View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by ScholarBank@NUS

Status of Phase Change Memory in Memory Hierarchy and its impact on Relational Database

Masters Thesis

submitted by

Suraj Pathak

suraj@comp.nus.edu.sg

under guidance of

Prof. Tay Yong Chiang

to

SCHOOL OF COMPUTING

NATIONAL UNIVERSITY OF SINGAPORE

December 2011

Abstract

Phase Change Memory(PCM) is a new form of Non-volatile memory that has advantages like read access almost as close to a DRAM, write speed about 100 times faster than traditional hard disks and flash SSD, and cell density about 10 times better than any kind of storage devices available today. With these advantages, it is feasible that PCM could be the future of data storage as it has the potential to replace both secondary storage and main memory.

In this thesis, we study the current status of PCM in the memory hierarchy, its characteristics , advantages and challenges in implementing the technology. Specifically, we study how the byte-writeable PCM can be used as a buffer for flash SSD to improve its write efficiency. Then in the second part, we study how traditional relational database management should be altered for a database completely implemented in PCM. Specifically, we study this effect by choosing hash-join algorithm.

The experiments are carried out in a simulated environment, by modifying a DRAM to act as a PCM. We use postgreSQL database for relational database experiment. The results show that PCM has many benefits in the current memory hierarchy. First, if it is used in a small scale, it can be used as a buffer for flash to improve its write efficiency. Then, if PCM were to replace the DRAM as main memory, we can modify the traditional database algorithms marginally to accommodate the new PCM-based database.

Acknowledgement

I owe my deepest gratitude to people around me without whose help and support I would not have been able to finish my thesis.

First of all, I would like to thank my principal supervisor Prof. Y.C. Tay for his continuous support, help and patience with me. I would also like to specially thank my co-supervisor from Data Storage Institute of A-Star, Dr. Wei Qingsong for his kind guidance and support throughout my study. It was a pleasure to work with him and learn valuable knowledge from him.

I would like to thank my colleague and one of my best friends Gong Bozhao for his support during my initial stage of research.

I would also like to thank my dearest parents who have endured their son being away from them for most of the time but have supported me in my every life decisions.

Last but not the least, I would like to thank all the supervisors involved in the evaluation of this thesis. For any errors or inadequacies that may remain in this work, of course, the responsibility is entirely my own.

Contents

Abstract **i**

List of Tables **vi**

List of Figures **vii**

1 Introduction **1**

1.1 Our contribution 4

2 Phase Change Memory Technology **7**

2.1 PCM in Memory Hierarchy 8

2.2 Related work on PCM-based database 11

2.2.1 PCM as a secondary storage 11

2.2.2 PCM as a Main Memory 12

2.2.3 B+-tree design 12

2.2.4 Hash-join 13

2.2.5 Star Schema Benchmark 13

2.3 PCM: Opportunity and Challenges 15

3 PCM as a buffer for flash **17**

3.1 Flash SSD Technology: FTL and Buffer Management 17

3.1.1 Flash Translation Layer 18

3.1.2 SSD buffer management 19

3.1.3 Duplicate writes present on workloads 20

3.2 System Design 22
3.2.1 Overview 23
3.2.2 Redundant Write Finder 24
Fingerprint Store 24
Bidirectional Mapping 25
3.2.3 Writing frequent updates on PCM cell 26
F-Block to P-Block Mapping 29
Relative Address 29
Replacement Policy 29
3.2.4 Merging Technology 30
3.2.5 Endurance, Performance and Meta-data Management . . 31

4 Impact of PCM on database algorithms 33
4.1 PCM based hash join Algorithms 33
4.1.1 Algorithm Analysis Parameters 33
4.1.2 Row-stored Database 34
4.1.3 Column-stored Database 35

5 Experimental Evaluation 38
5.1 PCM as flash-Buffer 38
5.1.1 Experiment Setup 38
Simulators 38
Simulation of PCM Wear out 39
Simulation parameter Configurations 39
Workloads and Trace Collection 40
5.1.2 Results 41
Efficiency of duplication finder 41
Performance of flash buffer management 44
Making Sequential Flushes to flash 47
Combining all together 47
5.2 Hash-join algorithm in PCM-based Database 49

5.2.1 Simulation Parameters 51
5.2.2 Modified Hash-join for Row-stored and Column-stored Database 52
5.2.3 PCM as a Main Memory Extension 56

6 Conclusion **59**

Bibliography **60**

List of Tables

2.1 Performance and Density comparison of different Memory devices 10

2.2 Comparison of flash SSD and PCM 10

4.1 Terms used in analyzing hash join 34

5.1 Configurations of SSD simulator 40

5.2 Configuration of TPC-C Benchmarks for our experiment 40

5.3 Simulation Parameters 51

List of Figures

2.1 Position of PCM in Memory Hierarchy 8

2.2 Memory organization with PCM 9

2.3 Schema of the SSBM Benchmark 14

3.1 The percentage of redundant data in (a) Data disk; (b) Workload , cited from [14] 21

3.2 Illustration of System design 23

3.3 Basic Layout of the proposed buffer management scheme 27

3.4 Illustration of replacement policy 30

3.5 Illustration of Merging and Flushing block after replacement . . 31

5.1 The duplication data present in the workloads 42

5.2 The effect of fingerprint store size on (a) Search time per fingerprint; (b) Duplication detection rate 43

5.3 flash space saved by duplicate finder 44

5.4 The impact of data buffer size on write operations 45

5.5 The comparison of (a) Merge Numbers; (b) Erase Numbers; and (c) Write time for three techniques 46

5.6 The comparison of Energy consumption for (a) Write operation; (b) Read Operation; (c) Write + Read 48

5.7 Percent of sequential flush to flash due to PCM-based buffer management 49

5.8 Effect of duplication finder and pcm-based buffer extender on (a)Write Efficiency; (b) Lifetime; (c) Power save 50
5.9 Hash join Performance for various Database Size 54
5.10 Comparison of traditional and modified hash joins for R-S and C-S databases by increasing user size from 20 (U20) to 200(U200) 55
5.11 hash join Performance for a PCM-as-a-Main-Memory-Database . 57

Chapter 1

Introduction

Non-volatile Memory (NVM) has a day-to-day impact in our life. NVM known as flash memory is there with us to store music on our smart phone, photographs on cameras, documents we carry on USB thumb drives, and as the electronics in cars.

Phase Change Memory (PCM) [25] is one of such emerging NVM that has many attractive features over traditional hard disks and flash SSD. For example PCM read is more than ten times faster than flash Solid State Disks(SSD), and more than hundred times faster than hard disks, while PCM write is also faster than both flash SSD and hard disks. Besides PCM supports '*in-memory update*'. And the most important features of all of them is the minimum cell density [41]. These attractive features make PCM a potential candidate to replace flash and hard disks as the primary storage in small and large scale computers and data centres. Besides, since the reads in PCM are almost comparable to that of DRAM, it is not too late to think that eventually we may have a computer with PCM as the only memory, replacing both hard disks and DRAM[34].

Despite the above positive features, PCM is relatively slow in hitting the memory world by storm, mainly, because of its two main drawbacks. The writes are relatively slow compared to reads, and specifically 100 times slower than that

of DRAM [28]. And writes consume more energy, and causes wear-out of PCM cells. Over a lifetime of PCM, each cell can only be used for a limited number of times [29].

In the memory hierarchy, PCM falls in between flash SSD and DRAM main memory. As such, PCM could be a potential bridge between SSD and DRAM memory.

SSDs are gaining huge popularity as of late mainly because of their advantages over traditional hard disks, like faster read access, higher cell density and lower power consumption. Despite all these advantages, flash memory has not been able to completely take over the hard disks as a primary storage media in data centres because of their poor write performance and lifespan [9].

Even though SSD manufacturers claim that SSDs can sustain normal use for few to many years, there still exist three main technical concerns that inhibit data centers to use SSDs as the primary storage media. First concern is, as bit-density increases, flash memory chips become cheaper, but their reliability also decreases. In the last two years, for high-density flash memory, erase cycle number decreased from ten thousand to five thousand [7]. This could get even worse as the scaling goes up. Second concern is traditional redundancy solutions like RAID, which are effective in handling hard disk failures, are considered less effective for SSDS, because of the high probability of correlated device failures in SSD-based RAID [8]. The third concern is prior research on lifespan of flash memories and USB flash drives has shown both positive and negative reports [11, 22, 36]. And a recent Google report points out that endurance and retention of SSDs is yet to be proven [9].

Flash memory suffers from a random write issue when applied in enterprise environments where writes are frequent because of its 'erase-before-write' limitation. Because of this, it cannot update the data by directly overwriting it [24, 5]. While PCM has not this issue since it allows 'in-place-update' of data,

PCM also has a finite write lifetime like the flash memory.

In flash memory, read and write operations are performed in a granularity of a page (typically 512 Bytes to 8 KB) [17]. But to update a page, the old page has to be erased, and to make matters worse, erase cannot be performed on a single page. Rather, a whole block (erase unit) has to be erased to do the update.

Some file systems called 'log-based file system' have been proposed to use logging to allow 'out-of-place-updating' for flash [43]. Some research shows that performance of these file system does not fit well for frequent and small random updates, like in database online transactions (OLTP) [10, 32]. Recently, In-Page Logging (IPL) approach was proposed to overcome the issue of frequent and small random updates [32]. It partitions the block of Flash memory into data pages and log pages, and further divide log pages into multiple log sectors. When a data page is updated, the change on this update (the change only, not the whole page) is reflected in the log sector corresponding to this data page. Later when the block runs out of memory, the log sectors and data pages are merged together to form an up-to-date data page.

Although IPL succeeds in limiting the number of erase and write operations, it cannot change the fact that the log region is still stored inside the flash, which has inherent limitations like no in-place-update, frequent updates of log regions, etc.

In PCM, the minimum write units are at byte-level, that means they can be written at more than 10 times finer granularity than the flash disk [45]. Furthermore, PCM allows the in-place-update of the data. Thus it is not that difficult to think that PCM may be used as a buffer for flash SSD.

By exploiting the advantages of PCM, a d-PRAM (d-Phase Change Random

Access Memory) technique was proposed where the log-region that was kept in flash is now kept in PCM [44]. This solves the issues of IPL, but it still cannot take full advantage of PCM technology. It has been well documented that flash performs poorly for random writes [5]. By properly managing the log region of PCM (or PCM buffer region), we can promise that every merge operation will invoke a sequential write flush to the flash.

1.1 Our contribution

This thesis mainly focuses on two contributions of PCM: using PCM as a SSD-buffer to increase its write efficiency, and impact of PCM on relational database.

As the first and main contribution , an encryption-based method to find and remove redundant data from SSD is purposed. PCM is used as a log to store smaller writes to flash because of it has the highest cell density among the emerging memory technologies [30]. Then, capacity of PCM is good enough to qualify as the buffer of flash that works as a massive storage [29]. Also previous works on combining PCM and flash [37, 26] to form hybrid storage have already shown that combining these two is feasible.

The main contributions of the first part can be summarized as:

- Since normal workloads all contain significant redundant data, we propose a hash-based encryption method to identify the redundant data that is headed to be written on flash pages, and maintain the finder in PCM.
- Considering the in-page update property of PCM, we propose the use of PCM as an extended buffer for flash memory.
- We emulate the PCM log region like the internal structure of flash memory, with blocks and log-sectors. Because of this, when the logs are merged

with data pages of flash, a sequential flush is carried out to the flash. This help increase the write performance of flash memory.

- We propose a replacement policy based on block popularity of PCM to ensure that the PCM log region wears out evenly.
- We modify the Microsoft SSD simulator extension [6] to include duplication checking mechanism. This SSD simulator is an extension of widely-used Disk simulator Disksim [12], and implements the major components of flash memory like FTL, mapping, garbage collection and wear-leveling policies, and others. The current version does not have buffer extension for flash, which we implemented. So when a new write request comes to SSD, it is first brought into this flash buffer space, and when its operation is completed, the host is notified of it.
- We also implement the two log-based buffer management techniques, namely IPL [32] and dPRAM [44] to compare our buffer management scheme against these.
- To include the PCM simulator, we wrote our own PCM simulator using C++, and implemented it as an extension of Disksim just like the SSD simulator. We implement a fingerprint store, F-block to P-block mapping table and a PCM log region as explained in above sections.

In the second part of the thesis, we ask the question: if PCM is to replace the entire primary and secondary storage, how a database system should be optimized for PCM. Primary design goal of new database algorithms should be minimizing the number of writes, and the writes should be evenly distributed over the PCM cells. Specifically, a modified hash-join Algorithm PCM-based database system is proposed.

Recent work has shown than column-stored database perform better for read-intensive queries [4] than the row-stored database. Even though, it is nor-

mally up to the database vendor to choose which type of database to use for their system, we do a comparative study of using PCM as a column-stored and row-stored database. We propose modified hash-join algorithms for these database systems and compare them with the traditional hash-join for column-stored and row-stored database systems.

Besides that, we also consider how database algorithms should be modified if PCM is used as a main memory extension, instead of secondary memory. We propose a modified hash-join algorithm for this database as well. All these hash-join algorithms re-organize the data structure for joins, and trade off an increase in PCM reads by reducing PCM writes.

We measure the performance of these algorithms in terms of their impact on PCM Wear, PCM Energy, and Access Latency. We propose analytic metrics for measuring these parameters.

We use DRAM as an emulator for PCM. To emulate DRAM as a PCM, we change the read write time, and emulate the wear out behavior of PCM by introducing a counter on the DRAM cells that get written. We study PCM as a faster hard-disk as well as a DRAM extension. Simulation configurations for these two architectures are different. For PCM as a faster hard-disk, data would be required to brought into a DRAM to complete read or write, whereas in its use as a DRAM extension we suppose that data from PCM do not need to be brought into the DRAM to complete read/write operation. The experimental results show that the proposed new algorithms for hash-join significantly outperform traditional approaches in terms of time, energy and endurance (Section 4), supporting our analytical results. Moreover, experiment on multi-user environment shows that the results hold for a large database system with many transactions at the same time.

Chapter 2

Phase Change Memory Technology

Phase-Change memory (PCM) is a type of the next-generation storage-class memories (SCM) or Non-volatile Memories (NVM). PCM has read latency close to DRAM and high write endurance which makes it a promising technology for building large scale main memory system provided that one day it can have higher density than DRAM. The chalcogenide-based material used in making PCM allows it to switch between two states, amorphous and polycrystalline, by applying electrical pulses which control local heat generation inside a PCM cell [41].

Different from conventional RAM technologies, the information carrier in PCM is chalcogenide-based materials, such as $Ge_2Sb_2Te_5$ and $Ge_2Sb_2Te_4$ [25]. PCM exploits the property of these chalcogenide glasses which allows it to switch the material between two states, amorphous and polycrystalline, by applying electrical pulses which control local heat generation inside a PCM cell. Different heat-time profiles can be used to switch from one phase to another. The amorphous phase is characterized by high electrical resistivity, whereas the poly-

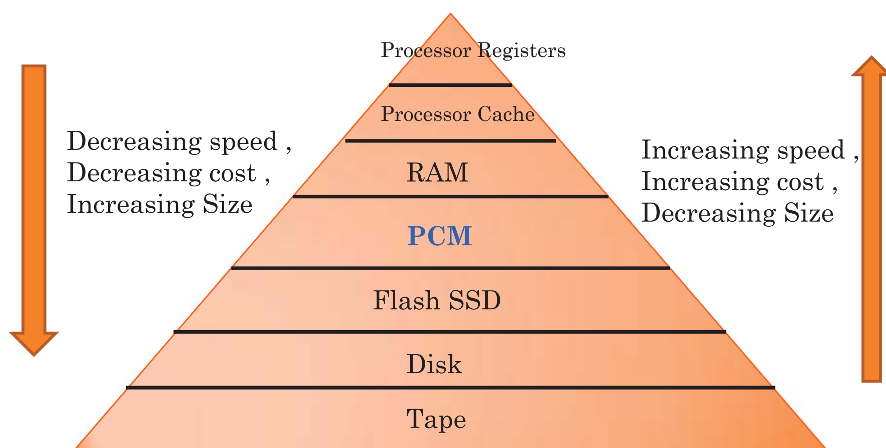


Figure 2.1: Position of PCM in Memory Hierarchy

crystalline phase exhibits low resistivity. The difference in resistivity between the two states can be 3 to 4 orders of magnitude [41].

2.1 PCM in Memory Hierarchy

PCM is a byte-addressable memory that has many features similar to that of DRAM except the life-time limitation [25]. In today's memory PCM falls in between DRAM and flash SSD in terms of read/write latency. Figure 2.1 shows the memory hierarchy.

Compared to DRAM, PCM's read latency is close to that of DRAM, while write latency is an order of magnitude slower. But PCM has a density advantage over DRAM. Also PCM is potentially cheaper, and more energy-efficient than DRAM in idle mode.

Compared to flash SSD, PCM can be programmed in any state, i.e. it supports the '*in-page update*' , and does not have the expensive '*erase*' operation that flash SSD has [33]. PCM has higher sequential and random read speed

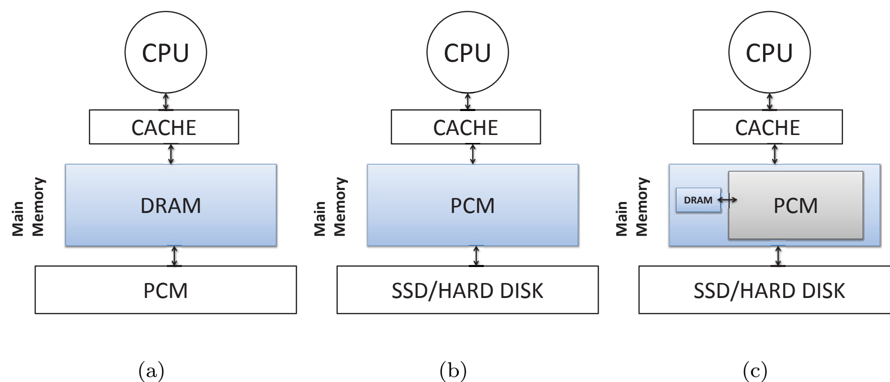


Figure 2.2: Memory organization with PCM

than SSD. And PCM's write endurance is also better.

Figure 2.2 shows three ways in which PCM can be incorporated in memory system [31, 39]. Proposal (a) uses PCM just as a plane replacement of SSD and hard disks. Proposal (b) replaces DRAM with PCM to achiever higher main memory capacity. Even though PCM is slower than DRAM, execution time on PCM can be reduced with clever optimizations.

Proposal (c) includes a small amount of DRAM in addition to PCM so that frequently accessed data can be kept in the DRAM buffer to improve performance and reduce PCM wear. It has been shown that a relatively small DRAM buffer (3% the size of PCM) can bridge the latency gap between DRAM and PCM[39].

As PCM technology evolves, it has shown more potential to replace NAND flash memory with advantages of in-place updates, fast read/write access, etc. Table 2.1 compares the performance and density characteristics of DRAM, PCM, NAND flash memory and hard disks. Table 2.2 compares the read/write characteristics of Flash SSD and PCM. Units of write and read operations for flash and PCM are different. While flash is written or read in units of page, PCM can be accessed in finer granularity (byte-based). This advantage makes PCM a viable option, in compared to traditional IPL [32] method, to use as a

log region to store the updated contents of Flash.

Currently, it is still not feasible to replace the whole NAND flash memory with PCM due to its high cost, limitation of manufacture and data density [28, 29]. Thus we propose to use PCM as an extension of buffer for flash. We manage the log region of PCM in such a way that it emulates the structure of flash memory. Specifically, we divide the PCM into a $n * m$ sized array of log sectors, where, n represents the block number (*P-Block*) and m represents the log sector number. Here, using a DRAM to as a log region instead of PCM does not make sense as DRAM is volatile, and the writes in the log region are supposed to be there as long as their parent block in flash needs them.

Attributes	DRAM	PCM	SSD	Hard Disk
Non-volatile	No	Yes	Yes	Yes
Idle Power	$100mW/GB$	$1mW/GB$	$10mW/GB$	$10W/TB$
Erase	No	No	Yes	No
Page Size	64 bytes	64 bytes	256 KB	512 bytes
Write Bandwidth	1GB/s	50-100MB/s	5-40MB/s	200MB/s
Page Write Latency	20-50 ns	1 us	500 us	5 ms
Page Read Latency	20-50 ns	50 ns	25 $/muS$	5 ms
Endurance	Infinity	10^6-10^8	10^5-10^4	Infinity
Maximum Density	4 Gbit	4 Gbit	64 Gbit	2 Tbyte

Table 2.1: Performance and Density comparison of different Memory devices

-	flash SSD	PCM
Cell size	$4F^2$	$4F^2$
Write Cycles	10^5	10^8
Read Time	$284\mu s/4KB$	$80ns/word$
Write Time	$1833\mu s/4KB$	$10\mu s/word$
Erase Time	$> 20ms/Unit$	N/A
Read Energy	$9.5\mu J/4KB$	$0.05nJ/word$
Write Energy	$76.1\mu J/4KB$	$0.094nJ/word$
Erase Energy	$16.5\mu J/4KB$	N/A

Table 2.2: Comparison of flash SSD and PCM

As we explained PCM has so many benefits we can say that it is just a matter of time before most of the data centres and database systems start using PCM as the main memory storage device. In the next chapter, we study the use

of PCM in the database management system. How some vendors have already started to optimize the database algorithms for PCM-based database. Then in next chapter, we talk briefly about how PCM's unique properties like faster read access, byte-writ-ability could be taken advantage of to actually improve the write efficiency of solid state devices.

2.2 Related work on PCM-based database

Since PCM is still in its early development phase, and a PCM product with significant size is still not out in the market, most of the studies on PCM-based database are based on emulating PCM using either a DRAM or by using a programmed simulator. Some researchers in Intel [15] have recently studied how some of the database algorithms should be optimized for PCM-based database. They propose optimization algorithms for B^+-Tree and hash-join. The algorithms tend to minimize the writes to PCM by trading off writes with reads.
In this thesis, we propose two modified hash-joined algorithms for PCM-based database when the PCM database is row-stored and column-stored respectively. When PCM is used as a main memory, like the way proposed in [15] paper, we can get a concept of how to design the database from "In-Memory Database"[21].

2.2.1 PCM as a secondary storage

When we use PCM as a secondary storage like SSD and hard disk, database algorithms proposed for such devices cannot fully exploit the advantages of PCM over such devices. For example random reads are almost as fast as sequential reads in PCM [27], so optimization for random writes are redundant for PCM. Similarly, PCM cell has a lifetime issue, so before writing data to PCM, we must consider if the writes are concentrated in only certain region of the PCM. Because once these few writes become unusable, whole PCM becomes less efficient. And in general, writes are expensive, consume more energy, and take more time.

Thus optimization is done on database algorithms to minimize write numbers, and if required, trade off reducing writes with increased number of reads.

2.2.2 PCM as a Main Memory

Similarly, when PCM is used as a main memory, the concept of in-memory database[21] cannot also be directly implemented in it. For one, it cannot be frequently written like DRAM. Recent studies have shown that PCM can be used as a large main memory while a small DRAM can be used to support the frequent writes towards the main memory. By combining a DRAM of size only about 3% the size of PCM can achieve significant performance boost [39]. In our experiment, we do consider PCM as the main component of main memory but also have a small amount of DRAM to handle frequent updates.

2.2.3 B+-tree design

A B+-tree is a type of tree which represents sorted data in a way that allows for efficient insertion, retrieval and removal of records, which are identified by a key. It is a multi-level index, dynamic tree with maximum and minimum bounds on the number of keys in each index segment (known as node). In B+-tree all the records are stored at the leaf level of the tree, the interior nodes store only the keys.

How the traditional B+-tree design should be optimized for PCM-based database is an interesting topic. Traditional B+-tree involves a number of split and merge operation, which means frequent writes to the database medium. Thus design of B+-tree for PCM should be focused on reducing the number of writes, i.e. reducing the number of splits and merge operation. Chen et. al from IBM [15] have done a brief study on possible optimization of B+-tree and hash-join for PCM-based database.

Their proposed B+-tree optimization is basically allowing the leaf nodes of a B+-tree to have keys in unsorted order. Then leaving one key field to contain the bit-map of the content of the nodes. This way insertion for a key will only

need to refer the bit-map and find an empty location. Deletion will need to modify the bit-map only.

2.2.4 Hash-join

Since grace hash-join or even the hybrid hash-join require a relation be split into smaller partitions based on the matching hash-keys, then re-writing these small partitions back into the storage medium, one way of reducing the frequent writes could be avoiding the re-writing part. A method called 'virtual partitioning' is proposed in [15]. Basically, the concept is partition the relation virtually, and instead of re-writing the partitions again, just re-writing an identifier of that record (record id) in the storage medium.

2.2.5 Star Schema Benchmark

In this thesis, we use the Star Schema Benchmark(SSBM)[16]to compare the performance of column-stored and row-stored databases.
SSBM is a data warehousing benchmark derived from TPC-H[3]. Star Schema is simple for users to write, and easier for databases to process. Queries are written with simple inner joins between the facts and a small number of dimensions. These are simpler and have fewer queries than TPC-H.
Schema: The bechmark consists of one fact table, the LINE-ORDER table, a 17-column table with information about individual orders, with a composite primary key of the ORDERKEY and LINENUMBER attributes. Other attributes include foreign key references to the CUSTOMER, PART, SUPPLIER, and DATE tables as well as attributes of each order, priority, quantity, price, and discount. Figure 2.3 shows the schema of the tables.

Queries: We use the following queries for our experiments:

1. Query 1: List the customer country, supplier country, and order quantity for orders made by customer who lives in Asia, for products supplied by

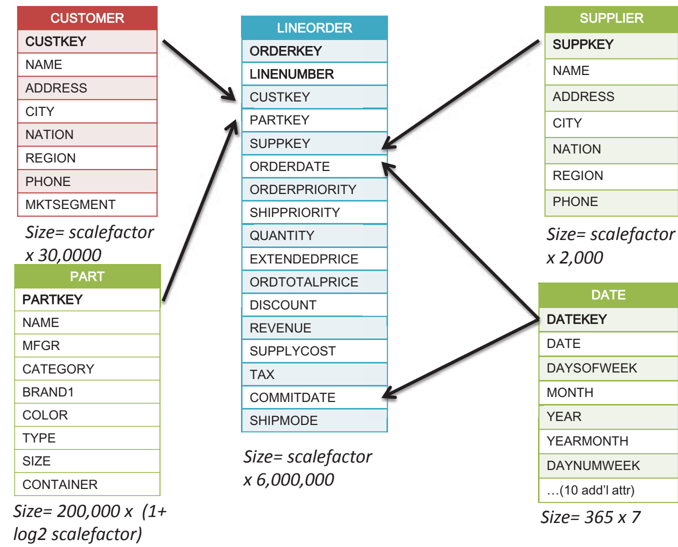


Figure 2.3: Schema of the SSBM Benchmark

an Asian supplier in the year '2009'.

```
SELECT c.nation , s.nation , d.year , lo.quantity
        FROM customer AS c , lineorder AS lo ,
      supplier AS s , dwdate AS d
        WHERE lo.custkey = c.custkey
        AND lo.suppkey = s.suppkey
        AND lo.orderdate = d.datekey
        AND c.region = ''ASIA''
        AND s.region = ''ASIA''
        AND d.year = 2009
```

2. Query 2: List the customer country and order quantity for orders of part type 'IC'.

```
SELECT c.nation , lo.qty
        FROM customer AS c , Lineorder AS lo
        WHERE c.custkey = l.custkey
        AND     p.parttype = ''IC''
        AND   l.partkey  = p.partkey
```

3. Query 3: List the supplier's name and region whose orders are above 500.

```
SELECT s.region , s.name, lo.qty
```

```
FROM supplier AS s, lineorder AS lo
WHERE s.suppkey = l.suppkey
AND lo.qty>500
```

Each of these queries involve a number of hash-join operations between relations. We run these queries in a multi-user environment. As all the database tables are kept in a single PCM device, there will be a fight between transactions over buffer space, and priority of access of data.

2.3 PCM: Opportunity and Challenges

PCM poses a great potential to replace both the primary storage device (main memory) and secondary storage device. Because of its low density, these devices could be very small in volume but have a huge memory space. And PCM's reads are already comparable to that of DRAM, the current choice for main memory. The two main concerns, however, for PCM are : slow writes (compared to DRAM), and limited lifetime. Besides these, error in PCM cells due to temperature change is another concern for PCM.
As such, PCM is important for the following reason:

- As multi-cores and CPU speed increase, so does the gulf between processor and storage speed, PCM narrows the distance from CPU to large data sets by 100X over SSD (high bandwidth).
- PCM increases the data available to CPU by 10X over DRAM (high density).
- PCM decreases the number of servers required to store a fixed set of data
- It allows us to
  - Put all the data into one single storage medium, i.e. PCM and get rid of hard disks as well as DRAM.

– Read the data only when we need it (because PCM is bit-alterable like DRAM).

– Not let the operating system get in our way as the PCM can be used the same way regardless of the operating system.

Chapter 3

PCM as a buffer for flash

In this chapter, we first introduce the flash SSD technology, its status in memory hierarchy and challenges in its development. Then we propose an idea of using PCM as a buffer for flash memory. By exploiting the faster read access, and byte-writeable nature of PCM, a combination of flash and PCM can improve the overall performance of flash SSD by a significant amount. System design, technical details, and analysis of how the system can help improve the SSD efficiency are included in this chapter.

3.1 Flash SSD Technology: FTL and Buffer Management

A flash memory package is usually composed of one or more *dies*. Each die is divided into multiple *planes*. A plane contains number of *blocks*. A block is the erase unit of flash. Each block is further divided into number of *pages*, normally 64 - 128 pages. Each page has a *data area* (normally $4KB$), and a *spare area* for storing meta-data [6]. Three basic operations on flash memory are *read*, *write*(update) and *erase*. Read and write are carried out in units of pages, whereas, erase operation is performed in units of block. An erase opera-

tion clears all the pages in that block [40].

Even though writes on flash are close to or in some cases better than that of hard disks, flash disks suffer from one fatal issue of limited lifespan. Over the lifetime of flash memory, it can only be written for a certain number of times. Hence many researches focus on wear-leveling techniques to wear out the flash evenly, or techniques to improve the lifetime of flash by reducing write traffic to flash.

Overall, three critical technical constraints on flash memory are : (1) No in-place overwrite - the whole erase block must be erased before updating a page in flash. (2) No random writes - in each erase block, the writes must be carried out sequentially. If the write is random, flash memory suffers from poor performance. (3) Limited erase cycles - like cited before, an erase block can wear out after a certain number of erases.

3.1.1 Flash Translation Layer

Because of the erase-before-write characteristics of flash memory, a software layer called flash Translation Layer (FTL) is implemented in flash SSD controller to emulate a hard disk drive by exposing an array of logical block addresses (LBAs) to the host. At the core an FTL uses a logical-to-physical address mapping table. If a physical address location mapped from a logical address contains previously written data, the input data is written to an empty physical location where no data were previously written. The mapping table is then updated due to the newly changed logical/physical address mapping. This protects one block from being erased by an overwrite operation [19].

Generally an FTL scheme can be classified into three groups depending on the granularity of address mapping: *page-level*, *block-level*, and *hybrid-level* FTL

schemes [19]. In the page-level FTL scheme, a logical page number (LPN) is mapped to a physical page number (PPN) in flash memory. This mapping technique has great garbage collection efficiency, but it commands a large RAM space to store the mapping table. *Garbage collector*(GC) is launched periodically to recycle invalidate physical pages, by copying the valid pages in a clean block, and erasing the old block. On the other hand, a block-level FTL is space efficient, but still requires an expensive read-modify-write operation when writing only part of a block. In order to overcome these disadvantages, the hybrid-level FTL scheme was proposed. Hybrid-level FTL uses a block-level mapping to handle most data blocks and a page-level mapping to handle a small set of log blocks, which actually works as a buffer to writes [23]. They are efficient both from garbage collection as well as the size of mapping table points of view.

Besides this general mapping scheme, some log-like write mechanism have also been proposed. Each write to a logical page invalidates the original flash page, and the new content is appended sequentially to a new block, like a log. The idea is similar to log-structured file systems. In-page Logging [32] and Hybrid-logging [44] are two of such examples where log-region is maintained in flash memory and phase-change memory respectively.

3.1.2 SSD buffer management

Many SSD controllers use a part of RAM as read buffer or write buffer. Different buffer cache management policies are proposed to improve performance and extend lifetime of flash memory. Both cache hit ratio and sequentiality are two critical factors determining the efficiency of buffer management for flash memory.

One problem of SSD is that the background garbage collection and wear-leveling compete for internal resources with the foreground user accesses. If most foreground user accesses can be hit in buffer cache, the influence of each other will

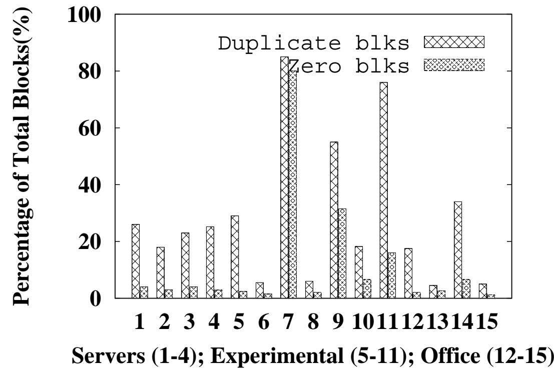
be significantly reduced. In addition, high cache hit ratio significantly reduces the direct accesses from/to flash memory which achieves low latency for foreground user accesses and saves resources for background tasks.

On the other hand, sequentiality of write accesses passed to flash memory is critical because *random write* has following negative impacts on SSD.

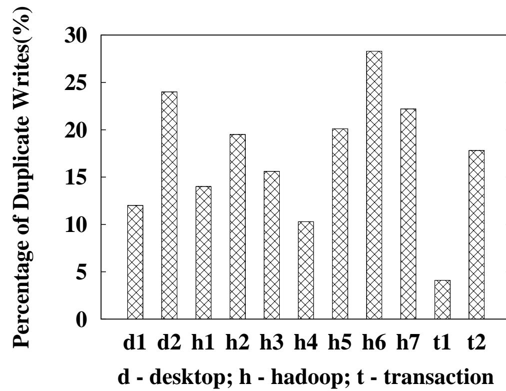
- Shorten the lifetime of SSD: The more the random write, the more the erase operations required, which means SSD lifetime will degrade significantly.
- High Garbage Collection overhead: The random writes means writes will be distributed all over the flash blocks, so during the merge phase garbage collection need to be run on all those blocks [23].
- Internal Fragmentation: Since flash memory does not support in-page update, after certain number of random writes, invalid pages will be distributed all over the blocks, causing internal fragmentation [13].
- Little chance for performance optimization: SSD leverages striping and interleaving to improve performance based on sequential locality [6, 40]. If a write is sequential, the data can be striped and written across different dies or planes in parallel. Interleaving is used to hide the latency of costly operations. Single multi-page read or write can be efficiently interleaved, while multiple single-page reads or writes can only be conducted in separate way. While above optimizations can dramatically improve performance for workload with more sequential locality, its ability to deal with random write is very limited because less sequential locality is left to exploit.

3.1.3 Duplicate writes present on workloads

Data duplication is a common phenomenon in file systems. For example some software developers have multiple versions of source code with only a slight vari-



(a)



(b)

Figure 3.1: The percentage of redundant data in (a) Data disk; (b) Workload , cited from [14]

ation. Users in operating system can create/delete the same file many times. When editing word documents, the editor tools often saves a copy of the document every few minutes, whose content are almost identical.

A study by Feng et. al [14] shows that duplicate blocks in disks used for database/web servers, office systems and experimental systems is very common. Figure 3.1(a) shows the duplication rates (percentage of duplicate blocks in total blocks) in a study of 15 disks. The duplication rate ranges from 7.9%

to 85.9% across the 15 disks. Similarly, Figure 3.1(b) shows the percentage of duplicate writes in 11 different workloads from three categories. It is found that 5.8-28.1% of the writes are duplicated. These findings suggest that by removing these redundant writes, we can effectively reduce write traffic to flash, and subsequently improve its endurance and write efficiency.

3.2 System Design

The system contains the following main design steps.

- *Redundant data finder*: First, we have to find if the write is a redundant write, i.e. it already exists on the flash page. If it is a redundant write, then we can avoid re-writing it.
- *Accelerating the redundant data finding mechanism*: For the better performance of the system, it is important that the redundant data finding mechanism does not become the bottleneck. We propose a couple of accelerating mechanism for quick searching of presence of redundant data.
- *Using PCM for log update of flash pages*: We would like to exploit the in-page update mechanism of PCM and take away all the frequent updates from flash pages into PCM.
- *Merging PCM logs with flash pages*: When the flash pages need to be updated, we bring the logs of PCM and the original flash pages together in DRAM, merge them to form an up-to-date page and flush them sequentially into flash. By this we can exploit the faster write performance of flash for sequential writes.
- *Lifetime and wear-leveling for PCM*: Since PCM blocks can only be written for certain amount of times, we design a wear-leveling mechanism to make sure the PCM blocks wear out evenly over time. This prolongs the useable lifetime of the PCM.

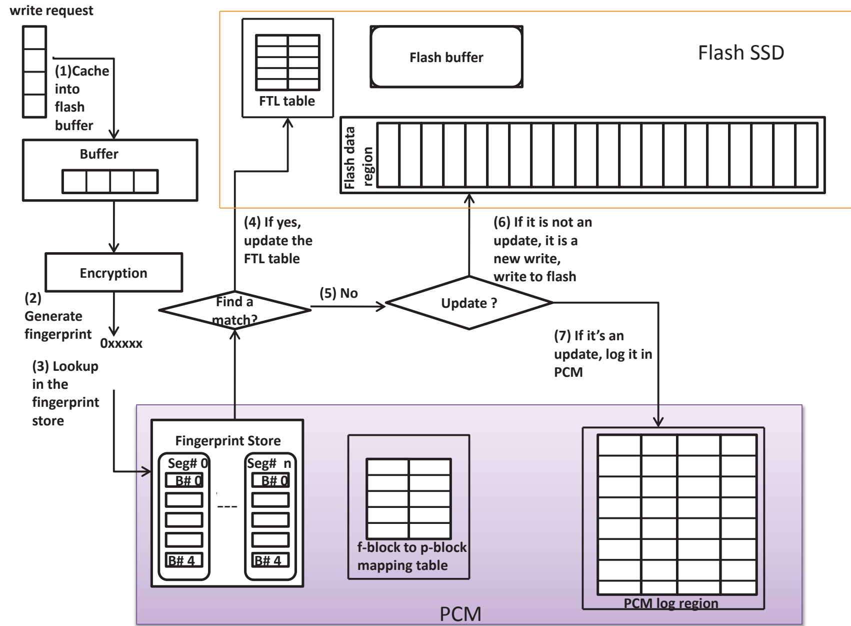


Figure 3.2: Illustration of System design

3.2.1 Overview

The goal of our design is to reduce unnecessary write traffic, increase the lifetime of flash, extend the available flash space and improve the overall write performance. Even though the space allocated for storing fingerprints is large enough, we cannot always guarantee to find the duplicate data and remove them immediately. Figure 3.2 illustrates the process of handling a write request in our design.

When a write request arrives at SSD, (1) the data is brought into SSD buffer; (2) this page in buffer is encrypted to find a fingerprint. The encrypt handler could be a dedicated processor or just a part of the controller logic; (3) the fingerprint is looked up in fingerprint store which is kept in PCM, which maintains the fingerprints of data already stored in flash; (4) if a match is found, it means the write already exists in flash. Thus the FTL mapping table is updated to map this logical page number to the existing physical page number, and correspondingly the write (which would be redundant) is avoided; (5) if no

match is found then there could be two cases; (6) if the write is a new write, not an update, it is written to the physical location it was supposed to be written; (7) but if it is an update, then instead of going back to the flash memory's flash pages, invalidating them and writing in some other flash pages, we calculate the difference between the original page in flash and new update, and store this change into PCM log region.

3.2.2 Redundant Write Finder

An important aspect of our design is in finding and removing the redundant updates that exist in the workload. A byte-by-byte comparison would be unnecessarily slow. A common practice is to use a cryptographic function to encrypt the incoming data and generate a unique identifier. Cryptographic hash function like SHA-1 [20] or MD5 [42] are two of the most popular ones. The SHA-1 hash function has been proven to be computationally infeasible to find two distinct inputs hashing to the same value [35]. We use SHA-1 hash function on the content of each flash page to generate a unique hash value, referred to as fingerprint. We choose flash page as the chunk size to do the encryption, because page (normally 4 KB in size) is the basic operation unit in flash, and the flash internal policies like FTL, are also designed in the units of page. Using these fingerprints, we can safely determine if the contents of two pages are the same.

Fingerprint Store

The fingerprint store is maintained in PCM instead of flash for the simple reason that reading from PCM is several times faster than reading from flash. Each fingerprint store value contains two components, fingerprint value and its physical location. For accelerating the searching process in fingerprint store, we first logically partition the fingerprint store into N segments.N is determined by the size of the PCM. For a given fingerprint f, we can map it to segment $(f \ mod \ N)$. Each segment contains a list of buckets. Each bucket is a 4 KB page in memory,

and contains multiple entries, each of which is a key-value pair of $<fingerprint$, $location>$.

To accelerate the search process,inside each buckets, the fingerprints are stored in their ascending order. When a fingerprint has to be checked in the store, first we calculate the $SegmentNo.$ by using the aforementioned hash function. Since each buckets in a segment is sorted, we do a range check in the bucket. That means we compare the fingerprint with the smallest and largest fingerprints in the bucket. If the fingerprint is out of the range, we pick another bucket and do the range check. Once we find the bucket that satisfies the range check, we do a binary search on that bucket. This way, we avoid the binary search in the entries of each of the bucket. To further accelerate this searching process, we can sort the buckets in the segment, randomly choose a bucket from the middle and do the range check. This way we can skip over most of the buckets and reduce the number of comparisons required.

Bidirectional Mapping

When we are searching for the physical location of a fingerprint, we search in a table that maps fingerprints with physical page numbers. This one-way mapping is not enough to maintain the fingerprint store, where when certain pages are updated or deleted, the fingerprint value and/or the physical location value also need to be changed. Thus we maintain a bidirectional mapping: mapping from fingerprint to physical location and vice-versa.

For example, in page number 1024, a new data is written. Now we need to delete the fingerprint that originally contained in physical number 1024, so by checking in the *location* to $fingerprint$ mapping, we can find the fingerprint and update it. This mapping table is maintained together in the fingerprint store in the PCM. Normally at the time of block erase, and later at merging of PCM log region and flash data region only does this table need to be referenced and updated.

3.2.3 Writing frequent updates on PCM cell

Unlike flash, PCM allows the in-page update. To exploit this advantage, we propose a buffer management method to use PCM as an extension of SSD buffer to drift the frequent updates on flash pages into PCM. The region on flash that stores the data is called data region, and the region on PCM where we store the update logs of flash pages is called log region. We cannot place the PCM-based log region inside the SSD data region due to their differences in processing technologies. Instead of using the SSD-pages themselves as the pages for logical update, we save the update requests to flash pages into PCM region. We want to manage the updates, or new writes to flash pages in the PCM log region in such a way that it is easy to fetch them when they are required. And when the data have to be flushed into flash, we want to make sure that these become as sequential as possible.

The basic layout for this buffer management is shown in Figure 3.3. The buffer of the SSD itself is not modified, existing buffer management technique of this flash is kept as it is. We partition the PCM log-region into multiple blocks, each block containing as many log sectors as the number of pages in a block of the flash SSD. Specifically, we divide the PCM into a $n * m$ sized array of log sectors. Where, n represent the block number (P-Block) and m represent the log sector number. When a flash page is modified, instead of directly going back to the flash block and updating the page, we log the changes made in the log sector of PCM.

Since PCM is byte-writable, we can shrink the write from page-size(KB) to byte when we actually record the update. This can save write time, and consequently save energy. When the log sectors in a block region are all altered, we bring these logs and the pages from the corresponding block of flash together into NAND-buffer, combine them and flush them sequentially into SSD Block. This process is called *merging*. A block number in flash (F-Block) to block

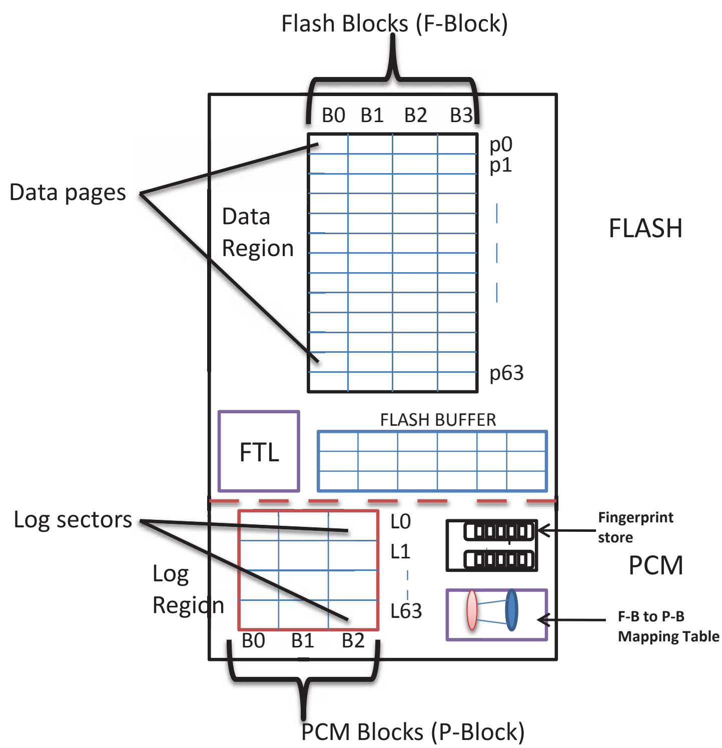


Figure 3.3: Basic Layout of the proposed buffer management scheme

number in PCM (P-Block) mapping table is maintained in PCM. This table need to be modified when the block of a PCM region is flushed and this block now contains logs from different flash block. The access to this PCM-extension architecture is described as follows:

- For a read operation, the address of the accessed data is sent to both the data region of flash page and log region of PCM. The exact log region is located by looking up on the F-Block to P-Block mapping table. If the log region has log records for this page, they are loaded into the data buffer as well as the original data page to create an up-to-date data page.

- For a write operation, there are multiple scenarios:

  - Case 1: The flash data page to which this write points is empty, then it is written directly on the data page. Otherwise we have cases 2-5.
  - Case 2: The flash block number of the page does not exist on the F-Block to P-Block mapping table, then there can be two cases. The PCM blocks are all occupied, or there is an empty block. If all the P-Blocks are occupied, a victim block is chosen by a replacement algorithm, which will be explained in next section. The contents of this block are flushed into flash. A new P-block is allocated for this F-block, and the log sector address for this page is calculated by a simple hash function. Then, the current update is written to this sector.
  - Case 3: If the log record for this page already exists, this log record and the current update are compared. The change is now recorded in the log sector.
  - Case 4: The F-Block to P-Block mapping table exists, but the log sector for this page does not exist. In this case, we calculate the log sector address for this page by the same hash function as above. Then the update is written to this sector.

F-Block to P-Block Mapping

We introduce a term *block popularity*, which is maintained in the F-Block to P-Block mapping table, is the counter that how many times a block in PCM (P-Block) is updated. Initially the *block popularity* of each block is set at 0. Every time a request comes for update a block, the block's *block popularity* is increased by 1.F-Block means the block numbers of flash disk. P-Block means the block numbers that are allocated in PCM to store the logs of data pages in flash. This table is actually an array of $n * 3$ size. Where the first element in a row represent F-Block, second element represent P-Block and the third element represent popularity of P-Block . This table is maintained in PCM. The size of the mapping table is very small (128 KB in our case) since it is just a $n * 3$ integer array. Every read or write operation in flash need a visit to this mapping table to find the log region of the data page. When a P-block is flushed out, or when log regions for new F-block are recorded, this mapping table is updated. The maintenance cost of this table is negligible compared to all other read write accesses.

Relative Address

For in-place update, the data address of the current update is compared to the data address of existing log records. We use a relative address($addr_{Rel}$) to represent the position of an update inside the accessed data page. It can be calculated as $addr_{Rel} = addr_{Update} - addr_{Page}$. $addr_{Update}$ and $addr_{Page}$ represent the addresses of the update and the currently accessed page, respectively.

Replacement Policy

For flash Buffer, we do not need to change the existing replacement policy. This replacement policy is for choosing a victim block from PCM's P-Blocks when the P-Block list is full and logs for new F-Block has to be written. It is preferable to keep those blocks which are frequently accessed, and which have more log sectors that are changed, in PCM. The block with least block popularity is

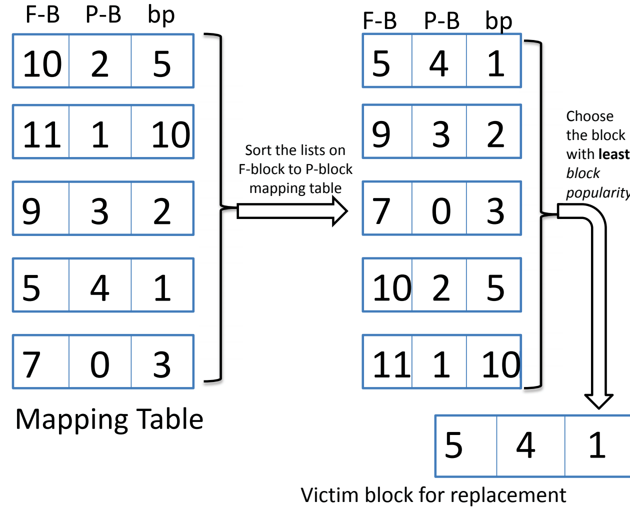


Figure 3.4: Illustration of replacement policy

chosen for replacement. After every log sector of a P-Block is updated, its block popularity is increased by 1. Once the block is flushed out, it is reset to zero.

Figure 3.4 depicts the replacement strategy. By replacing the least popular block to store logs for new block, the write intensities of log sectors is balanced. With this policy, the blocks with free log sectors (i.e. the blocks with less block popularity) will get chance to be written again. This will, in turn, help balance the write intensities among the P-Blocks. This helps the blocks of PCM to wear out evenly. This is why we said that without adopting external wear leveling mechanism, we manage to gain significant wear leveling balance.

3.2.4 Merging Technology

As explained in Figure 3.5, when a P-Block is chosen for replacement, the contents of the log sectors, as well the contents of its corresponding data region in flash are brought into flash Buffer. These two are compared, and merged to form an up-to-date data. Then, the content of that flash blocks are flushed in the flash in sequential fashion. This step completely avoids the traditional requirement where we needed to temporarily copy unchanged pages of the updating

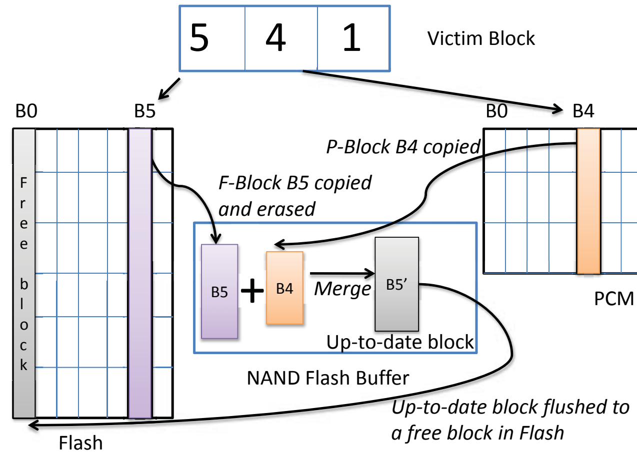


Figure 3.5: Illustration of Merging and Flushing block after replacement

block to another location and bring them back.

3.2.5 Endurance, Performance and Meta-data Management

Endurance of flash disk is an important issue since it has only a limited number of writes. By shifting the in-place updating to PCM architecture, endurance of the storage system can be improved. Write intensity to flash storage is now greatly reduced compared to flash-only storage. Compared to flash, PCM has a better endurance. This makes sure that log region of PCM will not wear out before the data region of flash does. But PCM also has a write lifetime issue, and if the updates are more converged in certain region of PCM, it may wear out faster than the flash disk. But the block popularity replacement policy can help even out the concentration of wear and tear in PCM.

PCM Log Sectors can easily be in-place updated, read faster and merged with the content in their corresponding data region. As explained above, the metadata should record F-block to P-Block mapping which is maintained in PCM.

Reading a log sector incurs extra reads, but it saves expensive writes to flash data

pages. Writes to PCM are faster than to flash, and require less energy. Thus overall, it should be able to improve the system throughput while consuming less energy to do so.

Chapter 4

Impact of PCM on database algorithms

This chapter examines how hash-join algorithms can be modified for PCM-based database systems. Specific algorithms, design steps for both column and row-based databases are proposed.

4.1 PCM based hash join Algorithms

In this section, we discuss the concepts used in calculating the performance measures. Then we write out the traditional and modified hash-join algorithms. Reducing PCM writes is the main goal in all modified hash-join algorithms.

4.1.1 Algorithm Analysis Parameters

One of the significant challenges in designing PCM-friendly algorithms is to cope with the asymmetry of PCM reads and writes. PCM writes are slower than reads, and consume more energy, and wear out PCM cells. Thus primary design goal of new database algorithm (hash-join in this thesis) is to minimize the number of PCM writes.

Since PCM is a byte-writable memory device, for both using PCM as a main memory and as a secondary memory, we consider the write granularity to be *bit*. We compare the performance of six types of hash-join algorithms namely, traditional and modified hash-joins for row-based PCM database, traditional and modified hash-joins for column-based PCM database, and traditional and modified hash-joins for PCM as a main memory database. We analyze the performance of a hash-join algorithm in terms of its effect on PCM-wear, PCM energy and access latency. Table 4.1 shows the terms with meaning used in analyzing hash joins. We use the following formulas to calculate PCM wear, PCM energy, and Access Latency:

1. $\textit{Total Wear} = \kappa N_w$
2. $\textit{Energy} = 8(N_w E_{wb} + N_r E_{rb})$
3. $\textit{Access Latency} = 8(N_w T_w + N_r T_r)$

Term	Description	Value used
N_w	Number of words written to PCM	-
N_r	Number of words read from PCM	-
E_{rb}	Energy consumed for reading a PCM bit	2 pJ
E_{wb}	Energy consumed for writing a PCM bit	16 pJ
T_r	Latency of reading a PCM bit	230 cycles
T_w	Latency of writing a PCM bit	450 cycles
W	Word size used in PCM writes	8 B
κ	Average number of modified bits per modified word	-

Table 4.1: Terms used in analyzing hash join

4.1.2 Row-stored Database

Algorithms 1 and 2 show the traditional [38] and modified hash-join algorithms for row-stored database. In traditional row-based algorithm, during partition phase, all the tuples are copied back to PCM. This write back is unbearably expensive for PCM as this causes PCM to wear out quickly. In modified algorithm, we propose a concept of copying only the tuple IDs that belong to the same partition. During partition phase, the tuples are read in, and while

creating hash-buckets for tuples belonging to same partition, we only copy the $tupleID$ of that tuple. Size of a $tupleID$ is an integer value(4 bytes) which equals to four units of PCM when PCM is written byte-wise.

The size of writes to PCM is significantly reduced. For instance, for a relation with 500,0000 records, the maximum size to be written to PCM using modified hash-join is equal to $500,000 * 4bytes = 0.5MB$. If we do the same copying using traditional hash-based join, the total write size is equal to the size of the relation itself. During join phase, the tuples are read using their tuple ID. Since random reads and sequential reads are almost equally fast for PCM, reading tuples randomly from the relation does not decrease the join processing time.

4.1.3 Column-stored Database

Algorithms 3 and 4, respectively, show the traditional and modified hash-join algorithms for column-stored database. Even though column-stored database is reported to perform better than row-stored database for general workloads [4], the critical point is maintaining column-stored database itself is a tedious job and is much costlier. In column-stored database, all the columns of a relation are stored like a separate relation with two attributes, $tupleID$ and $Value$. Thus queries that involve operation on a single column are obviously faster for column-stored database.

Traditional hash-joins for column-stored database is similar to the traditional row-stored hash-join algorithm, only difference is here instead of partitioning the whole relation, only the column taking part in join is partitioned. Similar to the concept in row-stored database, in modified hash-join for column-stored, only the $tupleID$ of tuples of the column taking part in join are copied back to PCM.

Algorithm 1: Traditional Row-based hash-join

```
Partition Phase
P = Total no. of Partitions
for (i = 0; i < R; i++) do
    r = record i in Relation R
    p = hash(r) modulo P
    copy r to Partition R_p
end
for (j = 0; j < S; j++) do
    s = record j in Relation S
    p = hash(s) modulo P
    copy s to Partition S_p
end
join Phase
for (p = 0; p < P; p++) do
    Build Phase
    for each i in R_p do
        r = record i in R
        insert r into hash table
    end
    Probe Phase
    for each j in S_p do
        s = record j in S
        probe s in hash table
        if there are match(es)
        then
            generate join results
            send results to upper
            layer
        end
    end
end
```

Algorithm 2: Modified Row-based hash-join

```
Virtual Partition Phase
P = Total no. of Partitions
for (i = 0; i < R; i++) do
    r = record i in Relation R
    p = hash(r) modulo P
    copy r.id to List Rlist_p
end
for (j = 0; j < S; j++) do
    s = record j in Relation S
    p = hash(s) modulo P
    copy s.id to List Slist_p
end
join Phase
for (p = 0; p < P; p++) do
    Build Phase
    for each i in Rlist_p do
        r = record i in R
        insert r into hash table
    end
    Probe Phase
    for each j in Slist_p do
        s = record j in S
        probe s in hash table
        if there are match(es)
        then
            generate join results
            send results to upper
            layer
        end
    end
end
```

Algorithm 3: Traditional Column-based hash-join

```
Data: R.A and S.B being the columns taking place in join
Partition Phase
P = Total no. of Partitions
for (i = 0; i < R.A; i++) do
    r = tuple i of R.A
    p = hash(r) modulo P
    copy r to Partition R_p
end
for (j = 0; j < S.B; j++) do
    s = tuple j of S.B
    p = hash(s) modulo P
    copy s to Partition S_p
end
join Phase
for (p = 0; p < P; p++) do
    Build Phase
    for each i in R_p do
        r = tuple i in R.A
        insert r into hash table
    end
    Probe Phase
    for each j in S_p do
        s = tuple j in S.B
        probe s in hash table
        if there are match(es) then
            generate join results
            send results to upper layer
        end
    end
end
```

Algorithm 4: Modified Column-based hash-join

```
Data: R.A and S.B being the columns taking place in join
Virtual Partition Phase
P = Total no. of Partitions
for (i = 0; i < R.A; i++) do
    r = tuple i of R.A
    p = hash(r) modulo P
    copy r.id to List Rlist_p
end
for (j = 0; j < S.B; j++) do
    s = tuple j of S.B
    p = hash(s) modulo P
    copy s.id to List Slist_p
end
join Phase
for (p = 0; p < P; p++) do
    Build Phase
    for each i in Rlist_p do
        r = tuple i in R.A
        insert r into hash table
    end
    Probe Phase
    for each j in Slist_p do
        s = tuple j in S.B
        probe s in hash table
        if there are match(es) then
            generate join results
            send results to upper layer
        end
    end
end
```

Chapter 5

Experimental Evaluation

We use two different experiment setup for two experiments. First section talks about the experimental setup, and experimental result of PCM as flash-buffer. Experimental analysis of hash-join algorithm is studied in second section. The second part includes simulation parameters, introduction of Star-Schema Benchmark used for the experiment, and analysis of the results obtained.

5.1 PCM as flash-Buffer

5.1.1 Experiment Setup

We implement and evaluate the design of our system using comprehensive trace-driven simulations. In this section, we will introduce our experiment setup, SSD and PCM simulators, workload and trace collection, and system configurations.

Simulators

We modify the Microsoft SSD simulator extension [6] to include duplication checking mechanism. This SSD simulator is an extension of widely-used Disk simulator Disksim [12], and implements the major components of flash memory like FTL, mapping, garbage collection and wear-leveling policies, and others. The current version does not have buffer extension for flash, which we imple-

mented. So when a new write request comes to SSD, it is first brought into this flash buffer space, and when its operation is completed, the host is notified of it.

Besides these, we also implement the two log-based buffer management techniques, namely IPL [32] and dPRAM [44] to compare our buffer management scheme against these.

To include the PCM simulator, we wrote our own PCM simulator using C++, and implemented it as an extension of Disksim just like the SSD simulator. We implement a fingerprint store, F-block to P-block mapping table and a PCM log region as explained in above sections.

Simulation of PCM Wear out

Calculating PCM wear out by simulation is a bit tricky, and in some sense a bit improbable. We need to know how often each unit of PCM cell is being used for read and write. In other words, this requires a statistics of number of reads and writes on a PCM cell. At the beginning of experiment, we set counter of read and write of each PCM cell to be zero. As it is written, or read, the write-counter and read-counter are increased by one. This way, we approximate the number of wear out on each PCM cell.

Simulation parameter Configurations

Table 5.1 gives the list of major configuration parameters for SSD. The page size of flash, erase unit (block) size of flash, and the log sector size of PCM are set to be 4 KB, 256 KB, and 512 Bytes, respectively. As a block in flash contain 64 pages, a P-block also contains 64 log sectors. Hence the size of a P-block is set to be 32 KB. Besides these, a space of 128 KB is allocated to store F-Block to P-Block mapping. For the fingerprint store, we dynamically allocate the PCM space as per required. For the given configuration of SSD, the maximum possible size for fingerprint store would equal to *total number of flash pages* times *size of a fingerprint record*. The size of a fingerprint record is

192 bytes (160 bytes for fingerprint and 32 bytes for location).

Description	**Configuration**
No. of Packages	10
Planes per Package	8
Blocks per Plane	2048
Pages per Block	64
Page Size	4 KB
Read Latency	25μs
Write Latency	200μs
Erase Latency	1.5ms

Table 5.1: Configurations of SSD simulator

Workloads and Trace Collection

We use 5 workloads from two representative categories and run the experiment with them.

- Desktop (*d1,d2*)- These are typical office workloads, e.g. word editing, internet surfing, programming codes, etc. We collect these traces from our office computers when the computers are in use like programming, word editing and internet surfing. The workloads run for 10 and 20 hours, respectively. They feature irregular idle intervals, and generally small read/write accesses.

Database size	1 GB
Users Simulated	100
Buffer pool size	20 to 100 MB

Table 5.2: Configuration of TPC-C Benchmarks for our experiment

- Online Transaction (*t1,t2,t3*)- We execute TPC-C [2] workloads that run for 30 minutes, 2 hours and 4 hours respectively, to mainly test the performance of flash buffer extension. Since TPC-C workloads contain intensive writes, these are ideal for testing the efficiency of PCM-extension for flash buffer. To generate workloads from the benchmarks, we use a workload generation tool, *Hammerora* [1]. Hammerora is an open source tool written in Tcl/Tk. It runs with a MySQL database server on a Linux platform

under different configuration, a combination of the database size, system buffer size and number of simulated users. When the database server is run in these configuration, the tool produces log records during query processing. Table 5.2 shows our configuration settings. We fill the database with records to make it 1 GB size. Then for various database buffer pool sizes, we simulate online transaction by 100 users.

5.1.2 Results

We first use the hash-based encryption method as explained in above sections to test for the presence of duplicate data in our workloads. Then we run these workloads in our simulation system to test the effectiveness of duplication finder. For the second part, to test the performance of PCM-based flash buffer extender, we compare our buffer management scheme against two of the existing buffer management schemes, namely IPL [32] and dPRAM [44]. Finally, we compare the overall power saving by our system against the baseline configuration (without the duplicate finder and PCM-based buffer extender).

Efficiency of duplication finder

First, we measure the percentage of the duplicate data in the five workloads that we run our experiment on. The part *dup perc* in Figure 5.1 shows the percentage of duplicate data present in these workloads. The desktop workloads contain as much as 25% duplicate data, while the transaction workloads contain almost 30% duplicate data when it is run for 4 hours.

Then we use our duplication finder mechanism to see how much of these duplicated writes can be removed. The part *dup removed* in Figure 5.1 shows the percentage of removed duplicate writes from the workloads. The duplication finding rate depends on the size of the fingerprint store. If we choose to store all the fingerprints in the store, almost 100% duplicates can be detected. If the fingerprint store is too big, search of a fingeprint will be slow. We choose to use

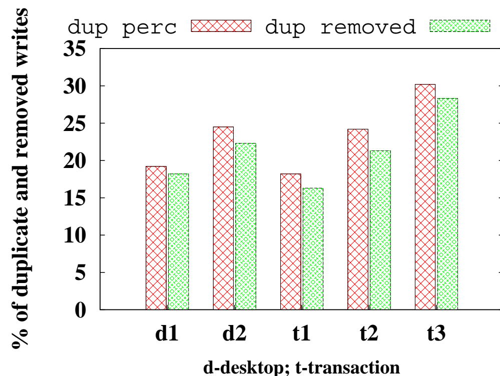
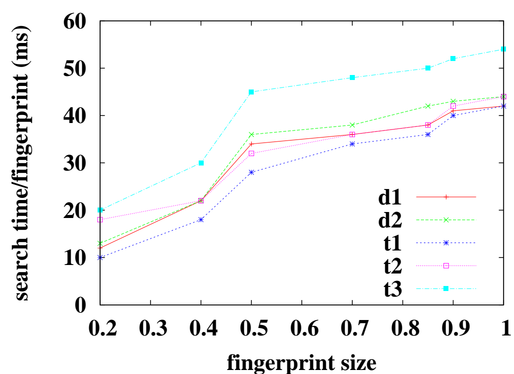


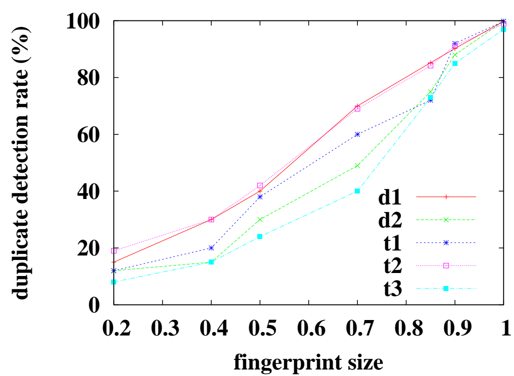
Figure 5.1: The duplication data present in the workloads

the maximum size fingerprint store in PCM. With that, the duplication finder can effectively detect almost all the duplicates as shown in the figure.
We study the effect of size of fingerprint store for the search time and detection of duplication. The results are shown in Figure 5.2. The x-axis shows the size of the fingerprint store. The maximum required fingerprint size is 1. As we can see from Figure 5.2(a), the smaller the size of fingerprint store, the faster is the search process, since the number of buckets needed to be compared is less. However, at the same time, this also means that duplication detection is decreased as the size of fingerprint store decreases (Figure 5.2(b)), as the fingerprint store cannot hold all the possible fingerprints, and when it is full, fingerprints are replaced according to *least recently used replacement scheme*. For workloads that contain high amount of duplicate data, having the maximum size fingerprint is a logical solution, as the *search time per fingerprint* does not increase sharply (Fig 5.2) once the fingerprint store size is more than half of the maximum possible size.

In addition to the simple removal of duplicate writes, this design scheme can also increase the available clean blocks for flash SSD. Figure 5.3 shows the percentage of saved flash space in units of blocks, compared to the baseline scheme where no duplication finder is used. For all the five workloads, the



(a)



(b)

Figure 5.2: The effect of fingerprint store size on (a) Search time per fingerprint; (b) Duplication detection rate

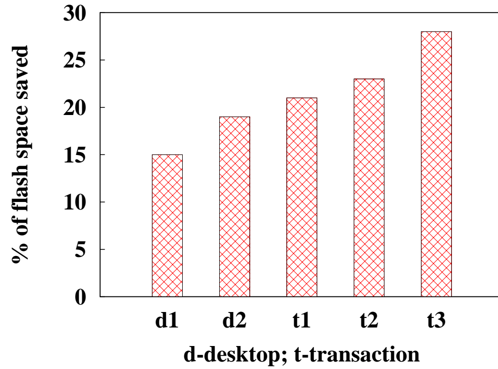


Figure 5.3: flash space saved by duplicate finder

saving is more than 15%. This is a significant amount, as explained in next section, this helps improve the endurance of flash SSD.

Performance of flash buffer management

In the second part of our experiment, we test the effectiveness of our PCM-based flash buffer extender. To make the case for our proposed design, we compare our scheme against two existing schemes IPL [32] and dPRAM [44]. We choose to use only the TCP-C transaction workloads for this experiment as these workload contain the frequent smaller writes that truly test the effectiveness of our PCM log region. We use a database size of 1 GB, and change the buffer pool size from 20 MB to 100 MB to test the efficiency of this scheme under different environments.

We know that large overhead of writes and erase operations are the main obstacles in deficiency of flash memory. As shown in Figure 5.4, number of writes to a database storage medium decrease as the data buffer size of the database increase. We consider different buffer sizes to show the efficiency of our buffer management scheme under different environments.

The comparison of merge numbers for different techniques is shown in Figure 5.5(a). *IPL* represents the pure NAND flash memory using IPL buffer management method. *d-PRAM* represents the dynamic hybrid architecture using the

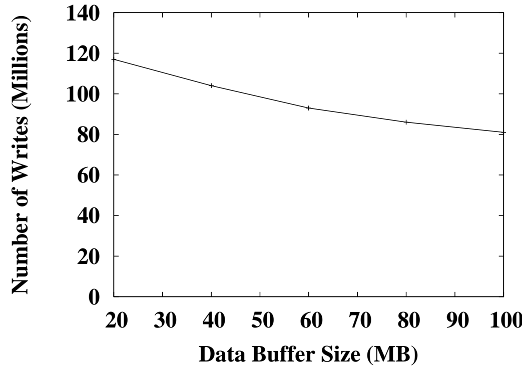
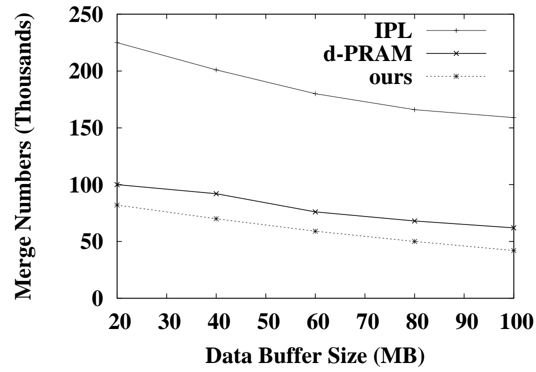


Figure 5.4: The impact of data buffer size on write operations

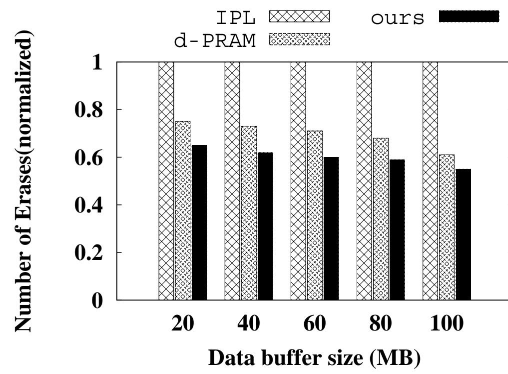
basic dynamic log assignment. *ours* represents our PCM-aided buffer management technique. Our method is able to further decrease the amount of merges, and consequently the write time, because, logs of all the pages of flash blocks are stored in block-like fashion in PCM, and when the whole block is ready to be flushed, only then are these merged with the original data.

We also compare the number of erases for these three techniques in Figure 5.5(b). Keeping the logs away in non-volatile medium brings down the frequency of block erase. Since the flushes to flash after merge operation are carried out sequentially all the time, each erase represents a write of all the pages of that erase unit. This enables us to avoid the requirement of erasing a whole erase unit (block) just to update a single page of that block. Our PCM-extension architecture outperforms both IPL and d-RAM in this regard.
We also calculate the total write time. This write time depends on the number of merge and erase operations. Thus, we can draw similar conclusion for the write time. The comparison of write times for three techniques is shown in Figure 5.5(c).

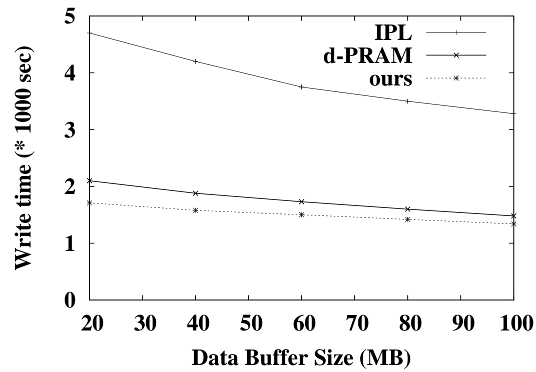
Besides the energy consumption for writes, we also compute the energy consumption for read operations. To calculate energy, we multiply the number of reads or writes by amount of energy required per read or write. In our method,



(a)



(b)



(c)

Figure 5.5: The comparison of (a) Merge Numbers; (b) Erase Numbers; and (c) Write time for three techniques

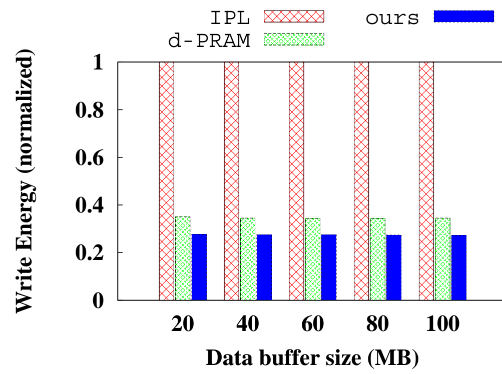
when a page is brought into flash buffer for write, the write is reflected in the PCM log region immediately and the page is discarded. This allows space in the buffer for a new incoming page without having to replace an old page. Thus frequently read pages are more likely to be found in the buffer than that is for IPL and d-PRAM method. The other reason this PCM-based buffer extension architecture saves energy in reading is, reading from PCM is faster than reading from flash. The energy for write, read and total energy are shown in Figures 5.2.2(a), 5.2.2(b) and 5.2.2(c), respectively.

Making Sequential Flushes to flash

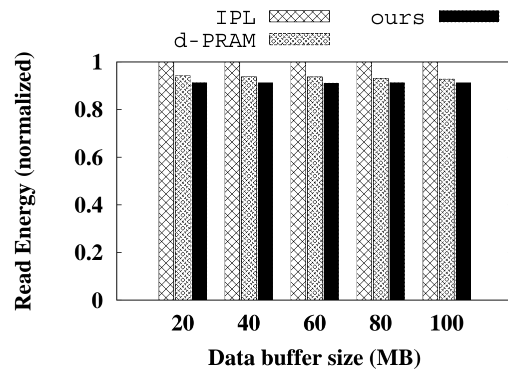
One of the major reason that the PCM-based flash buffer extender can be so effective is because it can delay the numerous small writes to flash, then when the writes do occur, it writes them sequentially. The increase in sequential flushes to flash as compared to the baseline configuration (without PCM extender) is shown in Figure 5.7. As we can see, the sequentiality of writes to flash increase by almost 50% for *t3* workload which is the longest of the three transaction workloads.

Combining all together

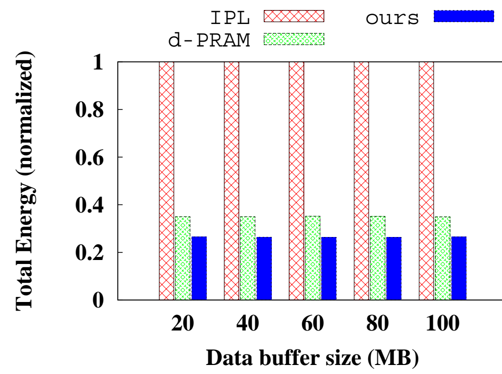
Both the first part (duplication remover) and the second part (flash buffer extender) can independently help improve the write efficiency, lifetime and energy savings of the system. When we combine these two together, the improvement will be even better. We compare these improvements versus the baseline configuration (with no duplication detector and buffer extender) in Figure 5.8. By just adding a PCM of 16 MB, we can achieve a lifetime improvement of almost four times for SSD while consuming less than 80% energy to do so.



(a)



(b)



(c)

Figure 5.6: The comparison of Energy consumption for (a) Write operation; (b) Read Operation; (c) Write + Read

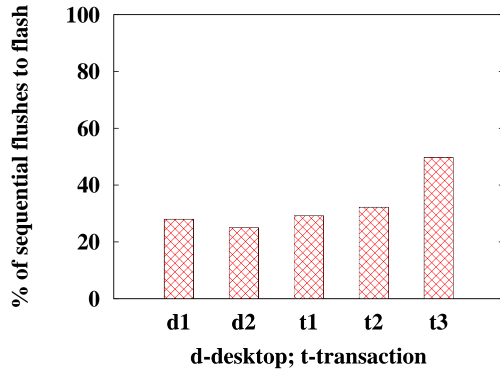
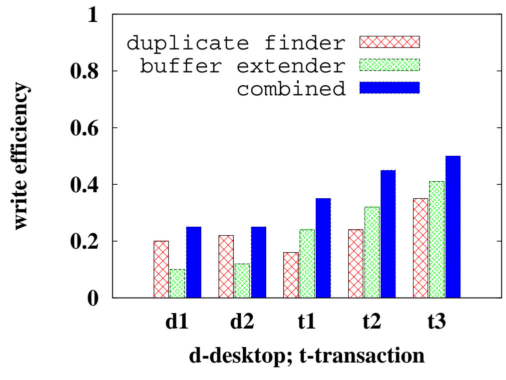


Figure 5.7: Percent of sequential flush to flash due to PCM-based buffer management

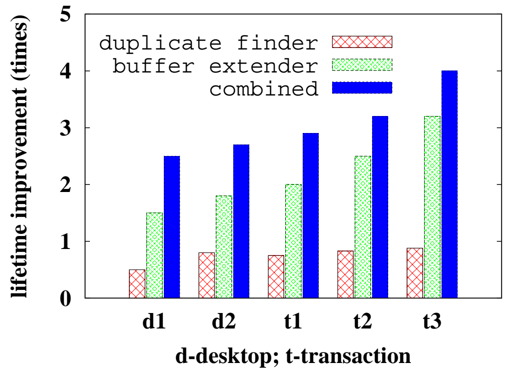
5.2 Hash-join algorithm in PCM-based Database

We evaluate the performance of traditional hash join algorithm [38] and modified hash join algorithms in terms of execution time, PCM energy and wear-out level, for column-based and row-based database as well as for database with PCM as the main memory by using cycled-simulations. First of all, we emulate the PCM on DRAM of a server machine whose configuration are shown in table 5.3.

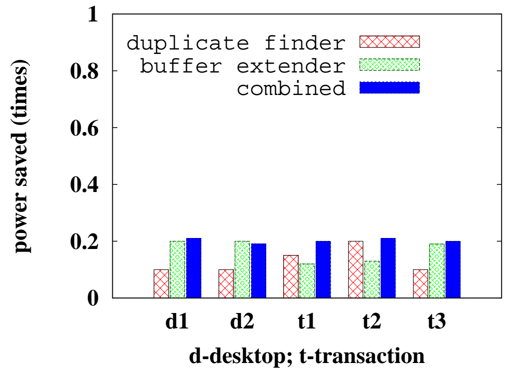
To emulate DRAM as a PCM, we change the read write time, and emulate the wear out behavior of PCM by introducing a counter on the DRAM cells that get written. We study PCM as a faster hard-disk as well as a DRAM extension. Simulation configurations for these two architectures are different. For PCM as a faster hard-disk, data would be required to brought into a DRAM to complete read or write, whereas in its use as a DRAM extension we suppose that data from PCM do not need to be brought into the DRAM to complete read/write operation.



(a)



(b)



(c)

Figure 5.8: Effect of duplication finder and pcm-based buffer extender on (a)Write Efficiency; (b) Lifetime; (c) Power save

5.2.1 Simulation Parameters

We basically use the DRAM memory and emulate it as a PCM by algorithmically delaying the read and write access time. Table 5.3 describes the simulation parameters.

Processor	2x Intel®Xeon®E7540 2.00GHz, 18M cache
DRAM simulated as PCM	8 GB
DRAM used as extension	256 MB
PCM	4 ranks, read latency for a cache line: 230 cycles, write latency per $8B$ modified word: 450 cycles, $E_{rb} = 2pJ$, $E_{wb} = 16pJ$

Table 5.3: Simulation Parameters

We use a DRAM and emulate it to be like a PCM by modifying the read-/write latency. To simulate the wear-out phenomena of PCM, we write a program that counts the cells that are written in DRAM. We also implement the general wear-out balancing algorithms used in PCM to even-out the wear across the PCM. The word size of 8 bytes per iteration of write operations is based on [18].

When PCM is being used as a faster hard-disk, we take into consideration the copying the PCM contents to DRAM during read, and copying content from DRAM to PCM during write. But, when PCM is used as a main memory, these two steps do not exist.

We run our experiment in a multiple-user environment. A number of database queries, that may access the same database tables at the same time are run. The processes are generally getting the relation for the purpose of a read, so general lock is inclusive lock. If update transactions come for the records that have inclusive locks from other process, the update process is held back.

These transactions are run simultaneously for a number of users each time. We do experiments for user numbers ranging from 20 to 50.

5.2.2 Modified Hash-join for Row-stored and Column-stored Database

We implemented four hash join algorithms for row-stored(R-S) and column-stored(C-S) databases as discussed in Section 3.2: traditional hash-join for R-S, modified hash join for R-S, traditional hash-join for C-S, and modified hash-join for C-S. The database based on star schema benchmark is stored in the emulated PCM. To partition a relation taking part in join, first we use a hash function to hash the record on its join key field then partition it according to the hash value. Depending on the hash-join method, either the partition containing whole record or only its tuple ID are written back to PCM for next step.

During join stage, first a partition from a smaller relation is brought into buffer, and hashed using a hash function. Then the partition with same partition number from bigger relation is brought and probed for matching tuple. The join results are then sent to a high-level operator, which produces the final result.

We run the experiment in a multi-user environment where many users are running a number of queries at the same time. Three queries explained in above section are frequently used by these queries that require hash-join operation. Besides these queries, the users run many other *insert*, *update*, and *delete* transactions that could change the result of a same query for different users. We first run the experiment by fixing the number of users at 100, but varying the size of database, i.e. the size of each relations by 1.0X to 4.0X factors. A factor of 1.0X represent the database size as explained in Star Schema in Figure 2.3. Then in the second experiment, we fix the initial size of database, but increase the number of users operating at the same time from 20 to 200.

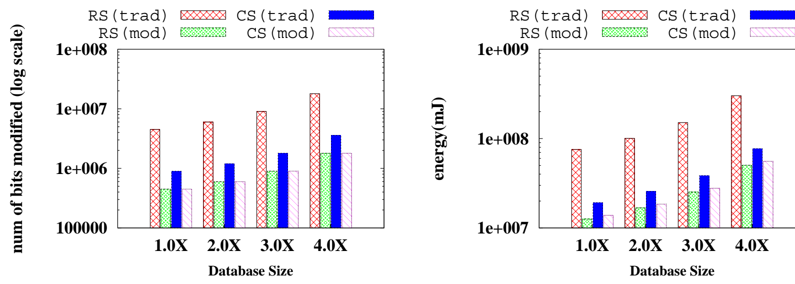
Figure 5.9 show the results for 4 types of hash-joins when size of database is gradually increased. Figure 5.9(a) show that row-stored database causes high

PCM wear compared to column-stored database for traditional hash-join. This is because all the records of a relation are written back to PCM for traditional method, which is bigger than the tuple ID list of modified method. This result validates our assumption in Section 4.1.2. But this does not necessarily mean column-stored always wins the PCM-wear race, because creating a column-stored database, then maintaining it is costlier than maintaining row-stored database because it requires inserting extra column of tuple ID for each of the attributes. The wear level is almost the same for both R-S and C-s modified hash-joins. Because both of them only copy the tuple ID back to PCM. Even though size of reads vary for these two types, wear only depends on the size of write, thus wear to PCM is almost the same.

Figure 5.9(b) compares the energy for two types of databases. It shows that modified hash-join for C-S database consumes slightly more energy than the modified hash-join for R-S. This result is particularly interesting, because from energy stand point, maintaining C-S database is more expensive. As the modified hash-join is also costlier for C-S, R-S wins the battle of PCM energy saved.

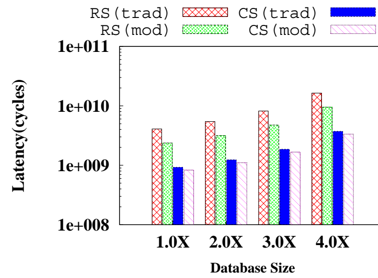
In terms of latency, column-stored is clearly better because it only needs to read the column that takes part in join. The results are shown in figure 5.9(c).

The results from figure 5.10 show that the hash-join performance for two types of PCM-databases when the database size is fixed at 1.0X and number of users accessing the database is increased from 20 to 200. Each user runs multiple queries of *insert*, *update* and *delete* operation besides the three queries explained in Section 2.2.5. The results show PCM wear, PCM Energy and access latency caused by hash-join operation while running the queries from multiple users for a certain amount of time.



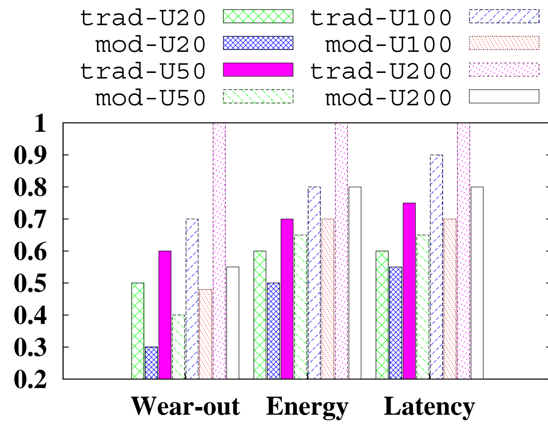
(a) Total Wear

(b) PCM Energy

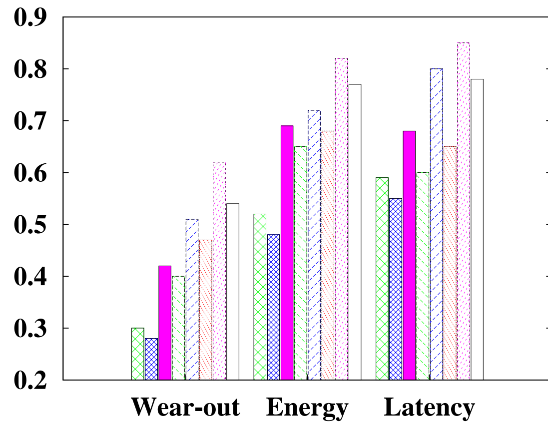


(c) Access Latency

Figure 5.9: Hash join Performance for various Database Size



(a) Row-Stored Database



(b) Column-Stored Database

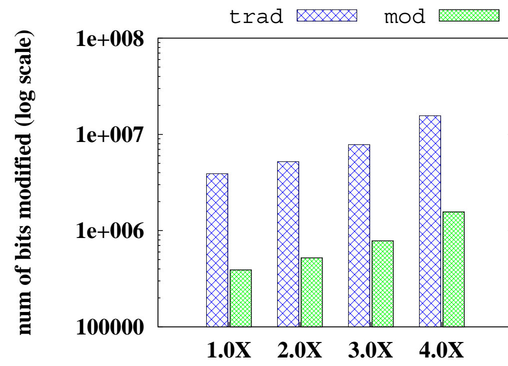
Figure 5.10: Comparison of traditional and modified hash joins for R-S and C-S databases by increasing user size from 20 (U20) to 200(U200)

In the figures, 'trad-U20' indicates traditional hash-join methods run in a 20-user environment, 'mod-U20' means modified hash-join run in a 20-user environment, and so forth. We consider the wear, energy and latency factor for R-S traditional hash-join when users are 200 (at 'trad-U200') to be the maximum value (equal to 1). Results for both databases show that modified hash-join outperform the traditional hash-join irrespective of the number of users running at the same time. And in general, column-stored database consumes less energy, creates less wear on PCM, and takes less time to execute. But again, this result does not consider the maintenance cost of C-S and R-S databases. Thus it will not be wise to declare C-S as a clean winner. Rather, we can say that for both types of databases, our modified hash-join technique brings a significant performance improvement.

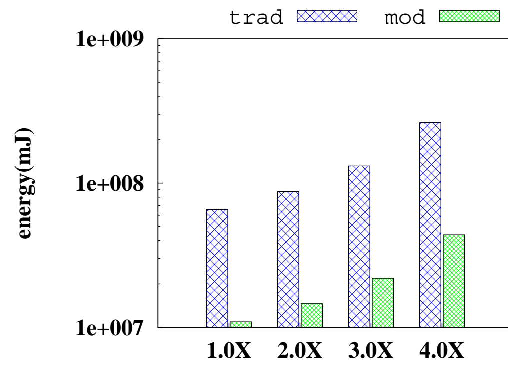
5.2.3 PCM as a Main Memory Extension

For this experiment, it is considered that the entire relations are stored in PCM, and CPU can directly read from and write to PCM. No data need be transferred between PCM and DRAM. Traditional hash-joins tend to create many partitions and re-write them in PCM. We can avoid the actual physical copying of the partitions into PCM by only saving the record ids of the records that belong to same partitions.

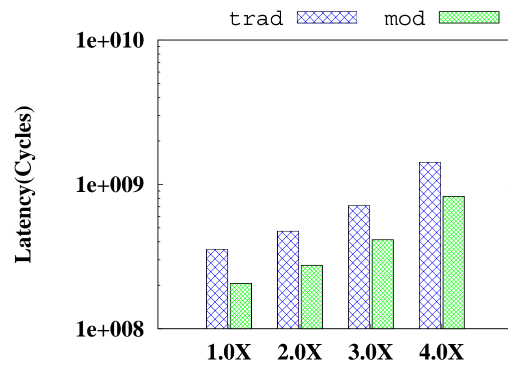
We compare of read/write access numbers, query execution time and wear out of PCM with varying record size of query output for traditional and modified hash-join methods. For this experiment, we chose the traditional row-based database storage model, and use the modified hash-join used for row-stored in above section. The algorithms vary in the fact that reading a record from PCM does not involve the step of bringing it back to DRAM anymore. Thus read/write access latency will change. PCM-wear and energy will primarily be similar to that of the the row-based approach.



(a) Total Wear



(b) PCM Energy



(c) Access Latency

Figure 5.11: hash join Performance for a PCM-as-a-Main-Memory-Database

The results in Figure 5.11 show that modified hash-join can reduce the number of writes by trading it with read numbers. Since wear out of PCM is directly related to the write numbers, this method can effectively improve the life-cycle by as much as 60%, energy by 50% and access latency by about 30 %.

Chapter 6

Conclusion

In this thesis, we study the position of Phase Change Memory(PCM) in the memory hierarchy, and how flash SSD buffer and database systems can exploit such memory. First, we propose two separate techniques to reduce the write traffic to flash SSD, and improve the write efficiency to increase the lifetime of SSD while bringing the power consumed by the overall system down.
We take advantage of the redundancy present on normal workloads to reduce the write traffic to flash SSD. We proposed a hash-based encryption method to detect and avoid the duplicate writes to flash (Fig 3.2). We show that this itself can save a lot of flash space., which ultimately help increase its lifetime (Fig 5.3).

In the second part, we migrated the frequent small updates on flash pages to PCM as logs. The inherent limitation of 'erase-before-write' on flash makes it very difficult to optimize the write operations on flash without taking support of external update-efficient medium. PCM is an emerging storage-class memory with advantages like non-volatility, faster read/write access, in-place updating and higher density. In this part, first of all, we record the changes on flash data pages into the log sectors of PCM, which is update-efficient as it allows the in-place update. Then, we organize the PCM buffer space in such a way that when we want to merge the logs with old data, the writes to flash are sequentialized.

We proposed a block popularity based replacement scheme for PCM to even out the wear-leveling among the blocks. We show by simulation result that this proposed method outperforms two existing log-buffer extension techniques.

Finally, we combine these two design techniques together and compare their performance against the baseline configuration. We can see that the system can improve the lifetime of flash SSD by more than four times (Fig 5.8(b)) while consuming 20% less power to do so (Fig 5.8(c)).

In the second work of the thesis, we study the performance of PCM-based database system. We study how the query optimization algorithms should be optimized when PCM is used as a row-stored database and as a column-stored database. We chose hash-join algorithm as a performance test-case to study the behavior of PCM-based database. We proposed two modified hash-join algorithm for both row-stored and column-stored database. We also study the performance of modified row-stored hash-join algorithm when it is used in a database system where PCM is used primarily as a extension of main memory.

This thesis does not conclude that PCM is more suitable for row-stored, or column-stored, or as a main-memory extension type databases. Rather, with experiments on all types of databases we show that to gain performance benefits, and to exploit the PCM's architectural advantages, we can modify these algorithms. We show, by experiment with emulated PCM simulator, that modified hash-join algorithm, which basically trades off the more expensive PCM writes with cheaper PCM reads, can improve the life-time, energy savings and execution time for PCM.

Bibliography

[1] Hammerora: The open source oracle load test tool. *http://hammerora.sourceforge.net/*.

[2] Tpc benchmark. *http://www.tpc.org/*.

[3] Tpc-h benchmark. *http://www.tpc.org/tpch*.

[4] D.J. Abadi, S.R. Madden, and N. Hachem. Column-stores vs. row-stores: How different are they really? In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 967–980. ACM, 2008.

[5] L. Adam. Flash storage memory. *Communications of the ACM*, 51(7), 2008.

[6] N. Agrawal, V. Prabhakaran, T. Wobber, J.D. Davis, M. Manasse, and R. Panigrahy. Design tradeoffs for ssd performance. In *USENIX 2008 Annual Technical Conference on Annual Technical Conference*, pages 57–70. USENIX Association, 2008.

[7] D.G. Andersen and S. Swanson. Rethinking flash in the data center. *IEEE Micro*, 30(4):52–54, 2010.

[8] M. Balakrishnan, A. Kadav, V. Prabhakaran, and D. Malkhi. Differential raid: rethinking raid for ssd reliability. *ACM Transactions on Storage (TOS)*, 6(2):4, 2010.

[9] L.A. Barroso. Warehouse-scale computing. In *Proceedings of the 2010 international conference on Management of data*, page 2. ACM, 2010.

[10] A. Birrell, M. Isard, C. Thacker, and T. Wobber. A design for high-performance flash disks. *ACM SIGOPS Operating Systems Review*, 41(2):88–93, 2007.

[11] S. Boboila and P. Desnoyers. Write endurance in flash drives: Measurements and analysis. In *Proceedings of the 8th USENIX conference on File and storage technologies*, pages 9–9. USENIX Association, 2010.

[12] J.S. Bucy, J. Schindler, and S.W. Schlosser. The disksim simulation environment version 4. 0 reference manual. *Environment*, (CSE-TR-358-98), 2008.

[13] F. Chen, D.A. Koufaty, and X. Zhang. Understanding intrinsic characteristics and system implications of flash memory based solid state drives. In *Proceedings of the eleventh international joint conference on Measurement and modeling of computer systems*, pages 181–192. ACM, 2009.

[14] F. Chen, T. Luo, and X. Zhang. Caftl: a content-aware flash translation layer enhancing the lifespan of flash memory based solid state drives. In *Proceedings of the 9th USENIX conference on File and stroage technologies*, pages 6–6. USENIX Association, 2011.

[15] S. Chen, P.B. Gibbons, and S. Nath. Rethinking database algorithms for phase change memory. In *Proc. CIDR*, 2011.

[16] X. Chen, P. O'Neil, and E. O'Neil. Adjoined dimension column index (adc index) to improve star schema query performance.

[17] M.L. Chiang and R.C. Chang. Cleaning policies in mobile computers using flash memory. *Journal of Systems and Software*, 48(3):213–231, 1999.

[18] S. Cho and H. Lee. Flip-n-write: a simple deterministic technique to improve pram write performance, energy and endurance. In *Microarchitecture,*

2009. MICRO-42. 42nd Annual IEEE/ACM International Symposium on, pages 347–357. IEEE, 2009.

[19] T.S. Chung, D.J. Park, S. Park, D.H. Lee, S.W. Lee, and H.J. Song. A survey of flash translation layer. *Journal of Systems Architecture*, 55(5):332–343, 2009.

[20] PUB FIPS. 180-1. secure hash standard. *US Department of Commerce*, 1995.

[21] H. Garcia-Molina and K. Salem. Main memory database systems: An overview. *Knowledge and Data Engineering, IEEE Transactions on*, 4(6):509–516, 1992.

[22] L.M. Grupp, A.M. Caulfield, J. Coburn, S. Swanson, E. Yaakobi, P.H. Siegel, and J.K. Wolf. Characterizing flash memory: anomalies, observations, and applications. In *Microarchitecture, 2009. MICRO-42. 42nd Annual IEEE/ACM International Symposium on*, pages 24–33. IEEE, 2009.

[23] A. Gupta, Y. Kim, and B. Urgaonkar. *DFTL: a flash translation layer employing demand-based selective caching of page-level address mappings*, volume 44. ACM, 2009.

[24] A. Inoue and D. Wong. Nand flash applications design guide. *Toshiba America Electronic Components Inc*, 2004.

[25] D.H. Kang, D.H. Ahn, K.B. Kim, JF Webb, and K.W. Yi. One-dimensional heat conduction model for an electrical phase change random access memory device with an 8f memory cell (f= 0.15 μm). *Journal of applied physics*, 94:3536, 2003.

[26] J.K. Kim, H.G. Lee, S. Choi, and K.I. Bahng. A pram and nand flash hybrid architecture for high-performance embedded storage subsystems. In *Proceedings of the 8th ACM international conference on Embedded software*, pages 31–40. ACM, 2008.

[27] S. Lai. Current status of the phase change memory and its future. In *Electron Devices Meeting, 2003. IEDM'03 Technical Digest. IEEE International*, pages 10–1. IEEE, 2003.

[28] C. Lam. Cell design considerations for phase change memory as a universal memory. In *VLSI Technology, Systems and Applications, 2008. VLSI-TSA 2008. International Symposium on*, pages 132–133. IEEE, 2008.

[29] C. Lam. Cell design considerations for phase change memory as a universal memory. In *VLSI Technology, Systems and Applications, 2008. VLSI-TSA 2008. International Symposium on*, pages 132–133. IEEE, 2008.

[30] C. Lam. Cell design considerations for phase change memory as a universal memory. In *VLSI Technology, Systems and Applications, 2008. VLSI-TSA 2008. International Symposium on*, pages 132–133. IEEE, 2008.

[31] B.C. Lee, E. Ipek, O. Mutlu, and D. Burger. Architecting phase change memory as a scalable dram alternative. In *ACM SIGARCH Computer Architecture News*, volume 37, pages 2–13. ACM, 2009.

[32] S.W. Lee and B. Moon. Design of flash-based dbms: an in-page logging approach. In *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, pages 55–66. ACM, 2007.

[33] S.W. Lee, B. Moon, C. Park, J.M. Kim, and S.W. Kim. A case for flash memory ssd in enterprise database applications. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 1075–1086. ACM, 2008.

[34] Mearian Lucas. Ibm announces computer memory breakthrough. *http://www.computerworld.com/s/article/9218031.*

[35] A.J. Menezes, P.C. Van Oorschot, and S.A. Vanstone. *Handbook of applied cryptography.* CRC, 1997.

[36] V. Mohan, T. Siddiqua, S. Gurumurthi, and M.R. Stan. How i learned to stop worrying and love flash endurance. In *Proceedings of the 2nd USENIX conference on Hot topics in storage and file systems*, pages 3–3. USENIX Association, 2010.

[37] Y. Park, S.H. Lim, C. Lee, K.H. Park, et al. Pffs: a scalable flash memory file system for the hybrid architecture of phase-change ram and nand flash. In *Proceedings of the 2008 ACM symposium on Applied computing*, pages 1498–1503. ACM, 2008.

[38] J.M. Patel, M.J. Carey, and M.K. Vernon. Accurate modeling of the hybrid hash join algorithm. In *ACM SIGMETRICS Performance Evaluation Review*, volume 22, pages 56–66. ACM, 1994.

[39] M.K. Qureshi, V. Srinivasan, and J.A. Rivers. Scalable high performance main memory system using phase-change memory technology. In *ACM SIGARCH Computer Architecture News*, volume 37, pages 24–33. ACM, 2009.

[40] A. Rajimwale, V. Prabhakaran, and J.D. Davis. Block management in solid-state devices. In *Proceedings of the 2009 conference on USENIX Annual technical conference*, pages 21–21. USENIX Association, 2009.

[41] S. Raoux, GW Burr, MJ Breitwisch, CT Rettner, Y.C. Chen, RM Shelby, M. Salinga, D. Krebs, S.H. Chen, H.L. Lung, et al. Phase-change random access memory: A scalable technology. *IBM Journal of Research and Development*, 52(4.5):465–479, 2008.

[42] R. Rivest. The md5 message-digest algorithm. 1992.

[43] M. Rosenblum and J.K. Ousterhout. The design and implementation of a log-structured file system. *ACM Transactions on Computer Systems (TOCS)*, 10(1):26–52, 1992.

[44] G. Sun, Y. Joo, Y. Chen, D. Niu, Y. Xie, Y. Chen, and H. Li. A hybrid solid-state storage architecture for the performance, energy consumption, and lifetime improvement. In *High Performance Computer Architecture (HPCA), 2010 IEEE 16th International Symposium on*, pages 1–12. IEEE, 2010.

[45] H.S.P. Wong, S.B. Kim, B. Lee, M.A. Caldwell, J. Liang, Y. Wu, R.G.D. Jeyasingh, and S. Yu. Recent progress of phase change memory (pcm) and resistive switching random access memory (rram). In *Solid-State and Integrated Circuit Technology (ICSICT), 2010 10th IEEE International Conference on*, pages 1055–1060. IEEE, 2010.