transistor (FET) to sense the current state of the phase change material. Because each PCM cell has one resistor and a BJT or a FET, it is usually known as a 1T1R cell. The structure of a PCM cell is shown in Figure 3.1.

The purpose of the tiny heater is to inject the current into the material so that it can be heated to a certain temperature to accomplish the phase change.

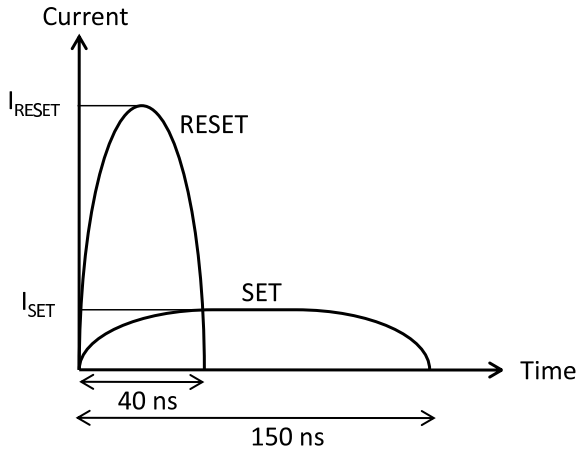

Figure 3.2: SET and RESET operations in PCM

When the material is heated above the crystallization temperature (300°C) but below the melting temperature (600°C) by applying a SET pulse, it becomes crystalline. However, if the material is heated above the melting temperature (600°C) by applying a RESET pulse, and then suddenly quenched before it can form any crystals, it becomes amorphous. The duration of the SET pulse is relatively longer than the RESET pulse. The SET and RESET pulses are shown in figure 3.2. It is also possible to turn the phase change material into a partially amorphous or partially crystalline state allowing it to show the multiple levels of resistivity. This forms the basis for the *multi-level* PCM cells [20, 21].

3.3 Reads and Writes

A SET or a RESET pulse is injected into the PCM cell for writing a logical 1 or a logical 0 respectively. SET pulse takes longer time than RESET pulse and determines the write latency for the PCM. We assume a SET latency of 150ns and a RESET latency of 40ns based on the work done by Lee et al., Bedeschi et al. and Ahn et al. [9, 20, 22].

For performing a read, the bitline shown in the figure 3.1 is precharged to the read voltage. If the memory cell is in SET state or crystalline state, the current flows from the cell to the access device and a logical 1 is detected. If the memory cell is in RESET state or amorphous state, the current does not flow through the cell and this corresponds to a logical 0. The reads are faster than writes; and for this work, we take the read latency of 55ns [9].

3.4 Endurance and Wear Leveling

Endurance defines the lifetime of a memory system such as flash, DRAM or PCM. In general, it is the time after which a memory cell cannot be accessed reliably and access to the cell might result in data errors. For PCM, state of the cell may not be changed or the flow of current through the phase-change material is below the minimum distinguishable level for the access device. The memory cell starts wearing out once the number of writes exceed the maximum limit, called the *endurance limit* of the memory cell. The endurance limit for PCM is about 10^8, which is about thousand times more than that of flash memory [23, 27]. This is one of the reasons flash is used as disk caches where the number of accesses is usually far less than the physical memory. DRAM and SRAM have a virtually infinite endurance limit of 10^{18}, which is one of the reasons that they are promising candidates for the frequently accessed higher levels of the memory hierarchy [23].

Due to non-uniformity of writes to the memory space, few PCM memory cells wear out earlier than others. There may be a few hot cells which are approaching their endurance limit; whereas, a significant number of other cells might still be underused. The process of writing to the PCM cells uniformly, so that all the cells approach their endurance limit almost at the same time, is known as *wear-leveling*. Wear-leveling algorithms try to map the writes to the less-used cells instead of their

designated heavily-used cells, so that the overall memory space is equally utilized and the lifetime of the memory chip is improved. These algorithms attempt to maximize the lifetime of the PCM while keeping the overhead minimum. Zhou et al. propose redundant bit-writes, row shifting and segment swapping to increase the lifetime of PCM when used as main memory from 100 days to 13 years [10]. Start-Gap wear-leveling mechanism attempts to minimize the overhead of the algorithm by using only two registers called *start* and *gap* and it manages to improve the lifetime of PCM from 5% to 97% of the theoretical maximum possible [11]. Whenever a new word of data is to be written to the PCM, not all the bits of the original word are modified. Flip-N-Write checks all the bits of the original word and either writes the new word or the flipped value of it [12]. With this strategy, the lifetime of the PCM was doubled. Overall, there is substantial scope for improvement to PCM lifetime before it can be commercially deployed as main memory in the current memory hierarchy.

3.5 Scalability

Then number of bits that can be programmed on a given chip-area is limited by the *minimum feature size* of the cell. Lower the feature size, denser is the chip. The feature size is further limited by the amount of the phase-change material (for example, chalcogenide glass) that needs to be present for each bit. The number of electrons stored in this material must be above the minimum level distinguishable by the access device. While PCM and flash memory are equally dense, PCM is about four times denser than DRAM [8]. It is expected that over the next few years, the minimum feature size of flash memory would approach a limit of 20nm [26]. However, in case of PCM, a 32nm device prototype has been already demonstrated and it is expected to further scale down to 9nm [19, 23]. One of the reasons why

PCM can scale down to this limit is because the effect of programming current of one cell on the states of other neighboring cells is negligible with decreasing feature size [24]. Also, with this, the electrode phase-change material contact area decreases leading to an increase in the resistivity of the material; causing the programming current to decrease. This implies that PCM would eventually become denser than flash memory. Another advantage is that with a smaller feature size, less amount of programming current is required to heat the phase-change material resulting in significant energy savings for the same capacity of the storage technology.

3.6 Energy

DRAM is already hitting power limits due its volatile nature [1]. Even if the memory is idle, DRAM has to be refreshed periodically. On the other hand, PCM consumes almost zero power in the idle state because of its non-volatile nature. However, the downside of PCM is that the I/O energy consumed is higher than that of DRAM. This is because in order to perform a write, the material has to be heated above the melting temperature during RESET and above the crystallization temperature during SET. The RESET pulse is short and injects high current to heat the phase-change material resulting in higher energy consumption than the SET pulse, which is relatively longer and operates at a lower current. A recent study shows that the array read and write energy of PCM is about two times and about forty-three times higher than DRAM [9]. Based on this work, we consider a read energy of 2.47pJ/bit and a write energy of 16.82pJ/bit for the scope of our experiments.

3.7 Cost

The most important factor that decides the presence of any technology in the market is its cost. A cost comparison between PCM and DRAM has been shown in [25]. The current cost of PCM is an order of magnitude higher than DRAM, however, it is expected to improve over the next few years with better process scaling and moving from single-level cells to multi-level cells [25, 26]. When compared to flash memory (NAND), the cost of PCM is about two orders of magnitude higher [26]. The reason why flash can be packed in higher densities at a lower price is because the programming is done at block granularity where each block comprises of thousands of bits. On the contrary, for PCM and DRAM, the programming is done at bit granularity.

3.8 PCM in the Memory Hierarchy

3.8.1 PCM as CPU Caches

Static RAM or SRAM is used for high performance CPU caches– L1, L2 and usually off-chip L3. Unlike DRAM that is optimized for density, CPU caches are optimized for performance and are less denser than DRAM due to larger number of transistors required to hold the charge in order to make them static. The long I/O latencies of PCM render it impossible to replace CPU caches. Also, because CPU caches constitute the most frequently updated level(s) of the memory hierarchy, PCM needs a virtually infinite endurance limit to replace them. Unfortunately, current state of PCM does not seem to have the characteristic to replace the I/O latencies and endurance limit of CPU caches and it is hard for PCM to find a place closer to the processor in the memory hierarchy.

3.8.2 PCM as DRAM

Dynamic RAM or DRAM is currently used as main memory in personal computers and commercial servers. A detailed description of the characteristics and working of DRAM has been given in the previous chapter. An important concern about DRAM is its power hungry nature. PCM consumes far less power than DRAM in spite of its high energy writes. Currently, the I/O latencies of PCM are higher than DRAM and it will take few years before PCM can match the performance of DRAM [9]. At present, research focus is on using a hybrid combination of DRAM and PCM as main memory where DRAM is used to hold the hot virtual memory pages to hide the longer latencies of PCM [8]. Our work in this thesis exclusively focuses on using PCM as buffer cache for commercial file servers and DRAM here caches file blocks instead of virtual memory pages. In infrastructure that use large servers, more than 90% of DRAM is dedicated to buffer cache and the main function of these servers is to handle file-I/O system calls efficiently by minimizing the access to the secondary storage.

It is also a major challenge for PCM to match the theoretically infinite endurance limit of DRAM. Zhou et al. proposed mechanisms to improve the lifetime of PCM and selected better peripheral circuitry to improve its I/O latencies, to enable it to be used as a replacement to DRAM on the memory bus [10]. Work done by Lee et al. focuses on buffer re-organization and partial writes, that is, writing only the dirty cache lines to the main memory, thus saving energy and reducing the number of writes to the main memory [9]. PCM is more scalable than DRAM and will definitely provide better capacity and density for the same chip area in the future. Although PCM has substantial potential for being included as main memory in the memory hierarchy, it might take years before it replaces an established technology like DRAM from the market.

3.8.3 PCM as Flash/Disk

PCM has better scalability, I/O latencies and write endurance limit than both NOR and NAND flash memory. Hence, flash memory looks like an easy target for PCM to replace. However, outperforming the cost of flash memory is a major challenge for PCM to meet. A similar analogy applies to disks. In spite of their slow speeds and high power consumption, their availability in high volumes at low prices would make it difficult for PCM to replace them as backup storage devices.

CHAPTER 4

Hybrid Buffer Cache Architecture

This chapter begins with an introduction to file-I/O system call handling. Sections 4.2-4.4 describe the proposed hybrid buffer cache architecture and the details of the various schemes for energy and performance improvement. In section 4.5, we change LRU, the de-facto replacement policy for buffer cache with LIRS and Optimal Replacement Policy.

4.1 File I/O System Call Handling

To access file system services, an application executes a file-I/O system call; for instance, a read or a write system call. Parameters to these system calls include a file descriptor, a buffer to hold the data and the number of bytes to be accessed. The kernel uses the file descriptor to retrieve the corresponding file-table entry and the *inode* of the file, and then tracks information about the blocks of the file in the buffer cache. This information about all the blocks in the buffer cache is maintained as a hash table by the kernel. If a hash table entry exists for the block, it is retrieved from the buffer cache; otherwise, the block is accessed from the disk. Figure 4.1 shows the steps performed during file I/O. On successfully locating the block in the buffer cache, the request to access the block from DRAM is sent to the memory controller. Memory controller performs the intended read or write operation.

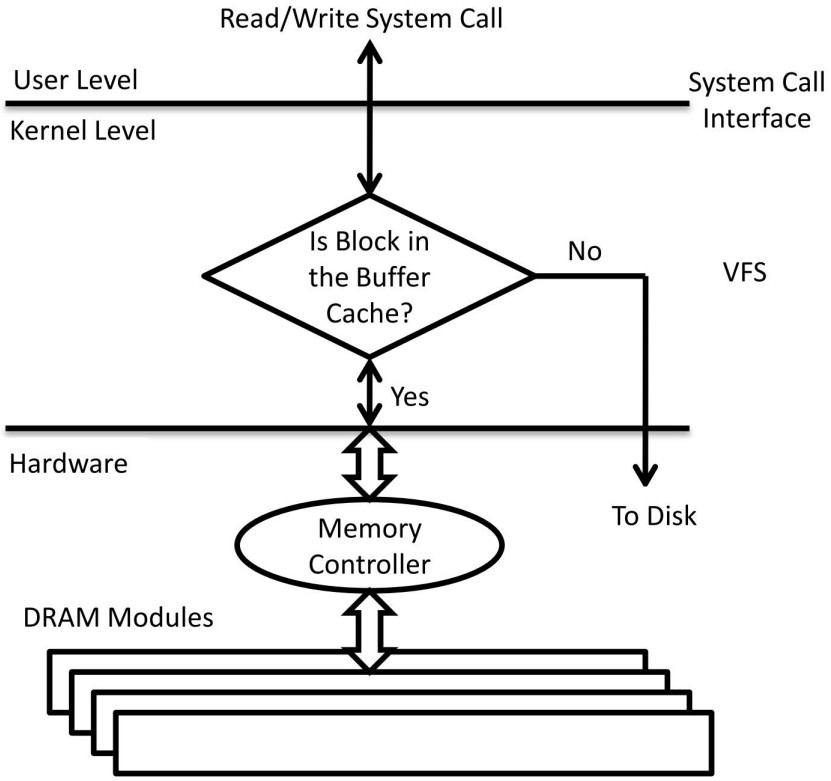

Figure 4.1: File-I/O system call handling

4.2 Baseline Buffer Cache using PCM

In this architecture, DRAM is replaced by PCM. PCM consumes lower energy than DRAM in the standby state resulting in significant energy savings; particularly in commercial file servers where hundreds of gigabytes of DRAM is used for buffer cache. We use this as our baseline architecture for the scope of this thesis. Figure 4.2 depicts this architecture. The sections that follow augment this baseline architecture to minimize the performance loss due to longer I/O latencies of PCM as compared to DRAM.

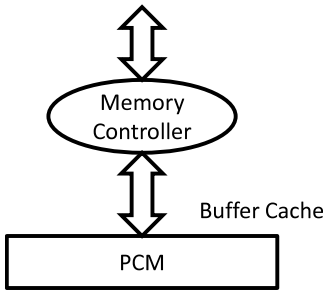

Figure 4.2: PCM as buffer cache

4.3 Hybrid Buffer Cache using PCM and DRAM

The performance of file-I/O intensive applications is adversely affected by longer read and write latencies of PCM. A comparison of the I/O energy and latency for DRAM and PCM is given in the table 5.2. Motivated by the advantages of the lower read and write latencies of DRAM and better energy efficiency of PCM, we designed a hybrid buffer cache using both PCM and DRAM. The following section describes the configuration in which DRAM is used as a cache to PCM.

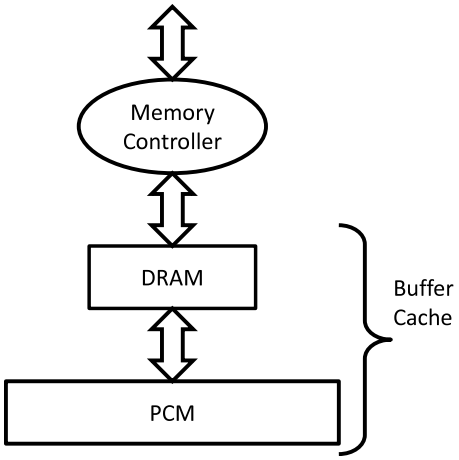

Figure 4.3: DRAM used as a cache to PCM

Figure 4.3 shows our hybrid buffer cache architecture. In order to determine the effect of DRAM-cache size on memory access time, the DRAM-cache size is

varied from from 3% to 6% of PCM. The DRAM-cache in this configuration is used only for read requests. If the requested block is not found in DRAM, the block is brought into DRAM-cache from PCM by replacing the least recently used block from the DRAM-cache. Writes are performed directly to PCM. However, if a block is present in both DRAM-cache and PCM, we use *write-through* approach by writing the block in both the DRAM-cache and PCM.

4.4 Dynamic Switching

In our hybrid buffer cache architecture, the inclusion of DRAM-cache is not always beneficial, particularly when hit ratio in the DRAM-cache is poor. In such cases, DRAM-cache may elongate the memory access time because most of the accesses to the DRAM-cache are misses and the block has to be accessed from PCM. To achieve a better performance out of this architecture, we augment the memory controller with a switching mechanism. Based on the execution of the application so far, the memory controller may choose one of the two routes for accessing the current block from the buffer cache. The first route can directly access the PCM without accessing the DRAM-cache. The second route first checks the DRAM-cache for the block being accessed and a miss results in an access to PCM. We refer to the first route as *pcm-route* and the second route as *dram-route*. A pictorial depiction of the two routes is shown in the figure 4.4.

Kernel maintains a temporary data structure called *meta-cache* that simulates the behavior of the DRAM-cache in hybrid architecture of the section 4.3 to select a route. From this *meta-cache*, memory controller decides if it is appropriate to continue using the DRAM-cache. Accordingly, it switches between the *dram-route* and the *pcm-route*. DRAM-cache is flushed whenever we switch from the

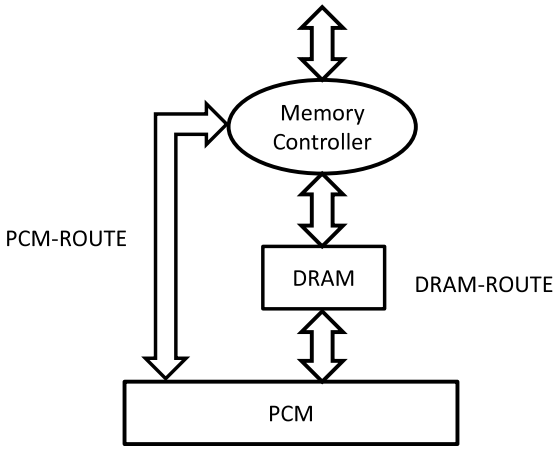

Figure 4.4: Buffer cache architecture for dynamic switching.

dram-route to the *pcm-route*. Memory controller selects the better route on the basis of a certain parameter during the execution of the application. Two parameters we have used are delay and hit ratio.

4.4.1 Switching on the Basis of Delay

This approach records the delay encountered during the execution of an application for the *dram-route* and the *pcm-route*. Memory controller dynamically chooses the route with the shorter delay. For each block access, if there is a hit in the *meta-cache*, the total delay for the two routes is calculated as below:

```
total delay using dram-route  =  total past delay using dram-route +
                                 DRAM access time for the current block

total delay using pcm-route  =  total past delay using pcm-route +
                                PCM access time for the current block
```

When there is a miss in the *meta-cache*, the delay is calculated as:

$$\texttt{total delay using dram-route} = \texttt{total past delay using dram-route} + \texttt{DRAM access time for the current block} + \texttt{PCM access time for the current block}$$

$$\texttt{total delay using pcm-route} = \texttt{total past delay using pcm-route} + \texttt{PCM access time for the current block}$$

Memory controller decides route for the current request on the basis of the delay encountered up to the previous request. Memory controller is already set to the correct route before it encounters the I/O request of the current block. Hence, checking the *meta-cache*, calculating the delays for both the routes and switching does not come in the critical path and they don't affect the total memory access time.

4.4.2 Switching on the Basis of Hit Ratio

The time to read a block from the buffer cache depends on read latency, read bandwidth and size of the block. We found that the block size for most of the accesses from the benchmarks we used in our experiments is 4KB. On the basis of these three parameters, we calculated the ratio of the total time to read a 4KB block from DRAM to that of PCM. We refer to this constant as *hit-constant*.

$$\texttt{hit-constant} = \left(\frac{(\texttt{4KB/read bandwidth of DRAM}) + \texttt{read latency for DRAM}}{(\texttt{4KB/read bandwidth of PCM}) + \texttt{read latency for PCM}} \right)$$

The read bandwidth of DRAM is 6400 MB/sec and the read latency is 12.5ns [9].

The read bandwidth of PCM is 4096 MB/sec and the read latency is 55ns[9]. We substitute these values in the equation above.

$$\texttt{hit constant} = \left(\frac{0.622}{0.953}\right) = 0.6526 \text{ or } 65.26\%$$

Based on the size of most of the block requests we observed in our benchmarks, we chose this as an approximate value, even though the exact value of hit-constant should vary depending on the size of the I/O request. The difference in the I/O latencies and bandwidth of PCM and DRAM certainly suggest that there must be a high hit ratio in the DRAM-cache for the *dram-route* to perform better than the *pcm-route*.

If the hit ratio in the *meta-cache* is equal to the hit constant, the total delay using both routes will be same. If the hit ratio in the *meta-cache* is more than the hit constant, dram-route is less expensive than the *pcm-route*, and vice-versa. Kernel keeps track of the hit ratio in the *meta-cache* and as long as the hit ratio is less than the hit constant, *pcm-route* is used to access the current block. When the hit ratio in the *meta-cache* exceeds hit constant, the kernel switches to the *dram-route*. The switching always favors the route with less delay to minimize the total execution time. The hit ratio in the *meta-cache* up to the previous request decides the route for the current block and the time it takes to calculate the hit ratio does not affect the total memory access time.

4.4.3 Energy Savings from Dynamic Switching

Dynamic switching utilizes the DRAM-cache when appropriate to improve total execution time, otherwise powers it down to save energy. When memory controller switches from *dram-route* to *pcm-route*, we flush DRAM and switch it to *power-down* mode. In *power-down* mode, DRAM consumes half the power it consumes in

precharged mode and the switching delay between the two states is 2.5ns [2].

4.4.4 Support for Dynamic Switching

The *meta-cache* management and processing is done by OS in the background without affecting the performance of the current file-I/O system call. However, there must be some communication mechanism between OS and memory controller, so that OS can relay its decision regarding path selection to the memory controller. To accomplish this, we add a flag in the memory controller. OS sets this flag and memory controller checks it at the end of the current request to set it to the appropriate path before the next I/O request arrives. The storage overhead of the flag is trivial compared to storage that is embedded into the current-day memory controllers. Processing *meta-cache*, setting the flag by OS and checking the flag by memory controller does not delay the processing of the current I/O request.

4.5 Replacement Policies for the Hybrid Buffer Cache

To exploit our design of the hybrid buffer cache in the most suitable manner, we modified the *Optimal Replacement Policy* and implemented *Low Inter-reference Recency Set (LIRS)* to check how our hybrid architecture performs with a replacement policy better than LRU [13, 14]. A brief description of each policy and the modification proposed are given in the sections below.

4.5.1 LIRS

In case of access patterns with weak locality, LRU replacement policy performs poorly. For example, when application accesses file blocks in a sequential manner

or the size of a repeatedly accessed file is little larger than size of the buffer cache. There are many algorithms proposed to handle the pathological cases of LRU e.g. LRU-K [15] and 2Q [16]. However, all these policies considerably increase the overhead of LRU. LIRS provides remedy to all the pathological cases of LRU with maintaining its simplicity and keeping the hot blocks in buffer cache all the time. LIRS makes replacement decisions on the basis of the *Inter Reference Recency (IRR)* of a block, which is the distance between the previous two accesses of a block. On the basis of the IRR, it divides blocks referenced by an application into two sets– LIR block set and the HIR block set. LIR block set contains the block with low IRR, that is, the blocks accessed frequently. However, the HIR block set holds the blocks with high IRR, that is, the blocks accessed less frequently. LIR block set always resides in the buffer cache, thus preventing the replacement of the hot blocks from it.

The idea of separating hot blocks from the cold blocks is well suited to our hybrid buffer cache design. Because I/O accesses to the DRAM-cache are faster compared to PCM, LIR block set can always be stored in the DRAM-cache. If the application has a significant level of locality in access patterns, it is unlikely that the LIR block set will be replaced out of the DRAM-cache. Once a fixed portion of the DRAM-cache has been allocated to the LIR block set, the remaining portion of the DRAM-cache and the entire PCM can be used to store the HIR block set. Hence the whole buffer cache blocks are either stored as LIR blocks or HIR blocks, depending on the IRR of the blocks.

4.5.2 Optimal Replacement Policy

Optimal Replacement Policy (OPT) replaces a block that is not accessed for the longest period of time. In order to find a victim block that should be replaced

from the buffer cache, the policy requires knowledge of the block accesses from the application in advance. To asses the performance of our hybrid buffer cache of section 4.3, we augmented the optimal replacement policy to replace a victim block from the DRAM-cache. For the purpose of implementation, the application has to be run twice– first time to store the block accesses from the application and second time, on the basis of the history stored during the first run, the victim block is selected for replacement from the buffer cache. During the first run of the application, in addition to storing information such as file inode, block number, request type (read, write, open etc.) and size of the request, we also store the *forward distance* of a block. *Forward distance* is defined as the number of block accesses from the application between the current access and the next access of the same block.

In our modified OPT scheme, if a block is accessed for the first time, we access it directly from the PCM; otherwise, we search through the DRAM-cache to find the victim block for the replacement. In case the forward distance of the current block is more than the forward distance of the victim block, we again directly access the PCM, because the victim block will be accessed before the current block. If there is a victim block in the DRAM-cache that has the *forward distance* less than the current block, it is selected for replacement from the DRAM-cache.

Hence, based on the *forward distance*, OPT always maintains the most frequently accessed set of blocks from the application in the buffer cache. In effect, OPT provides us with the theoretical minimum memory access time that we can achieve from our hybrid buffer cache architecture.

CHAPTER 5

Methodology and Evaluation

This chapter describes the simulation environment and evaluation of the energy requirements and performance of our hybrid buffer cache architecture.

5.1 Benchmarks

In order to evaluate our proposed schemes, we considered five benchmarks with different workload characteristics. The traces for these were obtained by modifying the *strace* Linux utility. *strace* obtains the inode number, the type of system call, size of the request and the current timestamp of an application by intercepting its file-I/O system calls. Table 5.1 shows the size of the data read and written, the hit ratio in the 4GB buffer cache and the hit ratio in the 128 MB DRAM-cache for each application. We now discuss each of these benchmarks in detail.

Application	Read Size (MB)	Write Size (MB)	Hit Ratio (4GB)	DRAM-cache Hit Ratio (128 MB)
Glimpse	33482.01	29542.19	94.09%	93.41%
Postmark	31371.48	32200.97	86.82%	4.01%
TPCC	346091.86	28545.75	98.99%	36.95%
TPCH	5342.76	47.25	95.36%	69.45%
Viewperf	11780.26	1.54	79.66%	83.89%

Table 5.1: Benchmark characteristics

Glimpse (GLobal IMPlicit SEarch) is a text indexing and retrieval program developed at The University of Arizona[28]. It builds the indices and searches through the entire file system. We have only considered the indexing part of *Glimpse* for our work.

Postmark is a mail-server benchmark designed to measure the transaction rates for a workload approximating a large mail server.

TPCC is an online transaction processing benchmark that simulates an intense computing environment where a large number of users execute transactions against a database. This kind of workload is found in an order-entry environment where a large number of order and delivery requests are processed.

TPCH is a decision support benchmark that queries a large database to address critical business problems.

Viewperf is an OpenGL performance benchmark from SPECviewperf. It performs stress tests against OpenGL and reports the results in frames per second.

5.2 Simulation Environment

We augmented the *Linux Kernel 2.4* buffer cache simulator with a physical memory simulator [2, 7]. The physical memory simulator simulates DRAM and has been further modified to simulate PCM. For both DRAM and PCM, we measured the hit ratio in the buffer cache, total energy consumed by the physical memory and total execution time for each trace. Table 5.2 lists the various parameters used to measure the energy and total execution time. For PCM, we assume a memory organization similar to DRAM; that is, memory is organized as channels, ranks, banks, rows and columns. Read latency is the time it takes to move data out of

a bank to the row buffer. Write latency is defined as the time it takes to write data from the row buffer to the memory bank. DRAM ranks are assumed to be always in precharged state. Read and write energy values denote the additional energy consumed in performing I/O in this state. We use the read/write latency and read/write energy values reported by Lee. et al. in [9] for PCM. We assume idle power of 8 mW/GB for PCM. Power consumed by a 1 GB DRAM rank in precharged state is 1877.1 mW based on [2, 4].

Attribute	DRAM	PCM
Read Latency (ns)	12.5	55
Write Latency (ns)	12.5	150
Read Energy (pJ/bit)	2.18	3.48
Write Energy (pJ/bit)	1.49	17.92
Read Bandwidth (MB/sec)	6400	4096
Write Bandwidth (MB/sec)	6400	400
Idle Power (mW/GB)	1877.1	8
Endurance	∞	10^8
Non-volatile	No	Yes

Table 5.2: Energy and timing parameters for DRAM and PCM

5.3 Evaluation

Achieving the energy efficiency of PCM and performance of DRAM at the same time is difficult to obtain. When we replace DRAM with PCM, the memory energy requirements are reduced exponentially. However, the longer I/O latencies of PCM hurt the memory access time affecting performance. In this section, we evaluate the mechanisms we proposed from the point of view of performance and energy-efficiency.

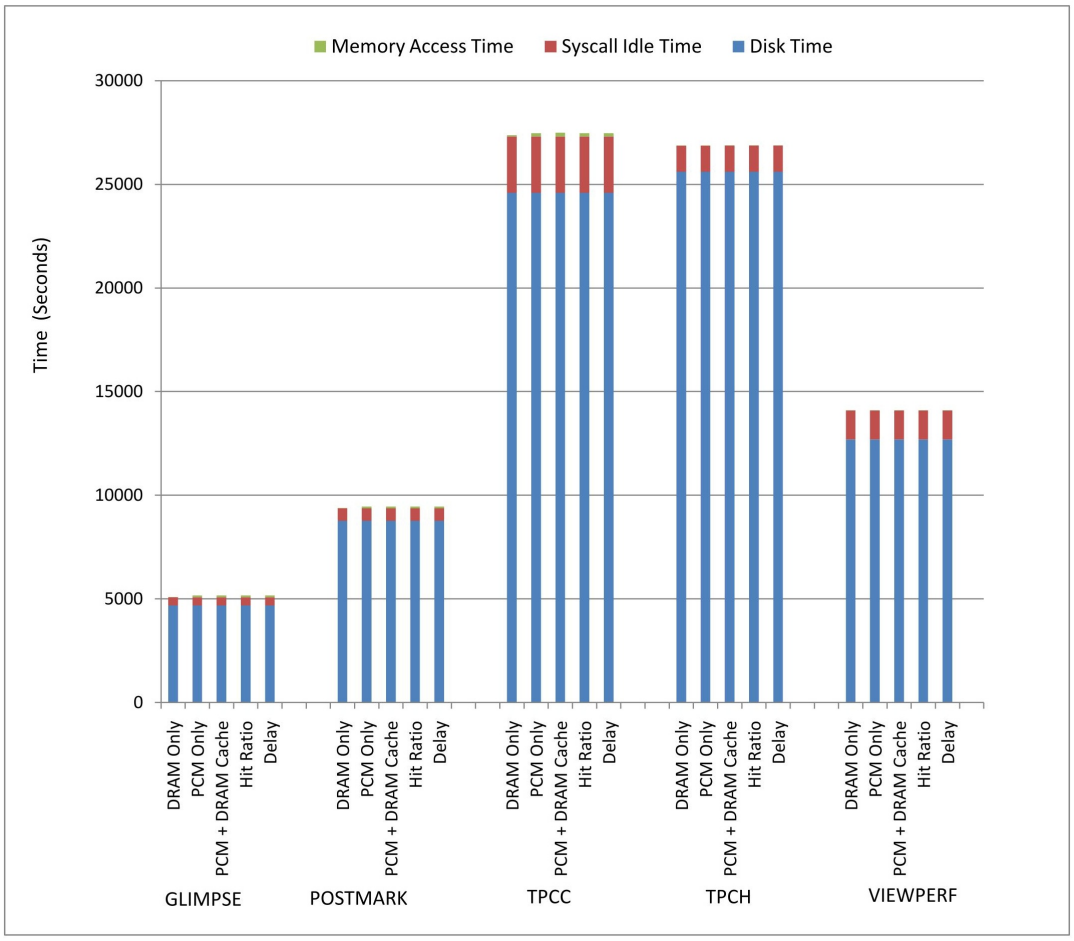

Figure 5.1: Total execution time for various workloads.

5.3.1 Total Execution Time

We now examine the total execution time and memory access time for the schemes that we proposed. For this discussion, we shall use the term *buffer cache* to refer to the 4GB buffer cache and the term *DRAM-cache* to refer to the DRAM that we use in our hybrid configuration. Figure 5.1 represents the total time taken by each application for various schemes. We have measured the total execution time as:

$$\texttt{Total execution time} = \texttt{Memory access time} + \texttt{System call idle time} + \texttt{Disk access time}$$

Memory access time is the total time taken for accessing 4GB buffer cache plus the DRAM-cache access time.

If there is a miss in the buffer cache, the block requested by the application needs to be retrieved from the disk. Disk access time constitutes the total time taken by all such misses.

The idle time is defined as the delay between the end of the current system call and the beginning of the next system call. The summation of all such idle times across the application's execution is referred to as system call idle time.

Figure 5.2 shows that the memory access time is less than 2% of the total execution time for all the applications. In most cases, more than 88% of the total execution time of an application is spent in disk accesses. System call idle time contributes to about 10% of the total execution time. Thus, main memory is idle most of the time and there is an excellent opportunity to save energy by using less power hungry memory technologies such as PCM. An alternative is to switch DRAM to a low power mode during these idle periods. However, switching among the power modes takes time and the memory controller might have to put a memory I/O request on wait until the requested rank is back to the precharged state. A detailed description of energy management for DRAM with minimizing the switching delay has been given in [2].

Figure 5.3 depicts the memory access time for each of the benchmarks for the proposed schemes. When we replace DRAM with PCM, the memory access time increases across all applications because of longer I/O latencies of PCM. However,

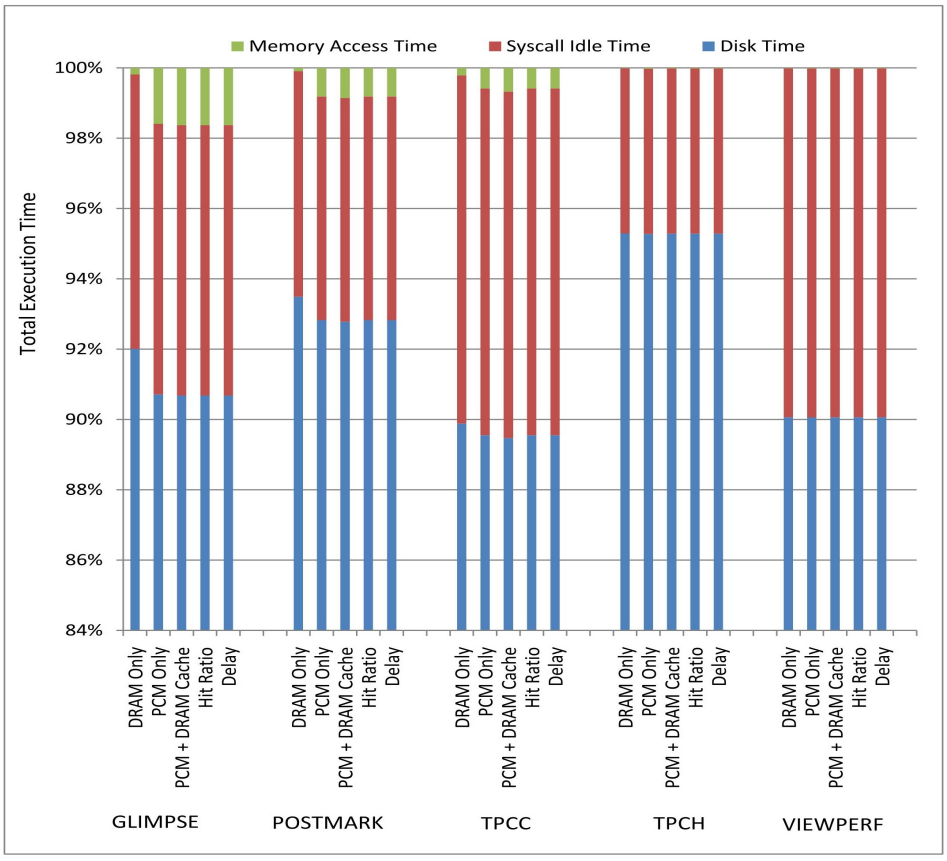

Figure 5.2: Percentage of the total execution time taken by the disk, memory and idle periods between system calls.

as shown in figure 5.5, this results in siginificant energy savings. In order to achieve the energy savings closer to the PCM-only system and performance closer to the DRAM-only system, we propose a hybrid configuration with DRAM acting as a cache to the 4GB PCM buffer cache.

Glimpse

The hit ratio for Glimpse in the DRAM-cache is over 90%. Contrary to our expectations, even with a high DRAM-cache hit ratio, a small increase in the memory access time is observed when compared to the PCM-only scheme. This

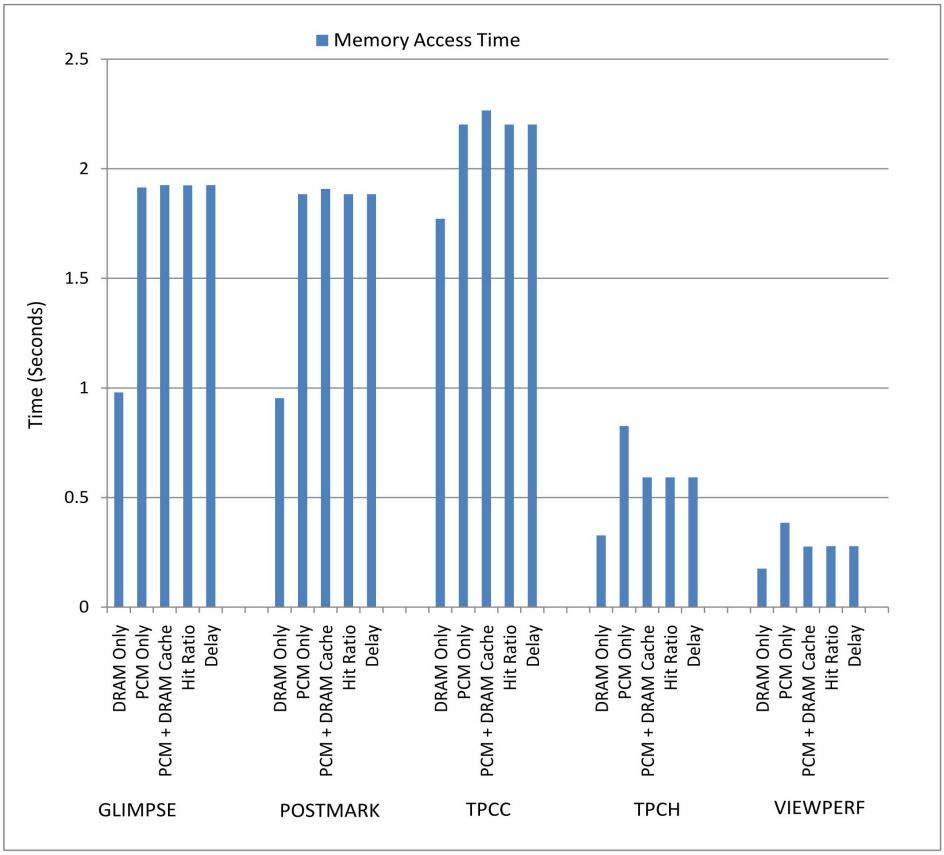

Figure 5.3: Memory access time of various benchmarks for different mechanisms.

is because almost half the I/O requests from Glimpse are write requests and a significant number of these write a block in both DRAM and PCM. Performing two writes instead of one increases the total execution time for all the schemes using the DRAM-cache for Glimpse.

Postmark

Postmark has a hit ratio of 4.01% in the DRAM-cache. Dynamic switching schemes based on delay and hit ratio access PCM directly, thus bringing the memory access time close to the PCM-only scheme as it performs better than the hybrid version. Dynamic switching attempts to prevent introducing an adverse effect on

the total execution time of the application by selecting the PCM-only version over the hybrid version with the DRAM-cache.

TPCC

The hit ratio for TPCC is about 35%. However, this is not sufficient to hide the long I/O latencies of PCM. Dynamic switching keeps it closer to the PCM-only scheme and results in significant energy savings.

TPCH and Viewperf

TPCH and Viewperf have very high hit ratios in the DRAM-cache and this leads to the significant performance improvement in case of the hybrid mechanism. An improvement of 58% and 78% respectively is observed in the memory access times when compared to the PCM-only system.

Memory access time with increasing DRAM-cache size: To determine the effect of the DRAM-cache size on the application's hit ratio, we increased the cache size from 128MB to 256MB. As depicted in figure 5.4, an improvement in memory access time for benchmarks with reasonable hit ratio (like TPCC, TPCH and Viewperf) was observed. However, the DRAM-cache size did not affect Glimpse and Postmark considerably. For Glimpse, the number of writes in both DRAM-cache and PCM increase with the DRAM-cache size, canceling the performance improvements of the increased hit ratio. For Postmark, the hit ratio in the DRAM-cache increases from about 4% to 7%, which implies that the hit ratio for Postmark is almost proportional to the size of the DRAM-cache (87% hit ratio for the 4GB buffer cache). In conclusion, increasing the DRAM-cache size does not impact the memory access time significantly.

Overall, for applications with a low hit ratio, we were able to hide the performance and energy disadvantages of the DRAM-cache with the help of dynamic

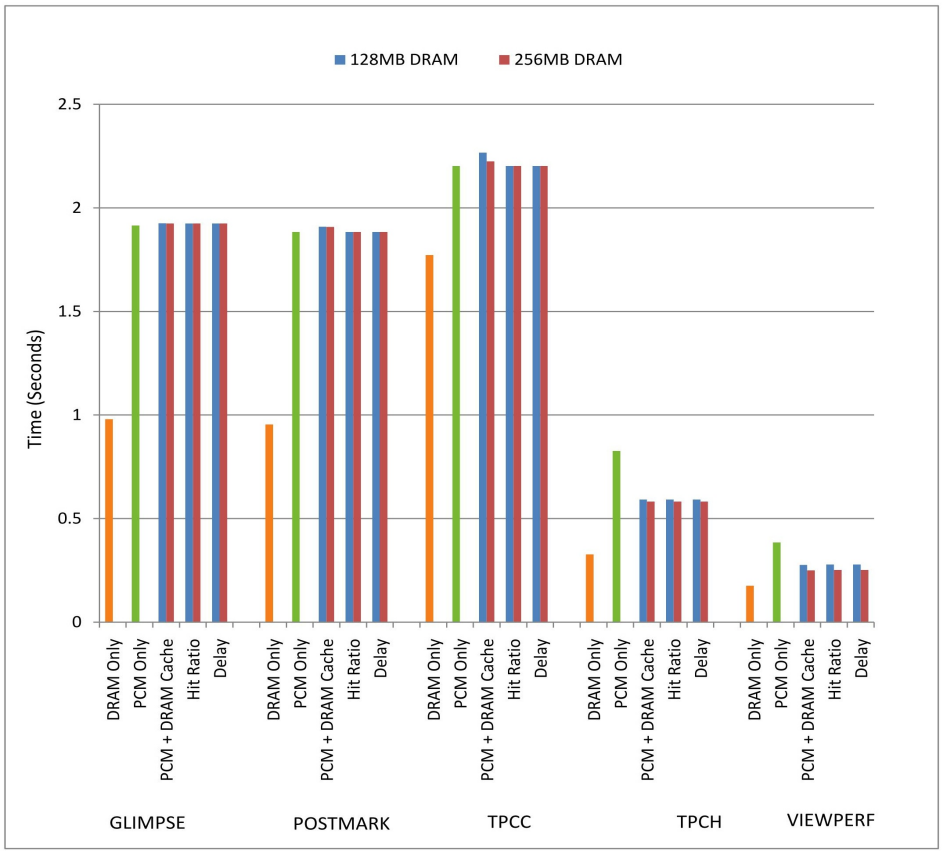

Figure 5.4: Memory access time of various benchmarks for different mechanisms when DRAM-cache size is increased from 128MB to 256MB

switching. For applications exhibiting a reasonable level of locality in their accesses, our hybrid mechanisms managed to achieve up to 78% improvement over the PCM-only version. However, lower I/O latencies of DRAM still make it the best memory technology for the real time applications where performance will always be a critical concern.

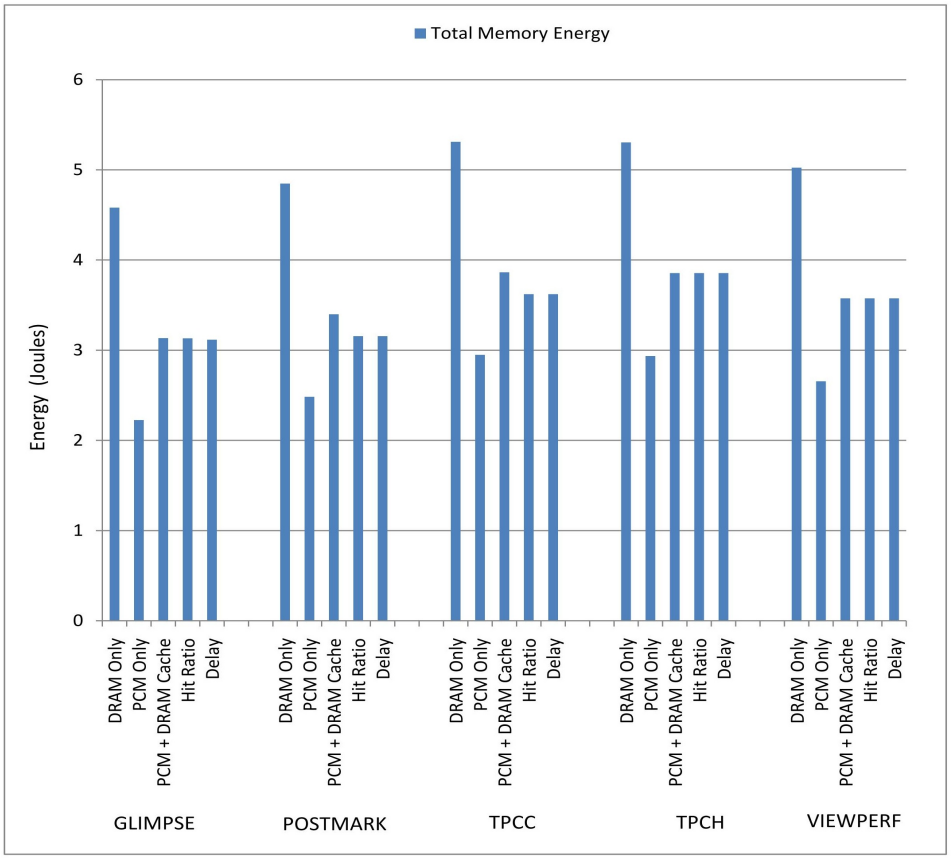

Figure 5.5: Memory energy requirements of various benchmarks for different mechanisms

5.3.2 Energy

In this section, we describe the memory energy savings that were achieved by replacing DRAM with PCM. The total memory energy is calculated as:

$$\begin{aligned} \texttt{Total memory energy} \;=\; & \texttt{Total memory read energy} + \\ & \texttt{Total memory write energy} + \\ & \texttt{Total memory idle energy} \end{aligned}$$

As inferred from figure 5.2, memory is idle most of the time. Memory idle energy is calculated as:

$$\texttt{Total memory idle energy} = \texttt{Total memory idle time} \times \texttt{Memory idle power}$$

$$\texttt{Total memory idle time} = \texttt{System call idle time} + \texttt{Disk access time}$$

The power and energy values for both PCM and DRAM are given in table 5.2. For applications with a low hit ratio, dynamic switching mechanisms set DRAM to the power-down mode. The energy consumed by a 1GB DRAM rank in power-down mode is 963.5mW [2]. Based on this, we measure the energy consumption in power-down mode for different sizes of the DRAM-cache. Figure 5.5 shows the energy consumed by memory for different schemes. Significant energy savings are obtained when DRAM is replaced with PCM. This is expected as PCM consumes low energy in standby state. Even though PCM has higher read and write energy, total memory energy is dominated by the memory idle energy, making PCM an extremely energy efficient memory technology. We also observe that adding DRAM-cache to buffer cache increases the overall memory energy consumption significantly, even when the DRAM-cache size is only 3% of PCM. We now describe the energy characteristics for each of our benchmarks.

Glimpse

The hybrid system that uses the DRAM-cache consumes significantly higher energy than the PCM-only system. When compared to the DRAM-only system, the overall energy consumption is about 4% of the total energy consumed. The dynamic switching mechanisms are were unable to switch the DRAM-cache to power-down mode. This is because Glimpse has a high hit ratio in the DRAM-cache and memory controller does not get the opportunity to switch the DRAM-cache to low power mode for energy savings.

Postmark

The low hit ratio of Postmark in the DRAM-cache provides an excellent opportunity for the dynamic switching mechanisms to power-down the DRAM-cache. We found that DRAM was powered down for the entire execution of Postmark for the hit ratio based switching mechanism and for 99% of the total execution time for the delay-based mechanism. The energy consumption of the dynamic switching mechanisms was higher than the PCM-only system. However, this was only 2% of the DRAM-only system.

TPCC

TPCC has around 35% hit ratio in the DRAM-cache. This is not sufficient for the dynamic switching mechanism to prefer the *dram-route*. Thus, the DRAM-cache was powered down and the *pcm-route* was used. For TPCC, all the mechanisms consume less than 5% of the total energy consumed by the DRAM-only system.

TPCH and Viewperf

TPCH and Viewperf have high DRAM-cache hit ratios. As a result the dynamic switching mechanisms prefer the *dram-route* and do not get an opportunity to power down the DRAM-cache. Hence, all the mechanisms using hybrid architecture consumed almost same amount of energy. Our hybrid mechanisms were able to save up to 94% of the total energy consumed by the DRAM-only system for TPCH and Viewperf, and achived better performance than the PCM-only system.

Memory energy requirements with increasing DRAM-cache size: As shown in the figure 5.6, the energy requirements for the benchmarks almost doubled when DRAM-cache size was doubled (from 128MB to 256MB). As discussed in

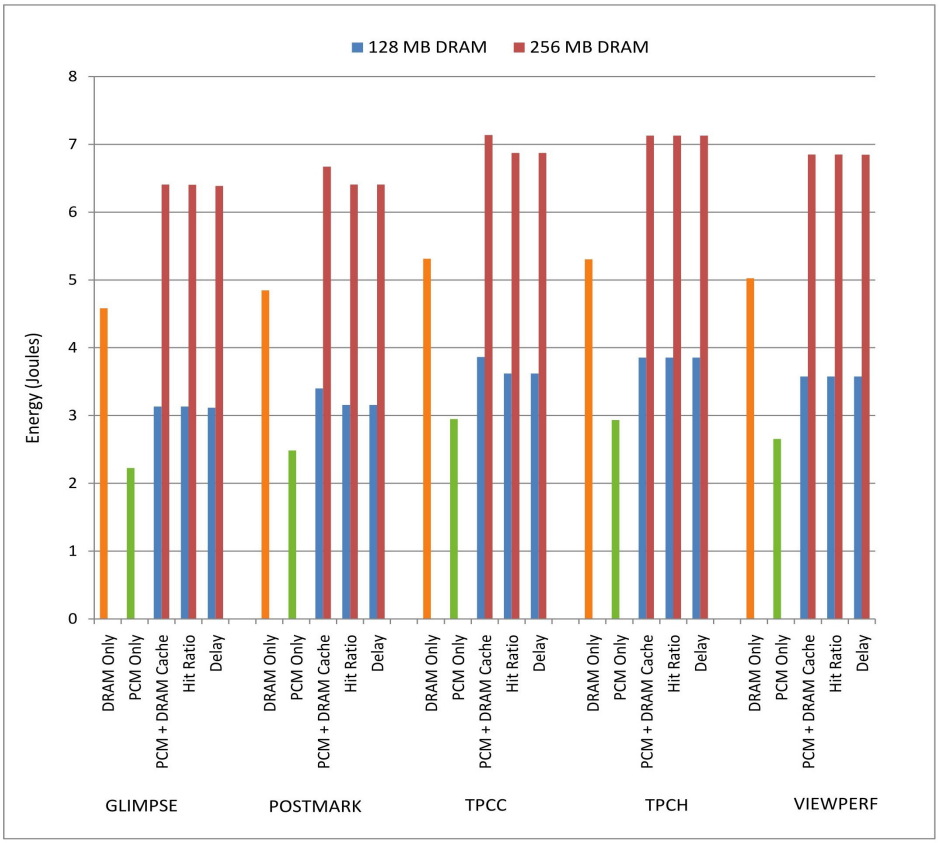

Figure 5.6: Memory energy characteristics of various benchmarks for different mechanisms when DRAM-cache size is increased from 128MB to 256MB

section 5.3.1, increasing the DRAM-cache size did not improve performance significantly. Hence, for energy-constrained servers, investing in a higher DRAM-cache size might not result in improved performance. In addition, the energy consumption would increase in proportion with the DRAM-cache size..

5.4 Memory Access Time with Other Replacement Policies

To examine the effect of other replacement policies on the DRAM-cache, we implemented *Low Inter-reference Recency Set (LIRS)* and *Optimal (OPT)*.

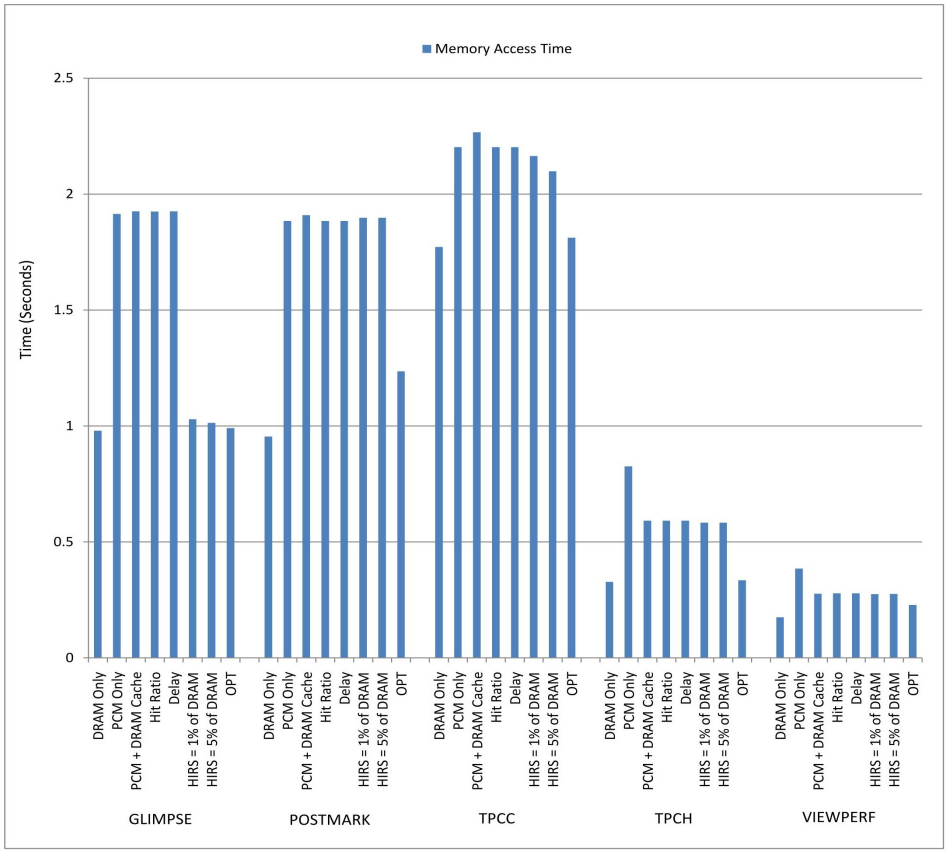

Figure 5.7: Memory access time of various benchmarks for different mechanisms with LIRS and OPT

The following sections describe these policies in detail for our hybrid buffer cache architecture.

5.4.1 LIRS

As discussed in the previous chapter, LIRS divides blocks into two sets– blocks with low IRR (LIR) and blocks with high IRR (HIR). LIR blocks are always maintained in the DRAM-cache while HIR blocks can be replaced out of the DRAM-cache. The status of a block can change from LIR to HIR and vice-versa during the execution

of the application.

For our evaluation, the HIR block set size was chosen to be 1% or 5% of the DRAM-cache size, while the remaining blocks were assigned to the LIR block set. Figure 5.7 shows the memory access time for these two different sizes of the HIR block set. Since, LIRS is an improvement over LRU, hot blocks are always resident in DRAM-cache. For applications with a very high hit ratio, namely, TPCH and Viewperf, an improvement in memory access time is not expected. TPCC and Postmark benefit from the change in the replacement policy. The hit ratio increases from 37% to 48% for TPCC and from 4% to 8% for Postmark when the HIR block set size was chosen to be 1% of the DRAM-cache. Changing the HIR block set size from 1% to 5% does not impact the hit ratio in the DRAM-cache significantly. This is because most of the DRAM-cache is still occupied by the LIR block set and only a few more blocks from the LIR block set are added to the HIR block set for replacement. We also observed that when the HIR block set size was set to 5% of the DRAM-cache size the hit ratio of TPCC went from 48% to 50%.

5.4.2 Optimal Replacement Policy

In order to determine the optimal memory access time for our hybrid architecture, we evaluated the benchmarks against our implementation of the OPT policy. This involves using OPT policy for the DRAM-cache, while maintaining LRU on PCM. With such a setup, we observed that the memory access time for TPCH for our arhictecture is 1.01 times the DRAM-only system. Similarly, in case of Glimpse and Postmark, the memory access time was 1.1 and 1.3 times the DRAM-only system respectively. For TPCC and Viewperf, this multiplier was between 1.01 and 1.3 as shown in figure 5.7. In conclusion, with the current status of PCM, it is difficult for our hybrid architecture to meet the high performance provided by the

DRAM-only system.

CHAPTER 6

Concluding Remarks

The work in this thesis was devoted to address the issue of energy inefficiency of buffer cache in commercial file servers. We evaluated PCM as a new buffer cache alternative in place of the currently used DRAM. In this chapter we summarize our work, discuss possible limitations and sketch out future directions.

6.1 Conclusions

In energy constrained data centers where performance can be marginally compromised for high savings in energy requirements, using PCM in place of DRAM looks promising. PCM-only system can reduce energy requirements in data centers to 2% of the DRAM-only system, with an increase of less than 2% in the total execution time. However, the memory access time was almost doubled due to high I/O latencies of PCM. To address the issue of increased memory access time, we designed a hybrid buffer cache architecture by using a small DRAM-cache with the PCM-only system to benefit from the high performance characteristics of DRAM. We used dynamic switching successfully to manage the DRAM-cache at runtime for energy-performance efficiency. For applications with a good hit ratio in the DRAM-cache, such as TPCH and Viewperf, the route through the DRAM-cache is selected most of the time. On the other hand, for applications with a low hit ratio in the DRAM-cache, such as Postmark and TPCC, the DRAM-cache is disabled.

With dynamic switching, we were able to reduce the energy consumption of the buffer cache to 5% of the DRAM-only system with up to 78% improvement in the memory access time when compared to the PCM-only system.

Even though dynamic switching was able to lower the memory energy requirements of the buffer cache significantly, it is still difficult to meet the performance achieved by the DRAM-only system. Even with the best replacement policy for the DRAM-cache, long I/O latencies of PCM will make it hard to meet the performance of the DRAM-only system.

6.2 Limitations

In Chapter 5, we observed that memory was idle for more than 90% of the time. Primary reason for high idle time is because we were running only one benchmark at a time. In commercial file servers, where more than one process is expected to run, idle time will be significantly lesser.

Even for data centers where performance can be compromised for high savings in energy, current cost of PCM will make it hard to replace DRAM. As discussed in Chapter 3, for the same size, cost of PCM is 50-100 times higher than that of DRAM. Current status of PCM is good for research purposes only and DRAM will continue leading the end user market for next few years until the issues of high I/O latencies, high write energy, lower endurance and the most important one, the cost of PCM are taken care of.

Dynamic switching requires support from both the memory controller and the operating system. Some additional storage is required to maintain meta-cache in the kernel. This additional storage would depend on the size of the DRAM-cache.

6.3 Future Work

This work can be extended in the following possible directions. First, size of the DRAM-cache can be varied at runtime. Our evaluation considered only one DRAM rank. Data centers today comprise of large buffer caches and the DRAM-cache can have multiple ranks. Memory controller can change the number of DRAM-cache ranks powered-on at runtime depending on the workload and the locality of reference of the applications.

Also, for write requests, if a block is present in both PCM and the DRAM-cache, we update the block at both the places simultaneously. Memory access time for applications like Glimpse, that perform a large number of simultaneous writes to PCM and the DRAM-cache can be improved by delaying writes to PCM. We can flush updates from the DRAM-cache periodically to PCM, just the way kernel writes dirty blocks from the buffer cache to the disk using `bdflush` or `kupdate`. Another option is to perform lazy writes to PCM as proposed by Qureshi et al. in [8].

In this work, only one benchmark is executed at a time to evaluate the energy and performance of the buffer cache. One possible direction is to run multiple benchmarks simultaneously so that the memory idle time is reduced.

REFERENCES

[1] C. Lefurgy, K. Rajaramani, F. Rawson, W. Felter, M. Kistler, and T.W. Keller, *Energy Management for Commercial Servers*, (IEEE Computer Society, December 2003).

[2] M. Bi, R. Duan, and C. Gniady, *Delay-Hiding Energy Management Mechanisms for DRAM*, (IEEE International Symposium on High Performance Computer Architecture, January 2010).

[3] M. Lee, E. Seo, J. Lee, and J. Kim, *Pabc: Power-aware buffer cache management for low power consumption*, (IEEE Transactions on Computers, April 2007).

[4] Micron. *Calculating Memory System Power for DDR2*, (Technical Note– 4704).

[5] M. J. Bach *The Design of the UNIX Operating System*, (Published by Prentice Hall Software Series).

[6] D. P. Bovet, and M. Cesati *Understanding the Linux Kernel (3rd Edition)*, (O'Reilly and Associates, Inc., 2005).

[7] A. R. Butt, C. Gniady, and Y. C. Hu. *The performance impact of kernel prefetching on buffer cache replacement algorithms.* (SIGMETRICS 2005, June 6-10, 2005, Banff, Alberta, Canada).

[8] M. K. Qureshi, V. Srinivasan, and J. A. Rivers, *Scalable High Performance Main Memory System Using Phase-Change Memory Technology*, (ISCA 2009, June 20-24, 2009, Austin, Texas, USA).

[9] B.C. Lee, E. Ipek, O. Mutlu, D. Burger, *Architecting Phase Change Memory as a Scalable DRAM Alternative*, (ISCA 2009, June 20-24, 2009, Austin, Texas, USA).

[10] P. Zhou, B. Zhao, J. Yang, and Y. Zhang, *A durable and energy efficient main memory using phase change memory*, (ISCA 2009, June 20-24, 2009, Austin, Texas, USA).

[11] M.K. Qureshi, J. Karidis, M. Franceschini, V. Srinivasan, L. Lastras, and B. Abali, *Enhancing lifetime and security of PCM-based main memory with start-gap wear leveling*, (MICRO 2009, December 12-16, 2009, New York, NY, USA).

[12] S. Cho, and H. Lee, *Flip-N-Write: A simple deterministic technique to improve PRAM write performance, energy and endurance*, (MICRO 2009, December 12-16, 2009, New York, NY, USA).

[13] S. Jiang and X. Zhang, *LIRS: An Efficient Low Inter-reference Recency Set Policy to Improve Buffer Cache Performance*, (SIGMETRICS 2002, June 15-19, 2002, Merina Del Rey, California, USA).

[14] L.A. Belady, *A Study of Replacement Algorithms for a Virtual-Storage Computer*, (IBM Systems Journal, Volume 5, Number 2(1966), pages 78-101).

[15] E. J. O'Neil, P.E. O'Neil, and G. Weikum, *The LRU-K Page Replacement Algorithm for Database Disk Buffering*, (ACM SIGMOD 1993, pages 297-306).

[16] T. Johnson and D. Shasha, *2Q: A Low Overhead High Performance Buffer Management Replacement Algorithm*, (VLDB 1993, pages 439-450).

[17] S.R. Ovshinsky. *Reversible Electrical Phenomena in Disordered Structures*, (Phys. Rev. Lett. Volume 21, 1968, pages 1450-1453).

[18] R.G. Neale, D.L. Nelson, and Gordon E Moore. *Nonvolatile and reprogrammable: the read-mostly memory is here*, (Electronics, September 1970).

[19] S. Raoux, G. W. Burr, M. J. Breitwisch, C. T. Rettner, Y.-C. Chen, R. M. Shelby, M. Salinga, D. Krebs, S.-H. Chen, H.-L.Lung, and C. H. Lam. *Phase-change random access memory: A scalable technology*, (IBM Journal of R. and D., 52(4/5):465479, 2008).

[20] F. Bedeschi, R. Fackenthal, C. Resta, E.M. Donze, M. Jagasivamani, E. Buda, F. Pellizzer, D. Chow, A. Cabrini, G.M.A. Calvi, R. Faravelli, A. Fantini, G. Torelli, D. Mills, R. Gastaldi, and G. Casagrande. *A multi-level-cell bipolar-selected phase-change memory*, (International Solid-State Circuits Conference, 2008).

[21] T. Nirschl, J.B. Phipp, T.D. Happ, G.W. Burr, B. Rajendran, M.-H. Lee, A. Schrott, M. Yang, M. Breitwisch, C.-F. Chen, E. Joseph, M. Lamorey, R. Cheek, S.-H. Chen, S. Zaidi, S. Raoux, Y.C. Chen, Y. Zhu, R. Bergmann, H.-L. Lung, and C. Lam. *Write strategies for 2 and 4-bit multi-level phase-change memory* (International Electronic Devices Meeting, 2008).

[22] S.J. Ahn, Y.J. Song, C.W. Jeong, J.M. Shin, Y. Fai, Y.N. Hwang, S.H. Lee, K.C. Ryoo, S.Y. Lee, J.H. Park, H. Horii, Y.H. Ha, J.H. Yi, B.J. Kuh, G.H. Koh, G.T. Jeong, H.S. Jeong, Kinam Kim, and B.I. Ryu. *Highly manufacturable high density phase change memory of 64Mb and beyond* (International Electronics Devices Meeting, 2004).

[23] *SIA. Process integration devices and structures.* (International Technology Roadmap for Semiconductors, 2007).

[24] S. Lai. *Current status of the phase change memory and its future.* (International Electronics Devices Meeting, 2003).

[25] E. Doller. *Phase change memory and its impacts on memory hierarchy.* (http://www.pdl.cmu.edu/SDI/2009/slides/Numonyx.pdf, 2009).

[26] *PCM becomes reality: Memory design will never be the same.* (Objective Analysis: Semiconductor market research, 2009).

[27] R. Freitas and W. Wilcke. *Storage-class memory: The next storage system technology.* (IBM Journal of R. and D.,52(4/5):439447, 2008).

[28] U. Manber and S. Wu. *GLIMPSE: A tool to search through entire file systems.* (USENIX Winter 1994 Technical Conference, Jan. 1994).

[29] B. Jacob, S. W. Ng, and D. T. Wang *Memory Systems– Cache, DRAM, Disk.* (Elsevier, 2008).

[30] H. Kim *Lecture notes on High-Performance Computer Architecture.* (College of Computing, Georgia Institute of Technology, Fall 2010).

[31] *Report to Congress on Server and Data Center Energy Efficiency.* (U.S. Environmental Protection Agency, ENERGY STAR Program, August 2007).