Rethinking Computer Architectures and Software Systems for Phase-Change Memory

CHENGWEN WU and GUANGYAN ZHANG, Tsinghua University
KEQIN LI, State University of New York

With dramatic growth of data and rapid enhancement of computing powers, data accesses become the bottleneck restricting overall performance of a computer system. Emerging phase-change memory (PCM) is byte-addressable like DRAM, persistent like hard disks and Flash SSD, and about four orders of magnitude faster than hard disks or Flash SSDs for typical file system I/Os. The maturity of PCM from research to production provides a new opportunity for improving the I/O performance of a system. However, PCM also has some weaknesses, for example, long write latency, limited write endurance, and high active energy. Existing processor cache systems, main memory systems, and online storage systems are unable to leverage the advantages of PCM, and/or to mitigate PCM's drawbacks. The reason behind this incompetence is that they are designed and optimized for SRAM, DRAM memory, and hard drives, respectively, instead of PCM memory. There have been some efforts concentrating on rethinking computer architectures and software systems for PCM. This article presents a detailed survey and review of the areas of computer architecture and software systems that are oriented to PCM devices. First, we identify key technical challenges that need to be addressed before this memory technology can be leveraged, in the form of processor cache, main memory, and online storage, to build high-performance computer systems. Second, we examine various designs of computer architectures and software systems that are PCM aware. Finally, we obtain several helpful observations and propose a few suggestions on how to leverage PCM to optimize the performance of a computer system.

Categories and Subject Descriptors: H.3.2 [**Information Storage and Retrieval**]: Information Storage

General Terms: Design, Algorithms, Management

Additional Key Words and Phrases: Computer architecture, energy consumption, I/O performance, phase-change memory, system software, write lifetime

ACM Reference Format:
Chengwen Wu, Guangyan Zhang, and Keqin Li. 2016. Rethinking computer architectures and software systems for phase-change memory. J. Emerg. Technol. Comput. Syst. 12, 4, Article 33 (May 2016), 40 pages.
DOI: http://dx.doi.org/10.1145/2893186

1. INTRODUCTION

Currently, to deal with the high demands of data-intensive applications, it is easy for modern and large-scale systems built by commercial processors to provide unbalanced

This work is supported by the National Natural Science Foundation of China under Grants 61170008, 61272055, 61433008, and U1435216; the National Grand Fundamental Research 973 Program of China under Grant 2014CB340402; and the National High Technology Research and Development Program of China under Grant 2013AA01A210.
Authors' addresses: C. Wu and G. Zhang (corresponding author), Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China; email: wcw14@mails.tsinghua.edu.cn, gyzh@tsinghua.edu.cn; K. Li, Department of Computer Science, State University of New York, New Paltz, New York 12561; email: lik@newpaltz.edu.

DOI: http://dx.doi.org/10.1145/2893186

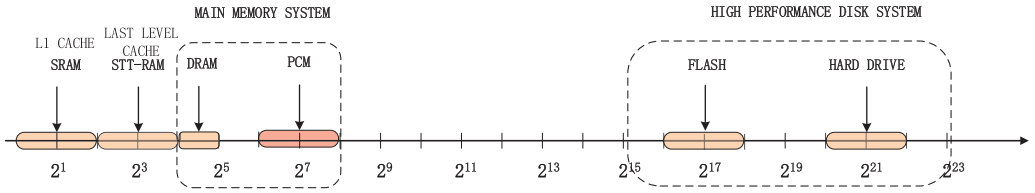


Fig. 1. Typical access latency (ns) of various devices (from Yang et al. [2015]).

computing resources with oversupplied CPU cycles and increasingly long latency of data accesses. In recent years, multicore technology has improved computing powers significantly, which aggravates the performance limitation of the memory hierarchy. Moreover, many applications, especially those in the server space, have large working sets, hence require high memory capacity. The demand for greater memory capacity is expected to continue into the future, even for commodity systems. On the other hand, multiple applications tend to be executed on a computer system simultaneously, which makes different data-access patterns interleaved. Furthermore, several applications are also I/O intensive, hence require a high-performance storage system. Consequently, the memory hierarchy plays a critical role in determining overall performance of a computer system.

The memory hierarchy of modern computers is designed such that the layers closer to the processor cores provide shorter latency, whereas those layers farther away provide higher capacity, albeit with longer access latency. For many years, static random-access memory (SRAM), dynamic random-access memory (DRAM), and rotating disks have served as the bedrock technologies for designing processor cache, main memory, and online storage, respectively. However, continued use of these technologies poses several challenges. While SRAM provides very low latency, it has the drawbacks of high-leakage power consumption and low density [Wu et al. 2009a]. While DRAM provides low latency, it suffers from high power consumption (especially leakage power) and scalability problems [Arden 2009]. While hard disk drives provide low cost-per-gigabyte of storage, they suffer from high-access latencies (several orders of magnitude higher than main memory) and also consume a significant amount of power. In IBM servers, for instance, about 50% of the energy is spent in off-chip memory hierarchy [Lefurgy et al. 2003]. While replacing rotating disks with flash memory in the form of solid-state disks (SSDs) can provide a large performance boost for storage, there is still a wide gap in the performance between main memory and online storage systems (Figure 1). In order to improve the overall performance of a computer system, it is necessary to narrow the performance gap between main memory and online storage.

One way to address the performance, capacity, power, and scalability problems of the traditional memory and storage technologies is to use phase-change memory (PCM). PCM is a nonvolatile memory technology that is being actively explored by academia and industry, and is closest to large-scale production. PCM stores data with phase-change material that can be in one of two states—crystalline or amorphous. It reads data by detecting the resistance of phase-change material in one state and attributing it to the state of a bit. The performance of PCM is between DRAM and online storage (Figure 1). The latency of PCM-based main memory is closer to that of DRAM [Park et al. 2010]. Since PCM is up to four orders of magnitude faster than today's hard disks for typical file system I/Os, incorporating PCM will improve the performance of a system when a page fault occurs [Qureshi et al. 2009]. Existing evaluation results show that PCM has some obvious advantages, such as high performance, high density, and low idle energy consumption.

In recent years, there have been exciting research achievements in each layer across the memory hierarchy to explore how to take advantage of PCM. Some representative efforts are listed as follows. From these achievements, we can safely predict that PCM will bring a significant and positive impact on the performance of a computer system when PCM is used widely.

—*Processor Cache:* Hybrid nonvolatile memory caches including SRAM, PCM, and magnetic RAM (MRAM) [Wu et al. 2009a, 2009b]; Pure PCM processor cache system [Joo et al. 2010].
—*Main Memory:* Hybrid main memory architecture with PCM and DRAM [Qureshi et al. 2009]; main memory systems using PCM [Lee et al. 2009; Zhou et al. 2009].
—*Online Storage:* Using PCM as disk cache in servers [Roberts 2011]; using PCM as solid state storage subsystems [Lu et al. 2012]; Flash file systems based on hybrid architecture of PCM and Flash [Park et al. 2008].

While PCM provides several benefits, it also poses certain key challenges that need to be addressed before this memory technology can be leveraged to build high-performance computer systems. Current processor cache systems, main memory systems, and storage system software are unable to adapt to PCM's characteristics well, because they are designed and optimized for SRAM, DRAM memory, and hard drives, instead of PCM memory. On the one hand, there are significant differences between PCM and DRAM that is used in main memory for many years. PCM is nonvolatile and has limited write endurance. These differences lead to the need to optimize or even redesign main memory system for PCM. On the other hand, PCM is byte-addressable and obviously faster than hard disks or Flash SSDs. Consequently, it is necessary to optimize the design of file systems designed for HDD or SSD when using PCM in the form of solid-state storage. In a word, the maturity of PCM from research to production provides new opportunities and challenges for designing computer architectures and software systems.

This article provides a detailed survey and review of the areas of computer architecture and software systems that are oriented to PCM devices. The contributions of this article are as follows. First, we identify key technical challenges that need to be addressed before this memory technology can be leveraged, in the form of processor cache, main memory, and online storage, to build high-performance computer systems. Second, we examine various designs of computer architectures and software systems that are PCM aware. Finally, we obtain several helpful observations and propose a few suggestions on how to leverage PCM to optimize the performance of a computer system.

This survey is restricted to advantages, challenges, and solutions when PCM is incorporated in computer systems. We largely ignore the details of PCM material and working mechanisms unless they affect designs of computer systems. Instead, we analyze advantages and challenges brought by PCM's external attributes, for example, performance, energy consumption, and write endurance. On the other hand, to design a well-engineered system, it is necessary to pay attention to many diverse goals, and the system designer must decide how to weigh different axes of interest during the design phase. This article focuses on how to design a system that adapts to PCM devices well while ignoring many other important considerations in the design of computer systems. The survey aims at guiding people to understand how choices made by different system designers affect the ability of computer systems to leverage advantages of PCM, and avoid or alleviate the shortcoming of PCM.

The rest of the article is organized as follows. Section 2 describes device attributes of PCM, identifies technical challenges to use PCM in processor cache, main memory, and online storage, and compares PCM with other resistive memory technologies. Section 3 summarizes the works of applying PCM to processor cache. Section 4 analyzes

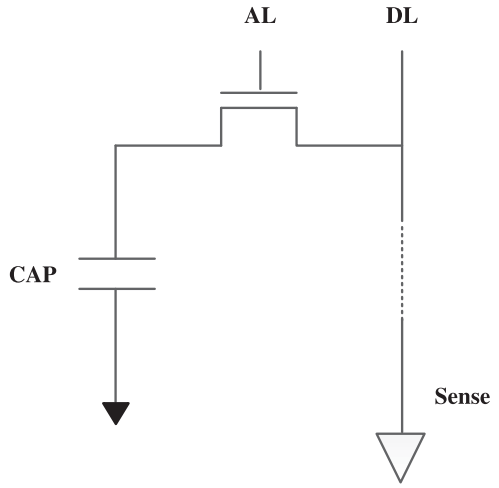


Fig. 2. A single of DRAM extracted from Drepper [2007].

performance, lifetime, and energy consumption optimizations for PCM as main memory. Section 5 discusses how to design and optimize software systems for PCM as storage devices. Some studies on using PCM as both main memory and storage are provided in Section 6. We provide our conclusions and some prospective research directions in Section 7.

2. DEVICE ATTRIBUTES AND TECHNICAL CHALLENGES

In this section, we first analyze attributes of PCM devices. Next, we discuss prospective applications of PCM devices, which is still an open issue. Then, we identify technical challenges when designing computer architectures and software systems for PCM. Finally, we give a comparison between PCM and other new resistive memory technologies.

2.1. PCM Attributes

2.1.1. Review of DRAM. To clearly describe attributes of PCM devices, we first review DRAM, which has been used in main memory for over 30 years. A typical DRAM cell has one capacitor and one transistor, as shown in Figure 2. The capacitor keeps the state of the DRAM cell, while the transistor is used to provide access to the state of the cell. When the state of the cell is read, the access line (*AL)* will be raised. This either causes a current to flow on the data line (*DL)* or not, depending on the charge in the capacitor. When the cell is written, the *DL* is appropriately set, then the *AL* will be raised for a time long enough to charge or drain the capacitor.

Being a type of charge-based memory, DRAM has some drawbacks. First, DRAM is hard to scale. The capacitor should be large enough for reliable state detection. Also, the access transistor should be large enough for low leakage and long retention time. Hence, it is difficult to improve the density and capacity of DRAM further. Second, DRAM is very energy-consuming [Drepper 2007]. In order to address the problem of current leakage, DRAM cells have to be refreshed frequently (every 64ms in most current DRAM devices). Furthermore, each read operation must be followed by an operation to recharge the capacitor, since the capacitor is discharged by read operations.

2.1.2. Properties of Phase Change Materials. Phase change materials have two metastable states—one amorphous, the other crystalline. They can be rapidly and

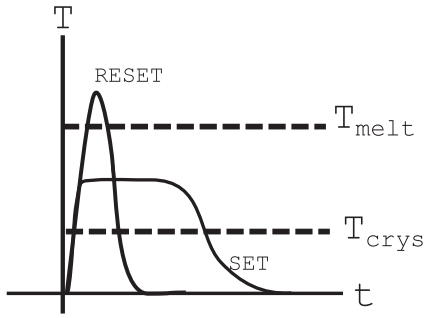


Fig. 3. Time and voltage temperature required to program PCM (from Raoux et al. [2008]).

repeatedly switched between the two states by applying heat using electrical pulses. The optical and electronic properties can vary significantly between the amorphous and crystalline states; the difference and repeated switching allows data storage. Although related surveys of using phase change material to store information began in the 1960s, the technological success of optical storage based on phase change materials was only enabled after the discovery of a new class of materials that fulfilled all the requirements for this technology. It was found that semiconductor alloys along the pseudobinary line between $GeTe$ and Sb_2Te_3 had large optical contrast and could be rapidly and repeatedly switched between the amorphous, low-reflectivity and crystalline, high-reflectivity phases using laser pulses [Yamada et al. 1987]. $Ge_2Sb_2Te_5$ (GST) is the most widely studied phase change material. It can crystallize in less than 100ns [Raoux et al. 2008]. Phase change material is the core part of PCM technology, and its properties to a large extent determine its functionality and success.

2.1.3. Principles of PCM. In contrast to DRAM, PCM is a resistance-based memory technology. The principle of PCM is to store data via different resistances of PCM materials in two phases. The SET operation is achieved by crystallizing the material and RESET by making it amorphous. These two operations are performed by heating the material with different voltage and time [Dong et al. 2009a]. As shown in Figure 3, SET operations are achieved by applying moderate power for a long series of electrical pulses, while RESET operations are achieved by a short duration of high-powered electrical pulses. In addition, the SET operation tends to dictate the write speed performance of PCM technology, since the required duration of this pulse depends on the crystallization speed of the phase change material. The RESET operation tends to be fairly current- and power-hungry; thus, care must be taken to choose an access device capable of delivering high current and power without requiring a significantly larger footprint than the PCM element itself. The read operation is performed by measuring the device resistance at low voltage, which does not alter the state of the memory cell.

2.1.4. Design of PCM Devices. Device design and integration are key components of PCM technology, which determine the PCM functionality, production cost, and the size of the access device (storage density). Before we begin our design, we should make clear the influences of different parameters of phase change material on devices, which will help us choose the best phase change material for the devices. We list some important parameters of phase change materials and their influences on PCM device performance in Table I. Much effort has been devoted to optimizing phase change materials for specific applications with different requirements for memory functionality such as switching speed, data retention, endurance, and switching power. The failure mechanisms of PCM devices are also an important issue that we need to take into

Table I. Some Material Parameters and Their Influences on Device Performance (data from Burr et al. [2010])

Phase change material parameter	Influence on PCM device performance
Crystallization temperature	Data retention and archival lifetime
Thermal stability of the amorphous phase	SET power
Melting temperature	RESET power
Resistivity in amorphous and crystalline phases	On/off ratio, SET and RESET current
Threshold voltage	SET voltage and reading voltage
Thermal conductivity in both phases	SET and RESET power
Crystallization speed	SET pulse duration (thus power), data rate
Melt-quenching speed	RESET pulse duration (thus power)

consideration when designing PCM devices. One case is that the cell can no longer be switched to the low-resistance state; another is elemental segregation, in particular, *Sb* (antimony) enrichment in the switching region caused by electromigration, which leads to poor data retention when the cell can no longer be switched to the high-resistance state. Another important aspect is the cell design, which requires the consideration of many factors, such as number of required process steps, parameter window for each process step, availability of the required deposition methods (e.g., atomic layer deposition of phase change materials), and other aspects of manufacturability [Raoux et al. 2014].

2.1.5. Single-Level Cells and Multilevel Cells. As discussed earlier, phase change material can switch between two phases; thus, the PCM cell can store two possible values, that is, a single bit. This traditional use of a memory cell is called single-level cell (SLC). A PCM cell's feature of being in different degrees of partial crystallization enables more than one bit to be stored in each cell, which can increase the storage capabilities of a single PCM cell. Several prototypes have demonstrated the multilevel cell (MLC) [Burr et al. 2008; Lee et al. 2009]. For example, in a MLC design proposed by Lin et al. [2009], two bits are stored in a single MLC, effectively doubling the storage capacity of the memory for the same area cost. The MLC property can be used to multiply the memory capacity at the cell level. MLC PCM requires high current when performing a RESET operation, which shortens its endurance significantly. To relieve this problem, Jiang et al. [2012] proposed elastic RESET (ER), which reduces RESET current to construct non-2^n-state MLC PCM and effectively reduces the RESET energy so that the PCM endurance can be extended exponentially. Since the value written to the cell can drift over time, MLC PCM is prone to a unique type of soft errors. Seong et al. [2013] proposed trilevel-cell PCM and demonstrated its ability to achieve a $10^5\times$ lower soft error rate than four-level-cell PCM and 36.4% performance improvement over the four-level-cell PCM. The trilevel-cell PCM shows a $1.33\times$ higher information density than single-level-cell PCM while achieving the soft error rate of DRAM. Liu et al. [2014] observed that, by lowering the retention guarantee of PCM, one can obtain a significant write latency reduction. Based on this observation, they proposed a dual-retention PCM architecture that meets the durability requirement for persistent store while relaxing the retention requirement of working memory in return for write-latency reduction.

2.1.6. Summary of PCM Attributes. Table II shows the attributes of PCM by a comparison with other kinds of memory and storage technologies. Those data are cited from Qureshi et al. [2009], Mogul et al. [2009], Burr et al. [2008], Dong et al. [2008], Kryder and Kim [2009], and Lewis and Lee [2009]. Compared to DRAM and NAND Flash, emerging PCM devices have some unique characteristics as follows.

—*Scalability.* Without the need for large capacitors, PCM offers a density advantage over DRAM. IBM demonstrated a 20nm device prototype in 2008 [Raoux et al. 2008],

Table II. Comparison of Various Storage Techniques

	SRAM	DRAM	PCM	NAND Flash	Disk
Maturity	Product	Product	Advanced development	Product	Product
Cell size	$>100F^2$	$6\text{-}8F^2$	$8\text{-}16F^2$	$4\text{-}5F^2$	$(2/3)F^2$
Byte-accessible	Yes	Yes	Yes	No	No
Read latency	<10ns	10-60ns	48ns	25us	8.5ms
Write latency	<10ns	10-60ns	40-150ns	200us	9.5ms
Energy per bit access	>1pJ	2pJ	100pJ	10nJ	100-1000mJ
Standby power	Yes	Yes	No	No	Yes
Endurance (# of writes per bit)	$>10^{15}$	$>10^{15}$	10^8	10^4	$>10^{15}$
Nonvolatility	No	No	Yes	Yes	Yes

which is expected to scale down to 9nm in 2022 [ITRS 2007]. In contrast, DRAM probably will not be able to scale down beyond the 40nm technology [Raoux et al. 2008; Lee et al. 2009]. Given this scaling advantage and multiple bits per cell feature, PCM can enable more memory capacity for the same chip area, or potentially a lower price per capacity.

—*Performance.* PCM is byte addressable and has better performance than NAND Flash. Compared to DRAM, PCM's read latency is close, while its write latency is about an order of magnitude slower [Lee et al. 2009]. Since PCM can be written without Flash's expensive ERASE operations, its sequential and random access have similar performance. The SET latency is the longest and determines the write performance. The reason behind this difference is shown in Figure 3. Latencies of 150ns for SET and 40ns for RESET operations have been demonstrated. The read latency is 48ns.

—*Energy Consumption.* The dynamic power (power used to switch cell states) consumed by PCM is actually higher than DRAM. The read operation to a PCM cell dissipates 40μW. Write energy mainly depends on the RESET operation, which dissipates 480μW, while SET dissipates 90μW. Constant refreshes due to current leakage make DRAM power-consuming. The nonvolatile nature enables PCM not to require any refresh; thus, static power consumption becomes negligible when compared to DRAM.

—*Endurance.* Endurance refers to the limited number of writes that a PCM can sustain. The reason behind write endurance is that PCM writes cause the thermal expansion and contraction of PCM material, which degrades the contact between electrode and storage. Consequently, currents are no more reliably injected into the cell. Beyond the write endurance, the content in a PCM cell can still be read, but it can no longer be changed. aThe current write endurance of PCM varies between 10^7 (for MLC) and 10^9 (for SLC) writes, but we can conservatively assume 10^8 as a reasonable reference value [Lee et al. 2009; Zhou et al. 2009]. In other words, PCM's write endurance is orders of magnitude higher than Flash.

From this discussion, it should be clear that PCM has emerged as a leading contender to take the role of the next-generation memory technology. Impressive progress has been made in basic materials, device engineering, and chip-level demonstrations, including the potential for MLC programming and 3D-stacking [Kau et al. 2009]. When designing computer architectures and software systems for computer systems incorporating PCM, we can take advantage of PCM's good attributes of byte-addressability, low read latency, no power overhead in an idle period, and nonvolatility, and avoid PCM's disadvantages of high write latency, high-access energy consumption, and limited write lifetime.

2.2. Technical Challenges

Since the performance of PCM is between DRAM and Flash technologies (Figure 1) [Qureshi et al. 2009], which are dominantly used as main memory and solid-state storage, respectively, it is desirable to use PCM in main memory or in online storage. PCM combines the best features of DRAM (read and write bandwidths) with those of NAND Flash memory (such as retention); thus, it satisfies the many stringent requirements that are necessary for enterprise applications. This provides an opportunity for optimizing overall performance of a computer system. In addition, people are also seeking the solutions of applying PCM to processor cache.

When PCM is used in processor cache, main memory, and online storage, however, we also encounter different technical challenges. We need to seek architectural and system-level solutions before PCM can enable a revolution in next-generation computer systems. Incorporating PCM into which level of computer storage hierarchy is still an open issue. The main possible applications for PCM memory include being used as processor cache, main memory, solid-state storage, and as both main memory and storage.

2.2.1. Challenges When Used as Processor Cache. Traditional multilevel SRAM-based cache hierarchies have the drawbacks of low density, high-leakage power, and complicated design complexity, which lead people to seek solutions by using new memory technology (e.g., PCM) as an alternative or a supplement to SRAM. However, there remain a large amount of issues to be addressed before we employ PCM to cache hierarchy. The potential challenges when applying PCM as processor cache are as follows.

—*Performance Optimization*. When using PCM as processor cache, its performance is hard to match SRAM, especially its write performance (see Table II). Therefore, it is necessary to explore corresponding methods to optimize system performance as much as possible when employing PCM into cache systems.
—*Lifetime Prolongation*. The most stringent requirement for PCM to replace SRAM is write endurance. The read/write endurance for SRAM is infinite. The read endurance is not a likely problem for PCM. However, the required write endurance for PCM to replace SRAM is 10^{18}, which is out of reach for PCM (even all existing NVM technologies). Luckily, since PCM is nonvolatile compared with SRAM, we can exploit this feature to trade data retention for improved endurance. Further work is required in this direction before PCM can be applied to processor cache.

2.2.2. Challenges When Used as Main Memory. Due to the advantage of PCM in the capacity scaling capability over DRAM, PCM is considered a replacement or enhancement for DRAM. Before PCM can be declared a suitable successor or a supplement for DRAM, a lot of research is required to address the current limitations of PCM's access latency, write endurance, and energy consumption [Lee et al. 2009, 2010]. Accordingly, the following technical challenges are required to be addressed when PCM is used as main memory.

—*Performance Optimization*. When PCM is incorporated into main memory, its access times have to be considered. Current projections show that when serving read requests, PCM is about 2-4× slower than DRAM. The latency to write a line in PCM memory is more than an order of magnitude higher than in DRAM. This means that a PCM-based main memory system is likely to have higher access latency, therefore requiring optimization of overall system performance.

—*Lifetime Prolongation*. PCM cells are projected to endure 10–100 million writes [ITRS 2007]. After that, the content of PCM cells can be read, but cannot be changed. This means that the lifetime of a PCM system would be limited due to endurance-related wear-out. When not used wisely, PCM may become faulty after a short period of time.
—*Active Energy Saving*. The static power consumed by PCM is very low, but the dynamic energy consumption is obviously higher than DRAM. Writes especially consume significantly high energy, owing to the need to heat them in order to switch their state. Therefore, it is desirable to seek solutions to saving active energy consumption, especially for write-intensive workloads. PCM attributes provide some opportunities to achieve this goal. For instance, it allows writing of a single bit of PCM, while multiple banks are accessed for every DRAM write operation.
—*Nonvolatility Leveraging*. Existing main memory systems are dominantly built with DRAM, which is a volatile memory. Existing memory management designs are based on the assumption that the memory device is volatile. However, PCM devices are nonvolatile. This nonvolatility may provide some obvious benefits. For example, the time to boot a computer will become very short if PCM stores the operating system of a computer system. This requires that the memory management system can make a distinction between PCM and DRAM, and enable their advantages. Moreover, guaranteeing data consistency is a new challenging problem when exploiting nonvolatile PCM in main memory. We should consider how to obtain high throughput while providing strong consistency guarantees.

If the preceding technical challenges can be addressed effectively, PCM could be a successor to DRAM as the main memory of future computer systems. At least, PCM can be used in the DRAM-and-PCM hybrid architecture, in which DRAM is disadvantageous. On the other hand, the introduction of last-level cache (LLC) may help us alleviate these problems, especially when LLC becomes much larger. However, the benefit we can get from LLC mainly depends on the locality of applications and the used caching algorithm.

2.2.3. Challenges When Used as Solid-State Storage. Since PCM is nonvolatile, one can use it to build SSDs. Compared with Flash-based SSDs, PCM-based SSDs offer great advantages of high performance, byte accessibility, superior endurance, and low static energy consumption. PCM offers a viable alternative to flash memory for building SSDs.

However, existing software systems, such as file systems and database systems, are designed and optimized for hard disks. Some systems are optimized for Flash SSDs. Consequently, we need to rethink core algorithms and software implementations in online storage for PCM. When using PCM-based SSDs, the following technical challenges need to be addressed.

—*Leveraging Byte-addressability*. PCM is persistent and byte-addressable. PCM-based storage systems can write only a few bytes of data in places in which a traditional block-based file system would write kilobytes. A new challenge is understanding how storage software exploits the attribute of byte-addressability to eliminate write amplification, and further improving write performance and prolonging PCM lifetime.
—*Exploiting High-Access Performance Efficiently*. Some of existing OS software architecture techniques, such as page cache policy and the read/write granularity, are designed for the efficient use of traditional HDD and SSD. However, since the access latency of PCM is several orders of magnitude shorter than these traditional storage devices, we should rethink these techniques when applying PCM as the secondary storage.

—*Maximizing Performance Advantage with Limited Endurance and Size*. The most attractive advantage of PCM-based SSDs is their low access latencies. However, there are also some restrictions in the scenario in which PCM-based SSDs are used. First, the lifetime of a PCM system would be limited due to endurance-related wear-out. Second, due to the cost restriction, the size of PCM-based SSDs deployed in a computer system will also be limited. In this scenario, it is challenging to maximize PCM's performance advantage with the restriction of its limited endurance and size.
—*Optimizing Performance and Energy Efficiency Simultaneously*. With PCM, there is obvious disparity in performance and energy efficiency. First, PCM writes, compared to its read operations, incur higher energy consumption, higher latency, and lower bandwidth. Second, there are obvious asymmetries between two kinds of writes—the SET latency is over three times the RESET latency, while the RESET operation dissipates about 1.5 times higher energy. With those complex attributes of PCM, it is difficult to optimize overall system performance and energy efficiency simultaneously by designing data presentation and data layout wisely.

If these challenges are addressed, PCM may someday replace flash memory or at least some of its uses for the purpose of high performance. Until then, PCM could also be used in flash and PCM hybrid architectures to counter some of the disadvantages of flash memory.

2.2.4. Challenges When Used as Both Main Memory and Storage. From the preceding analyses, we can see that PCM can be used as main memory or solid-state storage. An intuitive idea is that, in a PCM-based system, a portion of PCM is used as main memory while the other portion is used as online storage. In such a PCM-based system, to make a satisfactory trade-off between performance and cost, those two portions will have to be managed in an integrated manner. Due to more complex architectures, some new challenges are to be addressed.

—*Dynamic Partitioning between Memory and Storage*. The integration of resource management will allow the system to flexibly provision the available PCM resources across the memory and storage boundary. It is challenging to design a dynamic scheme that changes the status of a portion of PCM from memory to storage, or from storage to memory for better performance and reliability.
—*Lifetime Enhancement*. Using PCM as memory and storage simultaneously, the system may crash more easily since memory wear-out and storage wear-out can both result in the problem.
—*Energy Efficiency Improvement*. Apart from the lifetime issue mentioned earlier, energy consumption is another problem to be solved. Although using PCM as storage will consume less energy than those traditional storage devices, the energy consumption of memory is still larger. Thus, further efforts should be made to reduce the energy consumption for the whole system.
—*System Overhead Reduction*. In a realization of the idea, the physical memory resource manager will need to bookkeep the status of the storage resources as well as the memory resources. The implementation of integrated management may introduce a new layer of resource control, which will incur system overheads. Thus, we need to tackle this problem carefully to get good system performance.

2.3. Other Resistive Memory Technologies

Here, we provide a brief introduction of some other resistive memory technologies, discuss their features, and compare them with PCM.

Spin transfer torque RAM (STT-RAM) utilizes a magnetic tunnel junction (MTJ) as memory storage. An MTJ consists of two ferromagnetic layers separated by an oxide

Table III. Comparison of Different Resistive Memory Technologies [Mittal et al. 2015]

	PCM	STT-RAM	RRAM
Cell size (F^2)	4–12	6–50	4–10
Write Endurance (# of writes per bit)	10^8–10^9	4×10^{12}	10^{12}
Speed (R/W)	Slow/very slow	Fast/slow	Fast/slow
Leakage Power	Low	Low	Low
Dynamic Energy (R/W)	Medium/high	Low/high	Low/high
Retention Period	N/A	N/A (unless relaxed)	N/A

barrier layer. The magnetization direction of one ferromagnetic layer is fixed, while that of the other ferromagnetic layer can be altered by passing a current. The resistance of the MTJ is determined by the relative magnetization direction of these two layers. If the two layers have different directions, the resistance of the MTJ is high and vice versa. Using this property, a binary value is stored in an STT-RAM cell.

A typical Resistive RAM (RRAM) design uses a unipolar switching. In this design, the SET and RESET operations are conducted by using short and long pulses, or by using high and low voltage with the same voltage polarity.

Table III lists the characteristics of different resistive memory technologies. As we can see from the table, STT-RAM has a lower density than PCM and RRAM, but its access latency and write endurance are much better than PCM. RRAM has a longer write endurance and shorter access latency than PCM and its density is comparable to PCM, but its operation energy consumption is larger than PCM. Based on these comparisons, we can conclude that none of the existing resistive memory technologies outperforms all the others on all parameters, which requires us to exploit the corresponding memory technologies according to the specific applications. For example, PCM is more appropriate to employ in main memory or online storage than STT-RAM and RRAM, while STT-RAM and RRAM are more suitable for the design of processor cache.

3. USING PCM IN PROCESSOR CACHE

Traditionally, SRAM has been used as processor cache, due to its desirable properties such as very high write endurance, low access latency, efficient dynamic energy, and manufacturability. However, SRAM also suffers from large leakage power consumption and low density. Thus, using SRAM to design caches consumes a significant fraction of chip area and power budget. The introduction of PCM to processor cache can help alleviate these SRAM problems, but it also introduces challenges of much longer access latency and limited endurance compared to SRAM. To make PCM as processor cache practical, these challenges need to be addressed.

3.1. Hybrid Cache Hierarchy

When we design cache hierarchy, one idea is that we can use multiple memory technologies to get the best of them. Although the physical material and read/write properties of different memory technologies are varied, the similarity in the peripheral circuits allows similar operation from a logic designer's point of view. Still, the limitations of hybrid designs are that, from the perspective of manufacturing, they may incur higher cost of integration, verification, and testing than homogeneous caches. Mittal et al. [2015] compared different hybrid caches and found that SRAM + PCM caches offer higher area efficiency than SRAM + eDRAM and SRAM + STT-RAM caches. However, an important limitation of SRAM + PCM hybrid caches is the large write latency gap between SRAM and PCM; hence, a hit in different portions of an SRAM + PCM hybrid cache may lead to vastly different latencies. Variable hit latency adds complications and unpredictability to scheduling of dependent instructions.

Guo et al. [2012] proposed Wear-Resistant Hybrid Cache Architecture (WRHCA), which splits L2 cache into two parts, that is, a small L2 SRAM cache and a large PCM cache. The goal of their design is to make use of PCM to build a cache with large capacity and low power consumption, while keeping low write traffic to PCM to extend its lifetime. They recorded the read and write times of a newly loaded cache line to predict if this line of data belongs to dead on load (DOL) data or write-intensive (WI) data. The DOL data will be evicted as soon as possible, while WI data are mapped to SRAM or written back to memory. Therefore, their scheme can reduce the number of writes to PCM, hence improve the lifetime and save energy.

Wu et al. [2009a] proposed a hybrid L2 cache architecture, which divides L2 cache into large-read (PCM) and small-write (SRAM) regions of different memory technologies. To fully explore performance enhancement, a proper cache line replacement and data migration policy between the two regions was proposed. To be more specific, they utilized a counter to record the access frequency of cache lines in the read region; when it exceeds a threshold, the line is swapped with another line in the write region. In addition, to save energy, they kept the read region in state-preserving low-leakage (drowsy) mode. Apart from the one-level hybrid cache hierarchy, Wu et al. [2009a] proposed interlevel hybrid cache hierarchies in which different memory technologies are applied in different cache levels according to their properties. L1 and L2 cache use SRAM due to its performance advantages. L3 cache is hybrid by using eDRAM, STT-RAM, or PCM, or another option is that L3 uses eDRAM or STT-RAM, and L4 uses PCM due to its slow speed and high density. The experimental results show that the intralevel hybrid cache hierarchies can achieve 18% IPC improvement and up to 70% reduction in power consumption compared to a pure SRAM cache hierarchy.

3.2. Pure PCM Cache Hierarchy

Joo et al. [2010] explored a set of techniques to prolong a pure PCM cache's lifetime, which enable the pure PCM as a drop-in replacement of an SRAM cache. They exploited a read-before-write scheme, which can eliminate redundant writes by performing a pre-read operation. Since a read operation is much cheaper than write, we can achieve high performance when there are many redundant writes. To further reduce the number of writes to PCM cells, they employed a data-inverting scheme in the PCM cache write logic, that is, a value will be written in an inverted form if it requires less numbers to be written. To solve the uneven write distribution in a cache block, they further proposed a bit-level shifting technique, which spreads out the writes over the whole PCM cells in a cache block. In this wear leveling scheme, a bit-line shifter is used to decide the number of bits by which input data is shifted before being written.

3.3. Summary

The works in applying PCM to processor cache systems mainly addressed the two main challenges, that is, poor performance and limited endurance of PCM compared to SRAM. As we discussed earlier, hybrid and pure PCM are two available cache architectures. Compared to pure PCM cache architecture, we think that hybrid cache architecture has greater potential, since it can get the best of different memory technologies to boost overall performance. Of course, pure PCM cache architecture is also worth exploring. The work of Joo et al. [2010] is just the beginning. We would like to see further work in this direction.

For performance reasons, PCM or some other new memory technologies are often applied in the lower-level cache. In addition, we can utilize the nonvolatile feature of PCM (e.g., data retention) to enhance performance.

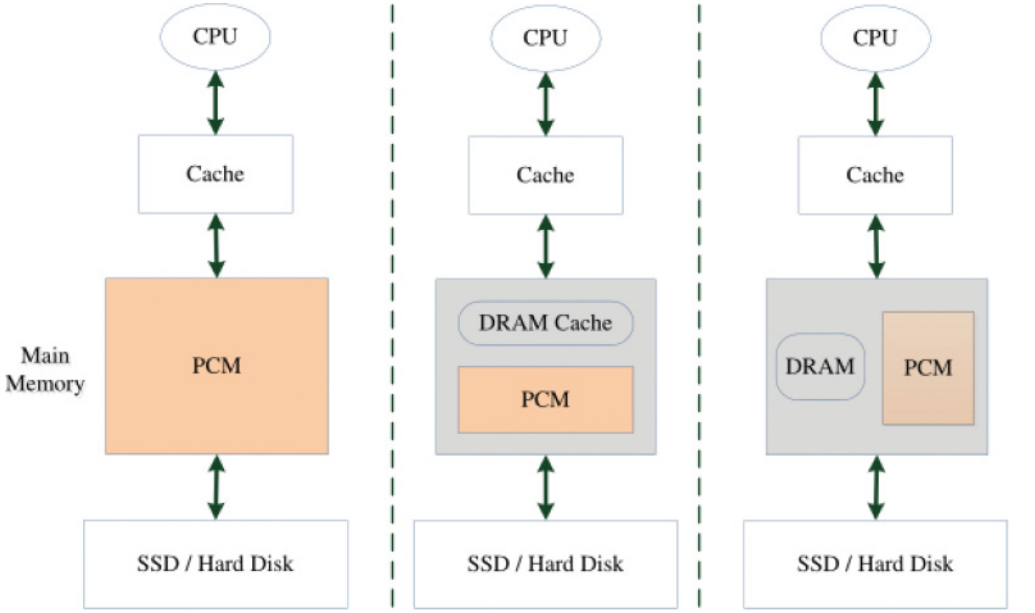


Fig. 4. Three alternative proposals for PCM-based main memory system (from Chen et al. [2011b]).

4. USING PCM IN MAIN MEMORY

To exploit PCM in main memory as a DRAM alternative, PCM must be architected to address its drawback of comparatively long latencies, high-energy writes, and finite endurance. The goals are that under representative workloads, read and write latencies cannot be too much longer than DRAM; energy consumption is identical to or lower than DRAM; and the lifetime can reach at least several years.

4.1. Performance Optimization

As shown in Figure 4, PCM-based main memory systems have three architectural alternatives:

—*PCM-only Main Memory Systems*. PCM is used to replace DRAM to achieve larger main memory capacity [Lee et al. 2009].
—*Hybrid Main Memory with Caching Architecture*. In addition to PCM, a small amount of DRAM is included in this kind of system so that frequently accessed data can be kept in the DRAM buffer, which can improve its access performance [Qureshi et al. 2009].
—*Hybrid Main Memory with Tiering Architecture*. PCM and DRAM are arranged horizontally to store different data, and can be directly accessed at the same level by users [Park et al. 2010; Mogul et al. 2009].

For each alternative, researchers proposed different approaches to optimizing the overall performance of main memory systems.

4.1.1. PCM-Only Main Memory Systems. Figure 5 shows the architecture of PCM array, which is similar to those for existing memory technologies. PCM cells in this architecture are hierarchically organized into banks, blocks, and subblocks with local, global

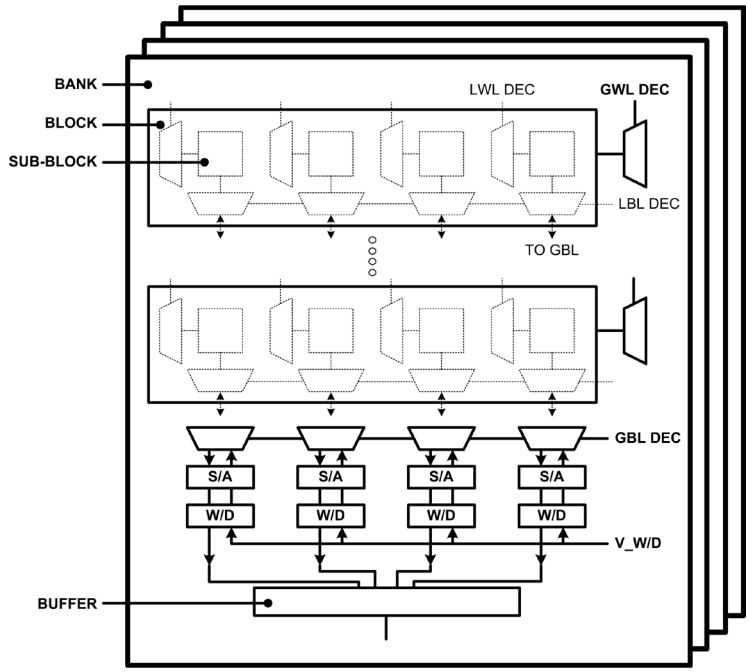


Fig. 5. PCM array architecture.

decoding for row, column addresses. Sense amplifiers (S/As) and word drivers (W/Ds) are multiplexed across blocks. Within the memory architecture, a row is in active state only when it is read from the array and latched in a buffer. In addition, memory access operations read data from and write data to the buffer directly. An access that requires an unbuffered row must evict the current row and read the desired row. Separate sensing and buffering enable multiplexed sense amplifiers. Local wordline decoders activate lines across multiple subblocks. A subset of these subblocks' data pass through local and global bitline decoders for sensing and buffering. This distributed bitline decoding enables buffer widths narrower than the total number of bitlines. The buffer width is a critical parameter in the design phase, which determines the required number of expensive current sense amplifiers.

Through the fundamental understanding of PCM technology parameters, Lee et al. [2009] proposed an area-neutral buffer reorganization approach that makes PCM a scalable DRAM alternative. They reorganized a single wide buffer into multiple narrow buffers to reduce both energy costs and access latencies. Narrow-buffer rows can mitigate write energy. However, narrow rows also negatively impact spatial locality, opportunities for write coalescing and, finally, application performance. The number of sense amplifiers decreases linearly with buffer width, significantly reducing the area since fewer of these large circuits are required. Considering area neutrality, narrower buffers mean more buffer rows. Multiple buffer rows can exploit locality for coalescing writes, hence hiding their latency.

Through experiments, Lee et al. [2009] found four 512B-wide buffers most effective for optimizing average latency and energy consumption across the workloads used in the experiments. A baseline PCM system is 1.6× slower and requires 2.2× more energy than a DRAM system. PCM buffer reorganizations reduce application execution time

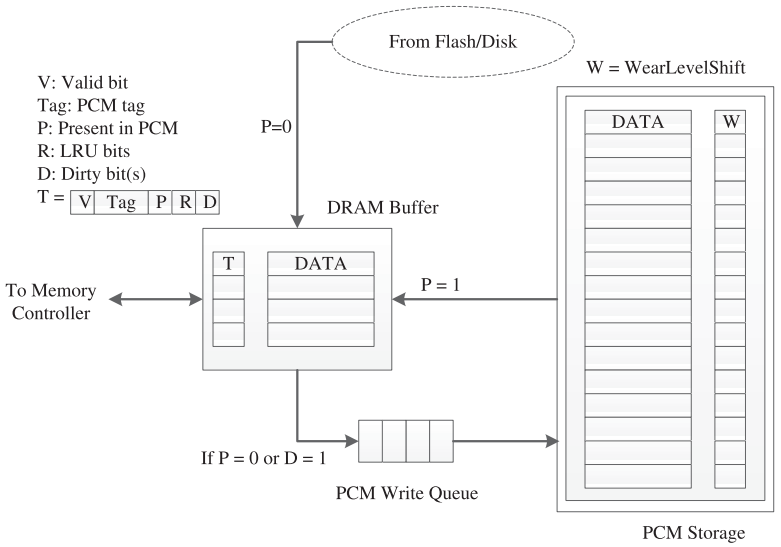


Fig. 6. The hybrid main memory of Qureshi et al. [2009].

from 1.6× to 1.2× and memory energy consumption from 2.2× to 1.0× relative to DRAM-based systems.

Due to poor performance under write-intensive workloads, the PCM-only main memory system should not be regarded as an independent solution. Instead, it should be used in a more complex memory architecture. It is necessary to redesign an entire hierarchy to overcome PCM shortcomings. The analysis in this survey is a step toward a fundamentally new memory hierarchy with deep implications across the hardware–software interface.

4.1.2. Hybrid Main Memory with Caching Architecture. Qureshi et al. [2009] proposed a simple architecture for a system using DRAM as a buffer for PCM. Much of the subsequent work assumes that the DRAM-boosted PCM main memory uses this architecture. In such an architecture (Figure 6), PCM is managed by the operating system via a page table. Being a buffer and an access interface of PCM, DRAM is managed by the DRAM controller, and is transparent to applications. The managing granularity of PCM and DRAM is one page. When reading a page, the system first checks it in the DRAM. If not found, the system retrieves it from PCM and even disks, then loads it into the DRAM. By increasing the size of a DRAM buffer, one can overcome the low-speed drawback of PCM. By performing experimental evaluation under different application workloads, Qureshi et al. [2009] pointed out that a memory system with 1GB DRAM and 32GB PCM has performance 3.0 times that of an 8GB DRAM memory system, while a memory system with 32GB DRAM has only 3.3 times performance.

The *lazy-write* technology can reduce the number of writes to PCM and improve performance. When fetching a page from a hard disk, the system writes the page to DRAM directly, while not writing it to PCM. A position is reserved in PCM. The page is written back to PCM on the eviction of the row from DRAM.

In order to endure high latencies of PCM writes, a write queue is introduced. With regard to the write speed of existing PCM, a write queue with a size of 100 pages is sufficient to avoid a write bottleneck [Qureshi et al. 2009]. In the traditional main memory, if a PCM write operation is being executed, the incoming read requests have to wait until this write operation is completed. The improved method is to process the

read operation first, while postponing the ongoing write. Once the read operation is completed, the frozen write operation will be continued according to the record in the write queue. In order to save energy, a threshold can be set adaptively. If the time that a write operation has spent, when a read request arrives, is higher than the threshold, the write will continue, rather than being frozen. Through the experiments, Qureshi et al. [2010] proved that this technology can reduce read latency by 75% and improve overall system performance by 46%.

Yoon et al. [2012] found that the miss rate of a buffer row can approximately reflect the role of the row on the performance. They proposed that the memory controller maintains a count of the row buffer misses for recently used rows in PCM. When the numbers of row buffer misses and accesses for a row exceed certain thresholds, the row is cached in DRAM. They used an adaptive method to adjust those thresholds.

4.1.3. Hybrid Main Memory with Tiering Architecture. In a tiering hybrid main memory system, PCM and DRAM can be directly accessed by the operating system. Joining of PCM increases the capacity of main memory. Compared with the preceding architecture of DRAM as the buffer, such an architecture can use the capacity of DRAM. But implementations are much more complex, some of which may generate a greater cost. Due to the difference in the capacity, performance, reliability, and volatility between DRAM and PCM, some intelligence will be required to decide what pages should migrate to PCM and when, according to the access pattern of the page.

Applying this intelligence, the OS appears to be the best place, which can choose the pages that have the best return on investment for migration. However, these pages should not be the kernel private address space that cannot be migrated easily, but should be the pages aimed for user address space and file system buffering, which consume most of DRAM.

A simple method is to use the segment type. The operating system allocates DRAM memory for the pages of the heap and stack segments, and PCM memory for other pages [Park et al. 2010]. This method performs better only when the ratio of DRAM and PCM is close to that of the heap and stack segments and other segments. Another solution [Mogul et al. 2009] is to use several heuristics. They include some tests that use static information, for example, page types, file types, file reference modes, and application-supplied page attributes. They also include some tests that use dynamic information, for example, file names and page history.

The preceding methods use all kinds of information from the operating system. Some other projects only take advantage of the information from the memory controller. A typical idea is to keep track of the access history of each page, and to write those pages with more accesses to DRAM. Dhiman et al. [2009] proposed recording the reference number of each page (using the SRAM cache to accelerate counting). When the write number of a page reaches the threshold, the page is migrated into DRAM. Ramos et al. [2011] proposed considering access number and access frequency comprehensively. They apply the multilevel LRU queues to promote the pages whose access number exceeds a certain threshold to the next queue. The pages of the several highest level are stored in DRAM.

In addition, it is worthwhile to note that if the data in PCM and DRAM are exclusive, the partial-write technology, mentioned in Section 3.2.1, cannot be used. When a page is evicted from DRAM into PCM, the system has to write a whole page even if the page has not been modified, which may increase the write number.

4.2. Lifetime Prolongation

Prolonging the PCM lifetime mainly consists of two technical ideas: reducing wear and wear leveling. Both techniques can be applied at the same time. The PCM lifetime has extended to a few years owing to the modern technology.

4.2.1. Reducing Wear on PCM. The amount of write times to a PCM unit is limited. Thus, decreasing PCM writes means reducing wear, which can be accomplished by changing the structure of the cache, using DRAM buffer (described in Section 3.1). There are also some other solutions, such as reducing the number of redundant writes, applications avoiding writes proactively, and so on.

Partial writes are useful to reduce redundant writes [Lee et al. 2009; Qureshi et al. 2009]. Main memory is divided into smaller blocks (e.g., 4B or 64B), and dirty bits are added for DRAM [Qureshi et al. 2009] or CPU [Lee et al. 2009] to trace whether a block is dirty. It synchronizes only small blocks that are dirty, instead of the whole line. This method reduces the number of writes, while it cannot eliminate redundant writes completely.

One solution to eliminating redundant writes completely is comparing every bit before a write and updating only changed bits [Zhou et al. 2009; Chen et al. 2011a]. It is implemented by adding an XNOR gate to compare new data with old data when writing. Yang et al. [2007] indicated that the extra reads for bit comparison bring only 1% performance overhead and 0.5% energy consumption. They proposed a data comparison write (DCW) scheme, which performs the read operation before the write operation to know the previously stored data in the selected persistent RAM (PRAM) cell. If the input data and the previously stored data are the same, no write operation is required. If not, the write operation is the same as the conventional write scheme. Based on DCW, Cho and Lee [2009] proposed Flip-N-Write, a simple but effective micro-architecture to improve PRAM's endurance. Its key idea is to replace a write operation with a read-modify-write operation to skip bit programming action if not needed (e.g., writing a "0" on "0"), and to limit the maximum number of bits to program by introducing a "flip bit" that indicates whether the associated PRAM word has been flipped or not. FlipMin [Jacobvitz et al. 2013] exploits coset coding to extend PCM's lifetime. The main function of coset coding is to perform a one-to-many mapping from each dataword to a coset of vectors. The fact of having various possible vectors provides the flexibility to choose the vector to write that optimizes lifetime. Compared with Flip-N-Write, FlipMin can further reduce the number of bits that flip per write. Qureshi et al. [2009] suggested that applications should proactively avoid writes to the main memory. For example, streaming applications with poor reuse do not benefit from the capacity boost provided by PCM. Thus, they added an extra bit called page level bypass (PLB), and assumed that the OS can turn on/off PLB for each application using a configuration bit. If the PLB bit is turned on, all pages of that application bypass the PCM storage to reduce wear. Wang et al. [2015] proposed a write-activity-aware page table management (WAPTM) scheme for PCM-based embedded systems, which aims to improve the system lifespan by reducing unnecessary writes in SLC PCM-based main memory. In contrast to compiler-based techniques, WAPTM utilizes architectural advances and is a pure OS-level technique. Their scheme includes two simple, yet effective, techniques: Smart-Init and Best-Fit. Smart-Init is mainly used to reduce bit flips when memory pages are accessed for the first time, and resets at most one bit in each page table entry (PTE), instead of 32 for 32b architecture. Best-Fit is an algorithm for allocating page frames, which requires the PTE as a parameter to find an appropriate free page frame and avoid bit flips in PCM cells when the process is running.

When a cache line is in dirty state and will not be used by the program again, we can view the cache line as useless. Bock et al. [2011] defined the write-back operation of the useless cache line as useless write-backs. Since the data will not be used again, the write-back operation can be safely avoided, thus improving the lifetime of PCM. SoftPCM [Fang et al. 2012] focuses on reducing write traffic of PCM in application-specific hardware for video applications. Based on the error tolerance characteristic of video applications, it relaxes the integrity of pixel data written into PCM. If the new

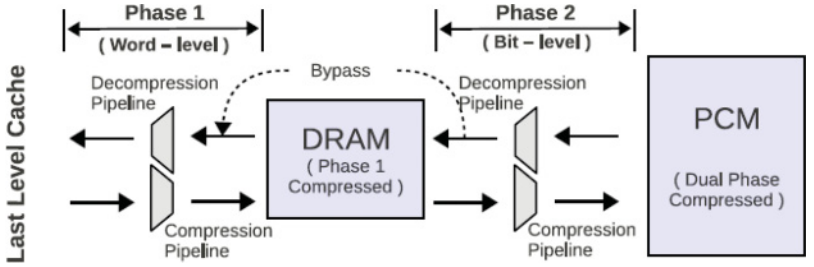


Fig. 7. High-level overview of the dual-phase compression technique (from Baek et al. [2012]).

data to be written is similar to the old, one can just ignore the write;thus, the lifetime of PCM can be improved.

Data compression is another kind of technique that can be used to reduce the write traffic to PCM. Sun et al. [2011] noted that the prior works such as DCW, Flip-N-Write, and so on, mainly based on the "bit-level" temporal locality. These methods find the redundant bits by comparing the value of the new data and the old. In some cases, however, these methods cannot work efficiently because they ignore the pattern of data. For example, two data, "11110000" and "11111111," are written into the same memory space and evict each other repeatedly. The bit-level temporal locality cannot be found in the last four bits. However, there exits the frequent-value locality in data, which can be exploited to reduce the writes. Baek et al. [2012] proposed the dual-phase compression technique (DPC) to improve the lifetime of hybrid DRAM/PCM main memory architectures. As shown in Figure 7, the DPC can be divided into two distinct phases. The first phase optimizes the DRAM cache accesses by utilizing a simplified, low-latency, word-level compression algorithm. The second phase exploits a bit-level compression algorithm to further reduce the number of PCM accesses. Delta-Compressed Caching [Du et al. 2013] can be used to reduce the writes of DRAM/PCM memory systems, which employs a new data-compression technique for modified data of DRAM cache and writes them back to PCM memory. Meanwhile, this kind of scheme can be applied to existing main memory compression frameworks. The experiments show that the delta compression technique reduces the number of PCM writes by 54.3%, and improves system performance and energy consumption by 24.4% and 11.0%, respectively, when given 14 representative memory-intensive workloads.

4.2.2. Wear Leveling. The key idea behind the method of wear leveling is changing the mapping from the virtual address space to the physical address space repeatedly, so that the write operations can be evenly distributed to the entire address space. Algebraic-based and request-based are the two kinds of classical algorithms on wear leveling. The obvious difference is that the former aims to design a general-purpose method that applies to any case, thus it does not take the access times of each block (a page or a cache line generally) into consideration; while the latter focuses on the dynamic wear leveling according to the access times of each block. When implemented, the latter has to maintain a reference table, which brings storage and access overheads; while the former needs to consider whether it can work well in the worst case.

Zhou et al. [2009] proposed an algebraic-based mapping algorithm called the *row-shifting* method. It shifts one byte in the current row every 256 writes. To achieve this goal, a circuit of row shifter and a register used to record the rotation offset should be added to the main memory array. Qureshi et al. [2009] proposed a fine-grained wear-leveling (FGWL) algorithm. Before writing a page, a pseudo-random number generator is consulted to get a random value as rotate number; all lines within the page are shifted by this rotate number. The rotate number is also stored in PCM, which can be

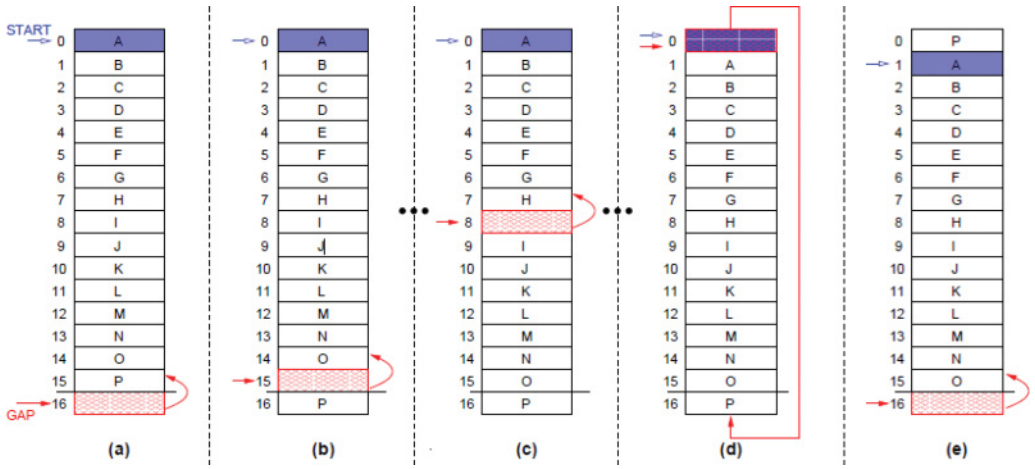


Fig. 8. Start-Gap wear leveling on a memory containing 16 lines (from Qureshi et al. [2009]).

referred to when the page is read. The implementation of the algorithm is simple, and balances writes within a page, but fails to keep the balance among pages. Intra-line flipping (ILF) [Zhao et al. 2014b] is a bit-level wear-leveling scheme, and the cost of this scheme is negligible. To swap the writes on hot and cold bits, ILF periodically flips the bit mapping in a memory line. Since the mapping is flipped in a regular manner without any counter or addressing mapping table, the flipping operation can be done very efficiently. ILF is furthermore compatible with many existing coarser-grained wear-leveling strategies, which can help us gain more endurance enhancement.

A typical optimization is Start-Gap Wear-leveling [Qureshi et al. 2009]. In this optimization, an extra spare memory line Gapline is added to the end of the main memory. Figure 8 shows an example of this scheme. Every Ψ=100 writes to main memory, Start-Gap moves one line from its location to a neighboring location, that is, the Gapline exchanges with the front line. When the Gapline reaches the start of the main memory, it exchanges with the last line so that the contents of all lines have shifted by exactly one location. By the movement mentioned earlier, the whole main memory can be rotated repeatedly, which can distribute the writes uniformly to the whole main memory. To solve the problem of heavily written lines being spatially nearby, they proposed two simple schemes for address-space randomization, which provides a (pseudo) random mapping of a given logical address to an intermediate address. In this way, all regions are likely to get write traffic very close to the average. The experimental results show that this Start-Gap scheme enhances the system endurance to 97%, which is an ideal condition. Curling-PCM [Liu et al. 2013] is a similar method applied first in embedded systems. The basic idea of the scheme is to periodically move the data of hot areas into the large cold areas within the PCM chip when given application-specific hot areas. Through this method, write traffic to hot areas can be evenly distributed so that the wear leveling can be achieved. FGWL [Qureshi et al. 2009] and Start-Gap [Qureshi et al. 2009] assume that PCM cells have the same endurance ability. However, Dong et al. [2011b] found that different PCM cells in the same chip may vary in endurance ability (perhaps due to nonuniform programming current), which is referred to as endurance variation; thus, it is obvious that balance writes will result in endurance degradation because the weakest cell retires earlier. Based on that, [Dong et al. 2011b] proposed wear rate leveling (WRL), a variant of wear leveling, which is used to balance wear rates (i.e., writes traffic/endurance) of cells across the PCM chip. Further, the

Max Hyperweight Rematching algorithm was proposed to co-optimize PCM lifetime and memory data migration overhead. Process variation refers to the variation in the attributes of transistors (e.g., length, width, oxide thickness) when integrated circuits are fabricated. With process variation, critical parameters shift from designed values. PCM cells require different minimum RESET and SET currents, therefore deliver various performance, energy, and endurance characteristics. Different from the approach of redistributing write operation, Zhao et al. [2014a] proposed SLC-enabled wear leveling (SEWL), a dynamic threshold-guided approach together with several SLC replacement policies to dynamically convert MLC pages to SLC mode to relieve risky cells from poor endurance and dense writes. In parallel, for the rest of the MLC pages, swapping was conducted to balance write operations.

The segment-swapping algorithm proposed by Zhou et al. [2009] is request-based. In this scheme, memory segments of high- and low-write accesses are swapped periodically. This process is implemented in the memory controller, which keeps track of each segment's writes counts and mapping table between the virtual and physical segment numbers. The optimal segment size is 1MB, and the optimal swap interval is every 2M writes in each segment. The energy overhead of the method is comparatively high, however. Dhiman et al. [2009] introduced a wear-leveling algorithm based on the page write counts to their PDRAM memory system. To help level wears of the PCM, the memory controller maintains a table of the number of write accesses. To reduce time and energy overheads for updating PCM access tables, they introduced a small SRAM-based cache in the controller, which caches the updates to the table, hence reducing the consequent number of PCM accesses. If the number of writes to any PCM page exceeds a given threshold, the controller generates a "page swap" interrupt to the processor, and allocates a new page from the DRAM allocator to the new page, then copies the contents of the old page to the new one. When the threshold equals to one, copying the page that is about to be written to the DRAM is something similar to DRAM buffer [Qureshi et al. 2009]. The difference lies in that the latter has to load the page into PCM whenever reading a page. If the threshold is set higher, the cost of copying the whole page can be avoided for just several accesses.

Hu et al. [2013] proposed software-enabled wear-leveling techniques in embedded systems, which are similar to Dhiman's. Their Software Wear-Leveling (SWL) algorithm is a polynomial-time algorithm and does not have hardware overhead. The idea of SWL is to use the Optimal Data Allocation (ODA) algorithm to generate the data allocation for each region of a program first. Then, one will know which variables are allocated into PCM in each region. SWL uses an array to keep track of the number of writes on each address of the PCM. With the help of this array, one can find an address assignment in PCM for each variable so that the number of writes on each address is less than or equal to the threshold Θ and the total moving cost is minimal. It comes naturally from the thought of an algorithm based on write accesses. However, since it is necessary to maintain a write counter per page, the overheads in performance and storage are comparatively high. The data-access pattern in embedded systems is fixed. Based on this observation, Long et al. [2014] proposed a space-based wear leveling at the software compiler level to enhance the lifetime of PCM-based embedded systems. Their key idea is to transform write traffic from a frequently written variable into an array, so that a large number of writes to the same memory location is evenly distributed to multiple memory locations.

To ease the memory pressure of smartphones, Zhong et al. [2014] implemented NVM-Swap, which builds a swap area for smartphone systems with NVM. Specifically, they proposed copy-on-write swap-in (COWS), so that read requests can get data directly from the swap area with zero memory copy. When a swapped-out page is accessed again, COWS sets up the page table mapping and returns the page in NVM directly, without

first copying the page out of the swap area. When a write happens to that page, COWS does actual swap-in by copying the page from NVM to DRAM. In addition, to make the NVM-Swap practical, Heap-Wear was introduced to distribute pages evenly across the whole NVM space, so that the lifetime of NVM-Swap can be further improved. Zhong et al. [2015] found that traditional wear-leveling techniques are not applicable to smartphone applications, which have vastly different data-access patterns. They noted that identifying a code page is easy, and that a code page is read only, making it a perfect candidate for swapping to write-limited NVRAM. Thus, they proposed nCode, which exploits NVRAM's byte addressable to execute the code page directly in the swap area, without being swapped back to main memory, further improving the endurance of NVRAM.

Fan et al. [2014] proposed WL-Reviver, a framework that allows any in-PCM wear-leveling scheme to keep delivering its designed leveling service even after failures occur in its working address space. WL-Reviver is a lightweight framework with very low overhead, which can efficiently revive a wear-leveling scheme without compromising the wear-leveling effect of this scheme.

4.2.3. Preventing Malicious Wear-Out. Traditional wear-leveling algorithms take only typical applications into account. However, a malicious application may constantly write to the same place in PCM, hence wearing out the PCM units in a short time. Some work [Seong et al. 2010a; Seznec 2010] implied that (1) using traditional algorithms, such as Start-Gap [Qureshi et al. 2009] and Segment Swapping [Zhou et al. 2009], the PCM can endure only a few minutes or several dozen hours; and (2) changing the mapping between the physical address and the physical PCM periodically can hide the real address, which helps to ease the problem. They [Seong et al. 2010a; Seznec 2010] also proposed the corresponding address remapping algorithm, respectively.

Security Refresh [Seong et al. 2010a] divides the memory into several subregions. The subregion index, the real address, is the result of the memory address XOR a value k, which is produced by a random number generator periodically. Every certain number of writes, the data of the block is moved to the new address, which was generated by XORing its corresponding memory address with a key k. After several operations, the whole region will be mapped to the new address generated by the operation of XORing. Another scheme [Seznec 2010] is similar to Security Refresh. They both can help transform the traditional wear-leveling algorithms into the algorithm that is able to resist malicious attacks.

Qureshi et al. [2011] further pointed out that the reason why Start-Gap wear leveling is attacked is its slow shift speed. It will not be so vulnerable to malicious attacks, even if we just accelerate the speed (reduce the value of Ψ). They found that the percentage of the number of most frequently written lines to the number of total writes within a period of time (attack density) can be used to distinguish malicious attacks from normal applications. In addition, they used the information of attack density for implementing Adaptive Wear Leveling (AWL) algorithms. In the normal case, AWL incurs negligible write overhead and can endure attacks for several years.

4.2.4. Tolerance to Unit Fault. In reality, lifetimes of different PCM units are not strictly the same, but they follow a certain distribution. Moreover, some units have a very short lifetime. Meanwhile, the lifetime of a system depends on the unit that has the shortest lifetime, which makes the real lifetime of a system shorter than the theoretical lifetime. On the other hand, since there must be some units that finally wear out, none of the wear-leveling algorithms will work perfectly. Hence, it is necessary to keep the system working normally, even if some of the units were worn out.

Conventional fault-tolerance covers soft faults and hard faults. Soft faults are the result of changing temperature, which makes the data stored change over time. But

this kind of fault occurs rarely in PCM. The cause of hard faults is that, when the write times exceed their limits, the PCM cannot be written any more, due to the material changes. Since soft faults seldom occur, we mainly discuss how to solve hard faults, as follows.

When hard faults happen on PCM, although we cannot rewrite the data, we can still read the data correctly. Hence, the read-write-read method can be used to detect hard faults [Schechter et al. 2010]. Based on this method, error-correcting pointers (ECPs), which record the error address and the real value to correct the faults, will incur lower overheads and longer lifetime than the traditional ECC [Schechter et al. 2010]. Six ECP entries per 512B can endure 6 errors. To save space, Qureshi [2011] attached only one ECP per block, using a global Hash list to record other ECP. Qureshi's method (Pay-As-You-Go) reduces the storage overhead of error collection by a factor of 3.1 compared to ECP-6, while still obtaining 13% longer lifetime.

Yoon et al. [2011] used fine-grained remapping with ECC and embedded pointers (FREE-p) to protect against both hard and soft faults. Furthermore, they proposed fine-grained remapping, which means a smaller block size (64B). When the number of errors exceeded the ECC's tolerance per block, they remapped blocks with a stored pointer to the new block. Note that the old block is not completely broken, the unbroken units can be used to replicate block pointers (i.e., 8b pointer will be replicated 8 times). The advantage of the method is that it can be implemented completely in memory controller, which can endure both hard and soft faults. Zombie [Azevedoy et al. 2013] allowed an error-correction mechanism to reuse good blocks (spare blocks) in disabled pages by pairing them with the blocks (primary blocks) in working pages. In this way, the spare block increases the error-correction resources of its primary block and keeps it alive longer. Moreover, in Zombie, there are two new error-correction mechanisms, ZombieMLC and ZombieXOR. ZombieSLC mechanisms use existing schemes for intrinsic block error correction. ZombieMLC is a new encoding mechanism, which can tolerate both stuck-at failures and drift.

Former wear-leveling and salvaging schemes have not been designed and integrated to work cooperatively to achieve the best PCM device lifetime. Simple integration of these approaches (e.g., Schechter et al. [2010]) will result in a noncontiguous PCM space. Jiang et al. [2013] proposed LLS, a Line-Level mapping and Salvaging design, which integrates wear-leveling and salvaging schemes, and copes well with modern OSes. Their scheme masks lower-level failures from OSes and applications by allocating a dynamic portion of total space in a PCM device as backup space, and mapping failed lines to backup PCM; meanwhile, LLS will construct a continuous PCM space.

The works mentioned earlier mainly tackle the problem of recovering from transient faults, and are not suitable for stuck-at faults. In addition, a failed cell with a stuck-at value is still readable. Based on this attribute, SAFER [Seong et al. 2010b] utilizes the failed bit to store data, which reduces the hardware overhead for error recovery. SAFER partitions a data block dynamically while ensuring that there is at most one fail bit per partition, and uses single-error correction techniques per partition for fail recovery. Further, SAFER needs to repartition the block if it contains two fail bits. Still, SAFER's space efficiency and fault-tolerant capability are not so desirable. Fan et al. [2013] proposed Aegis, an improved recovery solution with a systematic partition scheme using fewer groups to accommodate more faults compared with SAFER. Based on the observation that on a Cartesian plane any two different points on a line uniquely determine slope of the line, we can keep at most only one point on the original line to stay on a new line of a different slope when we change the slope of the line. Figure 9 shows an example of their partition scheme. The uniqueness of Aegis's partition scheme lies on the repartition scheme, which separates any two faults in the same group into

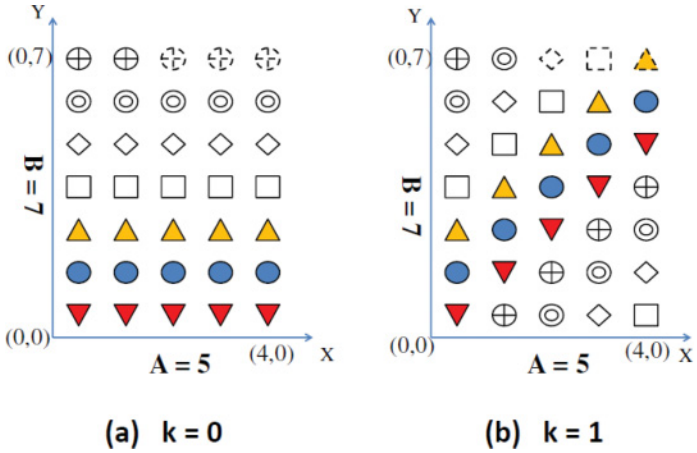


Fig. 9. Illustrating how bits in a 32b data block are partitioned into 7 groups, each of 5b. For (a), the partition configuration adopts slope k = 0, and (b) represents a different configuration using slope k = 1. In total, there are 7 configuration in the partition scheme defined by the A × B rectangle (5 × 7). The bits, each represented by a symbol, are mapped into a Cartesian plane. As the (5 × 7) rectangle contains three more positions than the 32b in the block, the three dotted symbols on the top right are unmapped. Different symbols are used to distinguish bits in different groups (from Fan et al. [2013]).

two different groups and helps to evenly distribute faults in a block across different groups, hence, promoting wear leveling within each block.

4.3. Active Energy Saving

4.3.1. Reduction of the Number of Writes. The main energy consumption of PCM comes from write operations. By reducing write times, energy consumption will be saved. Some optimization methods (e.g., changing cache structures, using DRAM buffers described in Section 3.1) can reduce write times. The ways to reduce the total number of wear, described in Section 3.2.1, include reducing redundant writes and adding a PLB switch. Refer to the earlier sections for more details.

4.3.2. Out-of-Position Writes. Chen et al. [2011a] noticed that RESET and SET operations need different power. Generally, the RESET operation requires about 1.5 times higher power than SET. Thus, it is meaningful to reduce as many as possible RESET operations. They pointed out that, compared with writing in position, it is more desirable to pick a free block to write in. To determine a good out-of-position PCM block, they devised a locality-sensitive hashing (LSH)-based algorithm, which can reduce RESET operations and get an additional 22.9% power savings.

4.3.3. Mapping and Buffering Data in MLC PCM. MLC PCM can store multiple bits per cell, which offers higher bit density. However, this kind of feature incurs higher latency and energy consumption of a memory access. Wang et al. [2011] noted that there are significant value-dependent energy variations in programming MLC PCM, for example, for a 2b MLC PCM, the average energy consumption for state “00,” “01,” “10,” and “11” are 36PJ, 307PJ, 547PJ, and 20PJ respectively. Thus, they proposed an energy-efficient PCM architecture using data encoding write to reduce system overall energy consumption. Furthermore, they adopted DCW to enhance the effectiveness of their scheme. Yoon et al. [2013] proposed a new approach to mapping and buffering data in MLC PCM to improve the performance and energy efficiency of a memory system. The latency and energy consumption to read or write to MLC PCM vary depending on the resistance state of the MLC. They exploit this asymmetry between the different

bits by decoupling the bits and mapping them to logically separate memory addresses. The method improves system performance by 19.2%, and memory energy efficiency by 14.4% over the state-of-the-art MLC PCM baseline system, which does not employ bit decoupling. Zhao et al. [2015] found that dynamic state remapping (DSR) generates locally optimal remapping decisions without considering the similarity between the new data and the old data stored in the memory. Therefore, when redundant write elimination (RWE) is applied, DSR may not be optimal, since it possibly introduces more transitions and more energy consumption. Based on that, they proposed a profiling-based state remapping scheme for MLC PCM, which can achieve 10.6% energy reduction, on average, within negligible hardware and performance overhead.

4.4. Consistency

While we can take fullest advantage of low latency and high bandwidths of PCM, we also face challenges of not sacrificing data reliability and consistency that users demand from storage. In particular, we need certain mechanisms to recover data to its last consistent state when a system failure or a crash occurs. In general, there are issues that need to be addressed by software for correctly implementing consistency.

4.4.1. Persistence Ordering. Updating persistent data structures imposes additional constraints on the ordering of statements due to the possibility of failure at arbitrary points in the program. Note that persistence ordering requires that the updates must be propagated all the way to the persistent memory in the specified order. It is not sufficient to just order the global visibility of these updates as in typical memory consistency protocols. Additional hardware support may be needed to ensure this memory behavior.

BPFS [Condit et al. 2009] exploits a new mechanism called *epoch barriers* for ordering of updates. In its scheme, a cache line is tagged with an epoch number, and the cache hardware is modified to guarantee that memory updates due to write-backs are always done in epoch order. In Mnemosyne [Volos et al. 2011], a combination of noncached write mode, cache-line flush instructions, and memory barriers was used to address the correctness of ordering of writes. Moraru et al. [2013] proposed a cache-line countermechanism, which is more general and flexible. In their scheme, applications themselves control the ordering of updates by delaying making those updates that depend on something still in the CPU caches, which requires less intrusive hardware support compared with BPFS's epoch barriers. Venkataraman et al. [2011] provided a software primitive flush, which combines with tracking recently written data to ensure the order of writes. In Mnemosyne [Volos et al. 2011], a novel implementation of persistent logs based on ordering primitives has been proposed. Doshi and Varman [2012] used ordering primitives sparingly, and only to ensure that the update trail of a transaction has been logged to a power-safe region of persistent memory before the transaction commits. The persistent log structures in Mnemosyne can be simplified and adopted in their scheme. Yang et al. [2015] observed that the operations to maintain memory writes to NVM in a certain order incur significant overhead; thus, they proposed NV-Tree , a consistent and cache-optimized B+Tree variant that reduces CPU cache-line flush for keeping data consistency in NVM. Specifically, NV-Tree decouples tree nodes into two parts, leaf nodes (LNs) as critical data and internal nodes (INs) as reconstructable data. By selectively enforcing consistency, adopting unsorted LN and organizing IN cache-optimized layout, NV-Tree can reduce the number of cache-line flushes under write-intensive workloads by more than 90% compared to CDDS-Tree [Venkataraman et al. 2011].

4.4.2. Persistence Atomicity. Transactional semantics require that updates to a set of related records must always occur as a group; either all the records must be updated or none should be updated.

BPFS [Condit et al. 2009] uses short-circuit shadow paging to keep atomic updates for tree-structured file systems. A copy-on-write scheme was used for the update of the blocks. In Mnemosyne [Volos et al. 2011], a software transactional memory (STM) system was exploited to control the executing applications to ensure the atomicity. Versioning is also a good way to handle atomicity. Consistent and Durable Data Structures (CDDS) [Venkataraman et al. 2011] designs a persistent multiversioning B-Tree that maintains several versions of a database at any instant. Every update to the data structure results in the creation of a new version. After all the modifications for an update have been made persistent, the most recent consistent version number is updated atomically.

4.4.3. Persistence Protection. Programming bugs in a persistent memory system can be insidious. Not only does the persistent nature of the changes make it impossible to simply reboot to a consistent memory state, but subtle pointer dependencies between data structures spread over volatile and nonvolatile memory regions of memory, increasing the challenge of robust programming tremendously [Coburn et al. 2011].

This robust persistence has been addressed in NV-Heaps [Coburn et al. 2011], and more corresponding work on this issue has been studied recently.

4.5. Summary

In this section, we review the works on designing PCM-based memory systems, which try to enable PCM to be a good alternative in main memory designs. These works aim at addressing the challenges from the perspective of performance, lifetime, energy, and consistency.

When applying PCM to main memory, there are three widely used architectures—PCM-only memory architecture, caching architecture, and hierarchical architecture. For PCM-only memory architecture, to overcome its drawbacks of long-access latency and high-energy consumption, a design with multiple buffer rows can be introduced to this architecture. For hybrid main memory with caching architecture, some caching optimization technologies are usually used to enable DRAM to filter frequent data access to PCM. For example, the lazy-write technology is able to reduce the write operations to PCM and improve overall performance. In hybrid main memory with tiering architecture, we need OS supports to place data wisely between DRAM and PCM according to data-access patterns for better performance. Currently, DRAM and PCM hybrid memory systems are better than those PCM-only memory systems, because we can utilize the strengths of both DRAM and PCM. In the future, it is unclear which architecture of the three is the best. In any event, we can choose the corresponding architecture according to our given application.

In Tables IV, V, and VI, we summarize the techniques used to improve the lifetime of PCM main memory systems. As shown in Table IV, there are three categories of methods for reducing the write operations to PCM. The first is to reduce the unmodified writes to PCM. This can be implemented by hardware, such as cache, memory controller, and chip. The second is to reduce useless data writes to PCM. This requires the support of the OS, and is usually harder to implement than reducing unmodified writes. SoftPCM is an application-specific method, which exploits the feature of video application to reduce PCM writes. The third is to compress the data to reduce the writes to PCM. Among those works, DPC will compress all the data to be written to PCM, while the other two compress the data selectively and more efficiently.

Table IV. Comparison of Various Techniques of Reducing Writes

Scheme	Key idea	Implementation
Partial writes	Reduce unmodified writes	Cache
LLWB	Reduce unmodified writes	Cache
DCW	Reduce unmodified writes	Chip
Flip-N-Write	Reduce unmodified writes	Chip
FlipMin	Reduce unmodified writes	Memory controller
PLB	Reduce unmodified writes	Cache
WAPTM	Reduce unmodified writes	OS, memory controller
Useless cache line	Reduce useless data writes	OS, memory controller
SoftPCM	Reduce useless data writes	Memory controller
Frequent-value locality	Data compression	Memory controller
DPC	Data compression	Cache, memory controller
Delta-compressed caching	Data compression	Cache

Table V. Comparison of Various Techniques of Wear Leveling

Scheme	Key idea	Random	Granularity	Implementation
Row shifting	Random shift within a line	Yes	Word	Memory controller
FGWL	All lines within the page are shifted by a random number	Yes	Line	Memory controller
ILF	Flip the bit mapping to swap writes on hot and cold bits	No	Bits	Memory controller
Start-Gap	Move one line from its location to a neighboring location	Yes	Line	Memory controller
Curling-PCM	Move the data of hot areas into the cold areas	No	Block	OS, memory controller
WRL	Distribute writes according to cell's wear rate	No	Word	OS, memory controller
SEWL	Convert MLC pages to SLC mode dynamically	No	Page	OS, memory controller
Segment swapping	Swap memory segments of high- and low-write accesses	No	Segment	Memory controller
PDRAM	Page write counts combine with memory management	No	Page	OS, memory controller
SWL	Use the ODA to generate the data allocation	No	Page	OS, memory controller
Space-based wear leveling	Transform writes from a frequently written variable into an array	No	Word	Compiler
NVM-Swap	A swap area for COWS in smartphone	No	Block	OS, memory controller
nCode	Swap code page in smartphone	No	Block	OS, memory controller
Security Refresh	Remap data block using random keys	Yes	Block	Memory controller
AWL	Use OAD to detect attack	No	Line	Memory controller

Table VI. Comparison of Various Techniques of Tolerating Fault

Scheme	Key idea	Reuse	Granularity	Implement	Overhead
ECP	Replace bad block with new	No	Block	Hardware	61b/block for ECP6
Pay-As-You-Go	Replace bad block with new	No	Block	Hardware	19.5b/block
FREE-p	Use bad blocks pair with well	No	Block	OS, Hardware	64b/block
Zombie	Use bad blocks pair with well	Yes	Block	Hardware	-
LLS	Replace bad block with new	No	Block	Hardware	61b/block
SAFER	Partition and inversion	No	Block	Hardware	55b/block
Aegis	Partition and inversion	No	Block	Hardware	27b/block

Table VII. Comparison of Various Techniques of Consistency

Work	Key idea	Main tackled issues
BPFS	Epoch barriers, short-circuit shadow paging, and copy-on-write	Persistence ordering, Persistence atomicity
Mnemosyne	Noncached write mode, cache-line flush instructions, memory barriers, and STM	Persistence ordering, Persistence atomicity
Cache-line counter	Applications control the ordering of updates by delaying making those updates that depend on something still in the CPU caches	Persistence ordering
Primitive flush	A combination of flush with tracking recently written data	Persistence ordering
Ordering primitives	Ensure the update trail of a transaction logged to a power-safe region of persistent memory before it commits	Persistence ordering
NV-Tree	Exploits a consistent and cache-optimized B+Tree variant	Persistence ordering
CDDS	Implements a persistent multiversioning B-Tree	Persistence atomicity
NV-Heaps	Provides a set of primitives, automatic garbage collection, and pointer safety	Persistence protection

Table V lists the works of wear leveling on PCM from the aspects of key idea, granularity, randomness, and implementation. These schemes can be divided into algebraic-based wear leveling and request-based wear leveling. Request-based methods are usually easy to implement, but introduce the overhead of storing the write times, while algebraic-based methods can reduce those overheads. Moreover, the granularity of wear leveling is another key factor that affects the performance. Generally speaking, coarse-granularity usually has low implementation overhead, but its global wear-leveling performance is low. This is contrary to fine-granularity wear leveling. Therefore, the combination of two granularities may be a better choice and deserves deeper study.

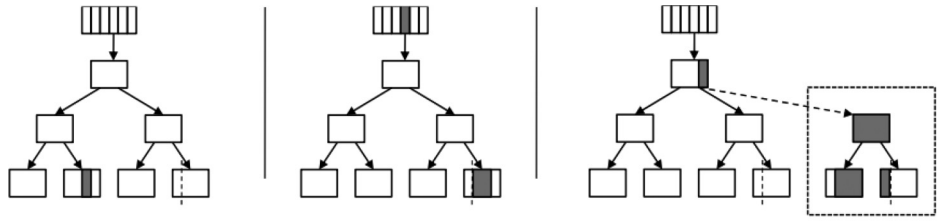
Table VI lists the fault-tolerant schemes to improve the lifetime of PCM. To achieve this goal, two kinds of methods can be employed: using well bits to correct the fault bits, and reusing the well bits in fault blocks. As we can see from Table VI, Pay-As-You-Go and Aegis are good choices to tolerate fault bits since their overheads are the lowest. For researchers, high-memory utilization and low extra bit overhead for fault tolerance is a research direction.

When reviewing works in memory energy saving, it is important to note that the introduction of PCM to main memory will bring great energy consumption. This is because PCM requires more energy for a single-write operation. Thus, reducing the writes to PCM can save energy. Since the RESET operation needs 1.5x higher power than the SET, we can try to exploit this feature to save energy. Further work can be carried out in this direction to reduce system energy consumption.

Table VII lists works aiming at ensuring data consistency while reducing corresponding overhead. The nonvolatile nature of PCM makes it easy to realize persistent data structures when power failures and system crashes occur. However, without careful designs, data may be inconsistent after the recovery from failures. Therefore, consistency is still a challenging problem when we exploit PCM as main memory. Although a lot of approaches have been proposed to solve this problem, it still deserves deeper study in the face of reducing consistent cost.

5. USING PCM IN ONLINE STORAGE

PCM's features of bitwise access and high speed make its performance better than conventional hard disks or Flash SSDs. Moreover, the emergence of PCM poses new challenges to traditional file systems, database systems, and so on. It is well known that those systems were originally designed for the use of HDD. Some have been optimized for Flash-based SSDs. Thus, they are required to be reconsidered for the introduction of PCM devices.



(a) in-place write (b) in-place append (c) partial copy-on-write

Fig. 10. Atomic updates in BPFS (from Condit et al. [2009]).

5.1. File Systems

The performance will be better by getting a HDD/Flash hybrid with PCM; the systems composed of PCM only perform much better than those composed of HDD or Flash. To make full use of device attributes of PCM, it is necessary to design different file systems specifically for different applications.

PFFS [Park et al. 2008] is a file system composed of a hybrid of PCM and Flash, which maintains its metadata in a small segment of PCM. There are two advantages in the design. First, PCM is accessed by bit addressing, which can reduce the wear overhead of updating metadata greatly. Second, there is no need to scan entire an Flash drive at mounting time. The reason is that any file can be found via PCM data; thus, the mounting time is constant. Furthermore, it solves the problem that the time of mounting a Flash drive increases linearly with the capacity, which provides a scalable Flash file system. The experiments show that the performance of PFFS for small files is 25% better than that of the file system YAFFS2, which uses Flash only, and the complexity of mounting time and memory footprint are both $O(1)$.

BPFS [Condit et al. 2009] is a file system designed for PCM, which enhances the performance and reliability of the file system, and brings new supports for the applications. The operations of the file system can be executed atomically and in program order; the data will be persistent as long as the cache is refreshed. The key technology of BPFS is short-circuit shadow paging, which takes advantage of in-place modification and atomic 64B writes to reduce the replications of shadow page. Traditional shadow paging needs to copy all the nodes from modified locations up to the root of the file system tree for every write, but since PCM is bit addressable, operations related to one block can be completed in place (Figure 10(a)(b)). As for general case, it can also copy all portions modified up to a recent common ancestor, then commit this change by performing an atomic update of a pointer (Figure 10(c)). In addition, to make sure that the write operation is executed in order and in granularity of 64B, some hardware is needed. The experiments show that the performance of BPFS on PCM is typically twice as high as NTFS on PCM, and 4 to 10 times higher than NTFS on hard disks. However, BPFS needs dedicated hardware supports. The main drawbacks of BPFS is that BPFS cannot update in place even for path nodes if the number of pointers to be updated within a path node becomes larger than the atomic update unit.

Existing versioning file systems do not perform well with the emerging PCM storage. Specifically, a large amount of additional writes, incurred by maintaining snapshots, degrade the performance of PCM seriously as write operations are the performance bottleneck of PCM. B. C. Lee et al. [2013] presented On-Demand Snapshot, a new versioning file system designed for PCM, which supports large write and reduces the writing overhead of maintaining a snapshot significantly. Unlike existing versioning file systems that incur cascaded writes up to the file system root, their scheme breaks the recursive update chain at the immediate parent level. The experiments show

that the On-Demand Snapshot file system improves the I/O throughput by 144%, on average, compared to ZFS.

SCMFS [Wu and Reddy 2011] is a file system designed on Storage Class Memory (SCM), which can be attached to the memory bus directly, thus reducing the latencies to access persistent storage. Moreover, we can simply access persistent storage through memory load/store instructions. However, the overheads caused by the file system layer itself is much lower than that of I/O latency; thus, its performance mainly depends on device attributes and the performance of the I/O scheduler. In this case, the storage device will share some critical system resources with the main memory, such as the memory bus, CPU caches, and TLBs. Therefore, the overhead of the file system will impact the performance of the whole system. Considering these factors, SCMFS utilizes the memory management unit (MMU) to map the file system address to the physical address on SCM. Meanwhile, SCMFS stores each file in continuous space to simplify the read/write processes.

In contrast to SCMFS, PMFS [Dulloor et al. 2014a] manages SCM completely independent of OS VMM. To be more specific, PMFS is a lightweight POSIX file system that utilizes SCM's byte addressability to optimize consistency, using a combination of atomic in-place updates, logging at cache-line granularity, and copy-on-write (COW). Further, SCMFS uses only clflush/mfence for ordering, and consistency validation is unclear, while PMFS provides strong consistency guarantees with a simple hardware primitive that provides software-enforceable guarantees of durability and ordering of stores to SCM. In addition, PMFS leverages the processor's write-protect control to protect the system from stray writes. In the end, memory-mapped I/O provides a memory-like access to storage, and PMFS implements an optimized memory-mapped I/O by mapping persistent memory pages directly into an application's address space.

5.2. Database Systems

Because PCM is nonvolatile, low latency and bitwise addressable, some works tried to use PCM to optimize the performance of relational databases. In general, the access speed of PCM is several orders of magnitude faster than that of hard disks. Through the use of PCM, the transaction log overhead in disk-based relational databases can be reduced. Gao et al. [2011] designed a logging method used in disk-based databases called PCMLogging, in which buffered updates and transaction logs are completely integrated. In this way, the dirty pages cached by PCM can also be used as log. There is no need to log dedicatedly. The experimental results show that, compared with a basic buffering and logging scheme, PCMLogging improves disk I/O performance by up to 40%.

In addition to adding PCM in databases to improve performance and reduce energy consumption, designing a friendly database algorithm is another study topic, which requires minimizing the number of expensive writes. Chen et al. [2011b] rethought database algorithms for PCM. They studied B+Tree and Hash join algorithms, which are applied to PCM. In this work, they found that it is more appropriate for PCM when improving B+Tree by no longer maintaining the order of leaf nodes and increasing the number of reads to get a reduction of the number of writes. Hash joins use the virtually partitioning algorithm instead of the original cache partitioning algorithm. This work mainly addressed NVM's write latency, endurance, and write-energy concerns, generally aiming to reduce the number of modified NVM bits.

The former work [Chen et al. 2011b] did not consider durable consistency for transaction processing. With emerging NVM, Pelley et al. [2013] reconsidered online transaction processing (OLTP) durability management to optimize recovery performance and forward-process throughput. They utilized cache to effectively reduce NVM read stalls. Treating NVM as a drop-in replacement for a disk allows near-instantaneous recovery,

but retains centralized logging overheads. In-place updates offer a simple design and can reduce these overheads, but introduce synchronization overhead to guarantee the correct write order to NVM. Therefore, they proposed a new recovery mechanism, NVM Group Commit, to minimize these stalls while still removing centralized logging and providing high throughput.

5.3. Persistent Data Structures

Compared with object serialization and accessing data structures by reading and writing files, persistent data structures can be mapped to user address space directly and are able to be used instantly. Persistent data structures are an attractive feature that can help ease the dependence on traditional file I/O. Disk-based systems or databases use persistent objects, such as Java Persistent API [Biswas and Ort 2006], or persistent data structures that are supported by special object-oriented databases. However, in contrast to a hard disk, PCM is bitwise addressable and can be accessed quickly. Therefore, designing a persistent data structure system is not only complex in terms of the architecture, but also costly in terms of software implementation. With PCM, therefore, we have to redesign a higher-performance method to access persistent data structures.

Mnemosyne [Volos et al. 2011] provides two persistent access ways: primitive and transactional. It uses persistent regions to achieve the user-mode accesses and supports different consistent-access models (single variable, append, shadow, in-place). NV-Heaps [Coburn et al. 2011] implements a persistent heap in user space and establishes robust protection mechanisms to reduce errors of pointers to volatile regions. Experiments show that, when used to implement the same data structure, NV-Heaps get a 32× higher performance in efficiency than Berkeley DB [Oracle 2010], and 244× that of Stasis [Sears and Brewer 2006]. In order to reduce software overhead, it can only use PCM as storage. Narayanan and Hodson [2012] utilized the residual power to write the contents of the registers and the cache to PCM only when the system (electricity) fails. They pointed out that the flush can be completed safely by consuming 35% of the residual energy at most, and the runtime performance is 1.6 to 13 times better than a persistent heap.

Journaling is a common technique in file-system and database-system designs, which logs the update to nonvolatile storage for high reliability and fast recovery. However, journaling will result in a bulk of storage writes, which deteriorate the system performance benefit of buffer caching. Previous works on this problem added nonvolatile memory as a separate journal area or as a write buffer of log files to improve performance. E. Lee et al. [2013] first proposed a scheme that intelligently incorporates the journaling functionality into the buffer cache architecture, hence minimizing additional memory copy and space overhead.

Persistent memory designs exploit logging or copy-on-write mechanisms to update persistent data, which reduces system performance to roughly half that of a native system without persistent support. Based on that fact, Zhao et al. [2013] designed Kiln, a persistent memory design that utilizes nonvolatile last-level cache and a nonvolatile memory to construct a persistent memory hierarchy. To ensure high efficiency and data persistence, Kiln allows direct updates to the real in-memory data structures. In addition, they developed a set of lightweight software and hardware extensions to facilitate atomicity and consistency support. However, the file cache of Kiln is simply used as the buffers of the journal, which is still the server for logging or COW, rather than as a place that enables an in-place update.

Kiln changes the microarchitecture to reduce the cost of logging. Kannan et al. [2014] focused on the software optimizations for existing hardware (volatile cache).

They analyzed the non-volatile memory (NVM) writes, and found that effective dual-use (capacity and persistent) NVM requires new methods that address cache sharing between persistent (consistency and durability) and nonpersistent applications in end-user devices. They noted that persistent applications will incur high cache miss due to consistency, durability, and failure recovery. Therefore, to overcome the problem of high cache misses, they proposed a page-contiguity algorithm that reduces interference-related cache misses. Moreover, they implemented a cache-efficient NVM write-aware memory allocator that greatly reduces cache-line flushes of allocator state. Last, they employed a hybrid logging that substantially reduces durability overheads.

5.4. High-Performance Computing

Applying PCM to high-performance computing can reduce both the energy consumption of systems and the overhead of operations such as backup, restore, and so on.

Li et al. [2012] considered first the impact of using PCM main memory directly on the existing scientific computing applications. They carefully studied four applications: Nek5000, CAM, GTC, and S3D, which can be applied to fluid simulation, weather forecasting, turbulence simulation, and turbulent combustion, respectively. Moreover, based on the research of the proportion of read/write and the rate of memory access, they pointed out that two of the applications, in which 31% and 27% of the data have a higher read/write ratio, are suitable for PCM. They further pointed out that replacing DRAM with PCM can save at least 31% of the energy consumption.

Dong et al. [2009b, 2011a] proposed a massively parallel processing (MPP)–oriented and PCM-based hybrid global/local checkpoint mechanism. It is inefficient to store a global checkpoint with a hard disk, for example, the checkpoint overhead is up to 236.7% for a system scale up to 10 petaFLOPs. In order to reduce the unacceptable overhead, they used PCM to store global and local checkpoints. In addition, to improve the speed of recording the checkpoint, it is necessary to increase the PCM bandwidth. The ideal situation is that the PCM bandwidth is close to the DRAM bandwidth, which means that PCM will not become the bottleneck. To achieve this goal, they used 72 parallel PCM chips. PCM-DIMM checkpointing enables MPP systems to scale up to 500 petaFlops while keeping the overhead within 10%. By exploiting 3D-stacked technology to integrate PCM with DRAM, backup can be completed at very high bandwidth while less than 5% overhead is needed, even at the level of 1 exaFLOPS.

It is worthwhile to note the use of PCM to design a new architecture for the computing nodes. Ranganathan [2011] proposed another data-center system component—Nanostore. Each Nanostore consists of one CPU core and multiple 3D-stacked PCMs; numerous Nanostores are used to build the entire system. The design flattens the hierarchy of main memory and storage into only one layer, which simplifies the whole system architecture. This design achieves smaller latency and lager bandwidth, which enable its higher performance. Moreover, the simplified storage architecture and the use of PCM make it more energy efficient. A single node that groups 9 Nanostores with 8+1 redundancy can provide 0.5TB PCM memory with Tera-ops of local computation collocated with about 256GB/s of data bandwidth [Ranganathan 2011].

5.5. Summary

Section 5 overviews the works that apply PCM to storage from the perspective of file system, database, persistent data structures, and high-performance computing. Some file-system designs mainly exploit the byte addressability of PCM, which can be used to improve the performance of updating metadata. On the other hand, SCMFS attaches SCM to the memory bus directly for better performance. But it is constrained by some critical system resources (e.g., memory bus, TLB). When using PCM in relational database designs to improve the performance, we can use a B+Tree algorithm to reduce

Table VIII. Typical Characteristics of PCM and Their Effect

Characteristics	Expected Impact
Density (vs. DRAM)	Larger memory
Performance (vs. Disk)	Faster storage, hybrid storage (positioned between memory and storage)
Standby power	Energy efficiency
Byte accessible (vs. Disk)	New storage paradigm (lightweight, low overhead)
Nonvolatile (vs. DRAM)	Enable memory-like devices' usage in storage

Table IX. The PCM Characteristics Used in Different Solutions and Corresponding Impact

Work	Field	Utilized features	Impact
PFFS, use PCM to store metadata in hybrid disk	File system	Byte-accessible, nonvolatile	Performance
BPFS, file system for PCM	File system	Byte-accessible, nonvolatile	Performance, robustness
New versioning file system	File system	Byte-accessible, nonvolatile	Support large write, snapshot
SCMF, file system for SCM on memory bus	File system	Byte-accessible, nonvolatile	Performance
PMFS, file system for PCM on memory bus	File system	Byte-accessible, nonvolatile	Performance, robustness
Utilize PCM to cache dirty pages without specialized logging	Database	Performance, Byte-accessible, nonvolatile	Performance
Optimize B+Tree, Hash join to reduce write	Database	Write latency (negative)	Performance
OLTP fast recovery with PCM	Database	Nonvolatile	Performance, robustness
Mnemosyne, NV-Heap	Persistent structure	Byte-accessible, nonvolatile	Performance, robustness
Replace read-dominant memory area with PCM	HPC	Low standby power	Energy consumption
PCM for checkpoint	HPC	Bandwidth, non-volatile	Performance, scalability
Computation node with pure PCM storage	HPC	Performance, Byte-accessible, non-volatile	Performance, energy

the write operations to PCM. Using PCM as a drop-in replacement for a disk is another method that can improve the performance by providing fast recovery. The employment of an in-place update scheme can mitigate logging overhead. The technologies for data persistence optimize the performance from two aspects—the first is the designs of persistent data structures, and the second is the schemes that support the data persistence efficiently (e.g., buffering, in-place updates). The schemes, applying PCM to the field of high-performance computing, either use PCM in existing architectures or design a new architecture to optimize the performance. Since it remains undecided whether current PCM technology will be a commercially and technically viable alternative to entrenched technologies such as Flash-based SSDs, we can get the best performance within a cost constraint by tiring storage systems with PCM, SSDs, and HDDs [Kim et al. 2014].

Table VIII summarizes the PCM attributes considered when designing system software. Table IX summarizes different solutions based on PCM features as well as the corresponding impacts. Due to PCM's low latency and byte-addressable characteristics compared to traditional storage devices, some traditional OS software architecture techniques (e.g., page cache) should be revised or even redesigned. When using an HDD + PCM hybrid storage architecture, the system stores the frequently accessed data in PCM, while storing the infrequently accessed data in the HDD. For example,

the file system stores the metadata in the PCM rather than the HDD. Database software stores the log data in the PCM instead of the HDD. Currently, a hybrid secondary storage architecture may be more promising compared to a pure one, since we can get the best of them to gain overall performance enhancement. Large-capacity and low-idle power consumption are two key characteristics of PCM, but the studies employing both are still relatively rare. Many efforts have taken advantage of PCM's attributes of byte-addressability and nonvolatility. With the emergence of storage devices with the characteristics of main memory, cross-border devices with the features of higher performance, lower-energy consumption, stronger robustness, less weight, and higher scalability are available.

6. USING PCM AS BOTH MAIN MEMORY AND STORAGE

Recently, researchers have started to explore schemes that utilize the nonvolatile memory as both main memory and secondary storage simultaneously. An NVM device is introduced into a computer system, and divided into two portions logically. One portion of the NVM device is used as main memory, while the other portion is used as storage. Furthermore, an integrated management of main memory and storage is explored to enhance the performance of the entire system.

6.1. System Architectures

Oikawa [2013] first designed and implemented the integrated management of main memory and storage. In this system, he constructed a file system that is used to manage the blocks created on NVM. Specifically, the block size of the file system is the same as the page size of the virtual memory, so that a file system block can be used as a physical memory page and mapped into a virtual memory address space. This system can achieve 23% to 26% of the performance gain, which is comparable to the DRAM case. However, it lacks disposals on NVM's write-endurance problem.

Baek et al. [2013] mainly focused on energy efficiency and performance of the integrated system. They evaluated energy efficiency by conducting experiments on RAM-Flash, RAM-SCM (storage-class memory), SCM-Flash, and SCM-only organization. The results show that applying SCM into memory hierarchy significantly reduces energy consumption, and the SCM-only organization performs the best. Furthermore, we can gain more energy efficiency by using the instant on/off scheme that turns systems off when they are idle. However, the introduction of SCM will degrade system performance, especially for memory-intensive applications. To relieve this problem, they proposed a metadata in-place scheme that manipulates metadata on SCM directly, enhancing system performance and reliability. To prevent partial writes of metadata from causing metadata inconsistency, however, extra operations such as atomic mechanism or consistency checking need to be performed. The idea of a metadata in-place update can also be applied to file data. Further, with SCM, file and memory objects can be managed as a single object in a unified way, which can make the operating system simple and efficient.

Jung and Cho [2013] proposed Memorage, a system that manages the PRAM main memory resource and the PRAM secondary storage resource in an integrated way to achieve the best trade-off between performance and cost. The motivation of Memorage is based on the fact that the traditional dichotomy of memory and storage will likely keep fast PRAM resources underutilized. In addition, according to some previous studies, a secondary storage device in a system is likely to have substantially unused space during its lifetime. To effectively address this wastefulness, Memorage's main memory borrows directly accessible memory resources from the PRAM storage device, which saves the overheads that swap operations of conventional OS virtual memory manager will cause. In addition, it can redistribute write traffic to the main memory and storage

Table X. Various Schemes when Using PCM as Both Main Memory and Storage

Work	Key idea	Implement
Oikawa [2013]	Integrating the management of memory and storage via a file system	OS
Baek et al. [2013]	Introducing SCM can save energy of a system; in order to improve performance of an in-place update scheme, a single object management scheme, and an instant on/off scheme was exploited.	OS
Jung and Cho [2013]	Improving performance by granting more direct access to physical PRAM capacity, and achieving longer lifetime by spreading writes to all PRAM capacity without being limited by the main memory-storage wall.	OS
Liu et al. [2014]	Providing consistency and durability for persistent store, while relaxing these restraints for working memory.	OS

by trading resources between the main memory and storage, by which it can gain a reasonable lifetime and reliability.

Liu et al. [2014] found that a unified architecture oblivious to consistency and durability would lead to suboptimal design. Hence, they proposed NVM Duet, a novel unified memory and storage architecture, which provides the required consistency and durability guarantees for storage while relaxing these two constraints if accesses to PCM are for memory. To achieve this goal, they implemented a new hardware/software interface that enables the memory controller to differentiate between the two types of PCM usage, and proposed a new memory scheduler to fully exploit the bank-level parallelism present in the address stream. Moreover, they put forward a smart refreshing scheme that eliminates unnecessary refresh operations.

6.2. Summary

Table X summarizes different schemes when implementing the integrated management of main memory and storage. Using PCM as both main memory and storage, and managing them in an integrated way, has shown the potential of improving system performance. This kind of architecture has a large amount of memory; thus, the page faults can be reduced and the wear-out problem can be eliminated. I/O overheads can be reduced and both the initialization and termination of program executions can be accelerated.

The main shortcoming of this integrated management is that critical system resources are limited, such as the memory bus and TLB. Hence, we can explore methods from both hardware and software to achieve the best trade-off between performance and cost. This integration of main memory and storage opens up a new research field of using PCM. Current works in this field are just a starting point; we are looking forward to seeing more works on this issue.

7. CONCLUSIONS AND FUTURE DIRECTIONS

This article has provided a detailed survey and review of the areas of computer architectures and software systems that are oriented to the newly emerging PCM devices. Existing processor cache, main memory systems, and storage software systems do not adapt to attributes of PCM devices well, because they are designed for DRAM and HDD. Compared with them, PCM devices have the advantages of byte accessibility; high-density, low-idle power consumption; and nonvolatility. The disadvantages are high write latency, limited write lifetime, and high-access power consumption. When PCM devices are used in computer systems, it is desirable to revise or even redesign computer architectures and software systems for attributes of

PCM devices. As the survey shows, the past several years have witnessed a dramatic increase in PCM-related activity, with many new applications being addressed.

Due to the longer access latency and limited write endurance compared to SRAM, employing PCM to processor cache is comparatively hard. However, existing works in this direction are inspiring. As described in Section 3, these works provide us with two available processor-cache hierarchies (hybrid and pure PCM). To make the systems practical, researchers designed a set of techniques to overcome the two drawbacks of PCM, such as reducing redundant writes, wear leveling, and so on.

This survey provides a new framework for analyzing the suitability of main memory management mechanisms to emerging PCM devices. We identified performance optimization, lifetime prolongation, and energy saving as necessary features of a main memory system aiming to address this problem. These features form the axes of the comparison framework that we used to survey selected systems in Section 4.

When designing software systems for a computer system with PCM devices, one takes account of the PCM attributes such as byte accessibility, nonvolatility, asymmetry between read and write performances, and low-idle power consumption. The design objectives include optimizing I/O performance, prolonging write lifetime, saving energy, and so on. This survey also provides a detailed categorization of file systems, database systems, and persistent data structures presented in the literature to date. Currently, one big hurdle for PCM research is the lack of real hardware for performance evaluation. However, there are some good simulation tools available for researchers, such as Intel's Persistent Memory Emulation Platform [Dulloor et al. 2014b] and NVMpro [Sengupta et al. 2015].

The future of PCM is bright. There are many areas to which the techniques associated with PCM can apply. However, to ensure the wide use of PCM, there are still many problems to be addressed, such as long write latency, limited write endurance, and high active energy. Furthermore, since today's software architectures are based on the fact that the main memory is volatile, the application of PCM may also affect the existing software architectures. It is necessary to rethink computer architectures and software systems when we do the research based on PCM. This detailed analysis of PCM enables us to identify some missing areas of PCM, some potential techniques that have yet to be applied, and emerging areas that need to be studied in greater detail.

A lot of studies have been conducted on the emerging new memory technologies, and the results are promising. Still, there are some interesting and open issues waiting to be solved. Here, we list some future directions of PCM from the aspects of using PCM as processor cache, as main memory, as storage, and as both memory and storage. First, using PCM as processor cache is challenging; we can try to exploit hybrid cache architecture by using other new memory technologies (e.g., eDRAM, STT-RAM, RRAM) to conduct our studies. Second, when using PCM as main memory, an optimizing buffer scheme is a good research direction, since it can help us to hide the long write latency of PCM. The granularity of the wear-leveling scheme may be another research direction. We can try to implement different granularities on different system levels for better performance. Fault tolerance is also an interesting issue. We can conduct our studies on high memory utilization and low extra overhead based on existing works. Reset and set operations consume different amounts of energy; we can exploit this discrepancy to save system energy. Third, according to different application features, we can design a proper logging mechanism based on the byte addressability of PCM. Another direction deserves further study, that is, how to relieve the pressure of some critical resources to obtain good performance, when attaching PCM storage directly to the memory bus. Last, using PCM as both memory and storage has good prospects to improve system performance. Current studies in this field are just a starting point. The integrated

management of memory and storage, as well as nonuniform NVM, deserves further study.

REFERENCES

W. Arden. 2009. Semiconductor Industries Association - International Technology Roadmap for Semiconductors. http://www.itrs2.net/itrs-reports.html.

R. Azevedoy, J. D. Davisz, and K. Straussz. 2013. Zombie memory: Extending memory lifetime by reviving dead blocks. In *The 40th Annual International Symposium on Computer Architecture*.

S. Baek, J. Choi, D. Lee, and S. H. Noh. 2013. Energy-efficient and high-performance software architecture for storage class memory. *ACM Transactions on Embedding Computing Systems* 12, 3, Article 81, 22 pages. DOI:http://dx.doi.org/10.1145/2442116.2442131

S. Baek, H. G. Lee, C. Nicopoulos, and J. Kim. 2012. A dual-phase compression mechanism for hybrid DRAM/PCM main memory architectures. In *Proceedings of the Great Lakes Symposium on VLSI (GLSVLSI'12)*. ACM, New York, NY, 345–350. DOI:http://dx.doi.org/10.1145/2206781.2206865

R. Biswas and E. Ort. 2006. The java persistence API - a simpler programming model for entity persistence.Retrieved March 30, 2016 from http://java.sun.com/developer/technicalArticles/J2EE/jpa/.

S. Bock, B. Childers, R. Melhem, D. Mossé, and Y. Zhang. 2011. Analyzing the impact of useless write-backs on the endurance and energy consumption of PCM main memory. In *IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS'11)*. IEEE, 56–65.

G. W. Burr, M. J. Breitwisch, M. Franceschini, D. Garetto, K. Gopalakrishnan, B. Jackson B. et al. 2010. Phase change memory technology. *Journal of Vacuum Science and Technology B* 28, 223–262.

G. W. Burr, B. N. Kurdi, J. C. Scott, C. H. Lam, K. Gopalakrishnan, and R. S. Shenoy. 2008. Overview of candidate device technologies for storage-class memory. *IBM Journal of Research and Development* 52, 4.5, 449–464.

J. Chen, R. C. Chiang, H. H. Huang, and G. Venkataramani. 2011a. Energy-aware writes to non-volatile main memory. *ACM SIGOPS Operating Systems Review* 45, 3, 48–52. DOI:http://dx.doi.org/10.1145/2094091.2094104 ACM New York, NY, USA.

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

S. Cho and H. Lee. 2009. Flip-n-write: A simple deterministic technique to improve PRAM write performance, energy and endurance. In *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'09)*. ACM, New York, NY, 347–357. DOI:http://dx.doi.org/10.1145/1669112.1669157

J. Coburn, A. M. Caulfield, A. Akel, L. M. Grupp, R. K. Gupta, R. Jhala, and S. Swanson. 2011. NV-heaps: Making persistent objects fast and safe with next-generation, non-volatile memories. In *Proceedings of the 16th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'11)*.

J. Condit, E. B. Nightingale, C. Frost, E. Ipek, B. Lee, D. Burger, and D. Coetzee. 2009. Better I/O through byte-addressable, persistent memory. In *Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles (SOSP'09)*. New York, NY, 133–146. DOI:http://dx.doi.org/10.1145/1629575.1629589

G. Dhiman, R. Ayoub, and T. Rosing. 2009. PDRAM: A hybrid PRAM and DRAM main memory system. In *Proceedings of the 46th ACM/IEEE Design Automation Conference (DAC'09)*. 664–669.

J. Dong, L. Zhang, Y. Han, and Y. Wang. 2011b. Wear rate leveling: Lifetime enhancement of PRAM with endurance variation. In *Proceedings of the 51st Annual Design Automation Conference (DAC'11)*. ACM, New York, NY, Article 36, 6 pages.

X. Dong, N. P. Jouppi, and Y. Xie. 2009a. PCRAMsim: System-level performance energy and area modeling for phase-change RAM. In *IEEE/ACM International Conference on Computer-Aided Design*.

X. Dong, N. Muralimanohar, N. Jouppi, R. Kaufmann, and Y. Xie. 2009b. Leveraging 3d PCRAM technologies to reduce checkpoint overhead for future exascale systems. In *Super Computing Conference (SC'09)*. Portland, OR.

X. Dong, X. Wu, G. Sun, Y. Xie, H. Li, and Y. Chen. 2008. Circuit and microarchitecture evaluation of 3D stacking magnetic RAM (MRAM) as a universal memory replacement. In *Proceedings of the 45th Annual Design Automation Conference*. Anaheim, CA.

X. Dong, Y. Xie, N. Muralimanohar, and N. P. Jouppi. 2011a. Hybrid checkpointing using emerging nonvolatile memories for future exascale systems. *ACM Transactions on Architecture and Code Optimization* 8, 6, 1–29.

K. Doshi and P. Varman. 2012. WrAP: Managing byte-addressable persistent memory. In *Memory Architecture and Organization Workshop (MEAOW'12)*.

U. Drepper. 2007. What Every Programmer Should Know About Memory. Retrieved March 30, 2016 from http://people.redhat.com/drepper/cpumemory.pdf.

Y. Du, M. Zhou, B. R. Childers, R. Melhem, and D. Mosse. 2013. Delta-compressed caching for overcoming the write bandwidth limitation of hybrid main memory. *ACM Transactions on Architecture and Code Optimization* 9, 4.

S. R. Dulloor, S. Kumar, A. Keshavamurthy, P. Lantz, D. Reddy, R. Sankaran, and J. Jackson. 2014a. System software for persistent memory. In *Proceedings of the 9th European Conference on Computer Systems (EuroSys'14)*. ACM, New York, NY, Article 15, 15 pages. DOI:http://dx.doi.org/10.1145/2592798.2592814

S. R. Dulloor, S. Kumar, A. Keshavamurthy, P. Lantz, D. Reddy, R. Sankaran, and J. Jackson. 2014b. System software for persistent memory. In *Proceedings of the 9th European Conference on Computer Systems (EuroSys'14)*. ACM, New York, NY, Article 15, 15 pages. DOI:http://dx.doi.org/10.1145/2592798.2592814

J. Fan, S. Jiang, J. Shu, L. Sun, and Q. Hu. 2014. WL-reviver: A framework for reviving any wear-leveling techniques in the face of failures on phase change memory. In *Proceedings of the 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*.

J. Fan, S. Jiang, J. Shu, H. Zhang, and W. Zhen. 2013. Aegis: Partitioning data block for efficient recovery of stuck-at-faults in phase change memory. In *46th Annual IEEE/ACM International Symposium on Microarchitecture*.

Y. Fang, H. Li, and X. Li. 2012. SoftPCM: Enhancing energy efficiency and lifetime of phase change memory in video applications via approximate write. In *2012 IEEE 21st Asian Test Symposium (ATS)*. IEEE, 131–136.

S. Gao, J. Xu, B. He, B. Choi, and H. Hu. 2011. PCMLogging: Reducing transaction logging overhead with PCM. In *Proceedings of the 20th ACM International Conference on Information and Knowledge Management (CIKM'11)*.

S. Guo, Z. Liu, D. Wang, H. Wang, and G. Li. 2012. Wear-resistant hybrid cache architecture with phase change memory. In *Proceedings of the 2012 IEEE 7th International Conference on Networking, Architecture, and Storage (NAS'12)*. IEEE Computer Society, Washington, DC, 268–272. DOI:http://dx.doi.org/10.1109/NAS.2012.37

J. Hu, Q. Zhuge, C. J. Xue, W. Tseng, and E. H.-M. Sha. 2013. Software enabled wear-leveling for hybrid PCM main memory on embedded systems. In *13th ACM/IEEE Design, Automation and Test in Europe*.

A. N. Jacobvitz, R. Calderbank, and D. J. Sorin. 2013. Coset coding to extend the lifetime of memory. In *Proceedings of the 2013 IEEE 19th International Symposium on High Performance Computer Architecture (HPCA'13)*. IEEE Computer Society, Washington, DC, 222–233. DOI:http://dx.doi.org/10.1109/HPCA.2013.6522321

ITRS. 2007. Process Integration, Devices and Structures, International Technology Roadmap for Semiconductors. http://www.itrs2.net/itrs-reports.html.

L. Jiang, Y. Du, B. Zhao, Y. Zhang, B. R. Childers, and J. Yang. 2013. Hardware-assisted cooperative integration of wear-leveling and salvaging for phase change memory. *ACM Transactions on Architecture and Code Optimization* 10, 2.

L. Jiang, Y. Zhang, and J. Yang. 2012. ER: Elastic reset for low power and long endurance MLC based phase change memory. In *Proceedings of the 2012 ACM/IEEE International Symposium on Low Power Electronics and Design (ISLPED'12)*. ACM, New York, NY, 39–44. DOI:http://dx.doi.org/10.1145/2333660.2333672

Y. Joo, D. Niu, X. Dong, G. Sun, N. Chang, and Y. Xie. 2010. Energy- and endurance-aware design of phase change memory caches. In *Proceedings of the Conference on Design, Automation and Test in Europe (DATE'10)*. European Design and Automation Association, Leuven, Belgium, 136–141. http://dl.acm.org/citation.cfm?id=1870926.1870961.

J.-Y. Jung and S. Cho. 2013. Memorage: Emerging persistent RAM based malleable main memory and storage architecture. In *Proceedings of the 27th International ACM Conference on International Conference on Supercomputing (ICS'13)*. ACM, New York, NY, 115–126. DOI:http://dx.doi.org/10.1145/2464996.2465005

S. Kannan, A. Gavrilovska, and K. Schwan. 2014. Reducing the cost of persistence for nonvolatile heaps in end user devices. In *IEEE 20th International Symposium on High Performance Computer Architecture (HPCA'14)*. IEEE, 512–523.

D. Kau, S. Tang, I. V. Karpov, and R. Dodge. 2009. A stackable cross point phase change memory. In *Electron Devices Meeting (IEDM'09)*. 1–4. DOI:http://dx.doi.org/10.1109/IEDM.2009.5424263

H. Kim, S. Seshadri, C. L. Dickey, and L. Chiu. 2014. Evaluating phase change memory for enterprise storage systems: A study of caching and tiering approaches. In *Proceedings of the 12th USENIX*

Conference on File and Storage Technologies (FAST'14). USENIX Association, Berkeley, CA, 33–45. http://dl.acm.org/citation.cfm?id=2591305.2591309.

M. H. Kryder and C. S. Kim. 2009. After hard drives - what comes next? *IEEE Transactions on Magnetics* 45, 3406–3413.

B. C. Lee, E. Ipek, O. Mutlu, and D. Burger. 2009. Architecting phase change memory as a scalable DRAM alternative. In *Proceedings of the 36th International Symposium on Computer Architecture (ISCA'09)*. Austin, TX, 2–13.

B. C. Lee, P. Zhou, J. Yang, Y. Zhang, B. Zhao, E. Ipek, O. Mutlu, and D. Burger. 2010. Phase-change technology and the future of main memory. *IEEE Micro* 30, 1, 131–141.

B. C. Lee, P. Zhou, J. Yang, Y. Zhang, B. Zhao, E. Ipek, O. Mutlu, and D. Burger. 2013. On-demand snapshot: An efficient versioning file system for phase-change memory. *IEEE Transactions on Knowledge and Data Engineering* 25, 12, 2841–2853.

E. Lee, H. Bahn, and S. H. Noh. 2013. Unioning of the buffer cache and journaling layers with non-volatile memory. In *Proceedings of the 11th USENIX Conference on File and Storage Technologies (FAST'13)*. USENIX Association, Berkeley, CA, 73–80. http://dl.acm.org/citation.cfm?id=2591272.2591280.

C. Lefurgy, K. Rajamani, F. Rawson, W. Felter, M. Kistler, and T. W. Keller. 2003. Energy management for commercial servers. *Computer* 36, 12, 39–48.

D. L. Lewis and H.-H. S. Lee. 2009. Architectural evaluation of 3D stacked RRAM caches. In *IEEE International 3D System Integration Conference*.

D. Li, J. Vetter, G. Marin, C. McCurdy, C. Cira, Z. Liu, and W. Yu. 2012. Identifying opportunities for byte-addressable non-volatile memory in extreme-scale scientific applications. In *Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS'12)*.

J.-T. Lin, Y.-B. Liao, M.-H. Chiang, I.-H. Chiu, C.-L. Lin, W.-C. Hsu, P.-C. Chiang, S.-S. Sheu, W.-H. Hsu, Y.-Y. Liu, K.-L. Su, M.-J. Kao, and M.-J. Tsai. 2009. Design optimization in write speed of multi-level cell application for phase change memory. In *Proceedings of the IEEE International Conference of Electron Devices and Solid-State Circuits (EDSSC'09)*. IEEE, Article 5394196, 4 pages. http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=5394196.

D. Liu, T. Wang, Y. Wang, Z. Shao, Q. Zhuge, and E. Sha. 2013. Curling-PCM: Application-specific wear leveling for phase change memory based embedded systems. In *Proceedings of the 18th Asia and South Pacific Design Automation Conference (ASP-DAC'13)*.

R.-S. Liu, D.-Y. Shen, C.-L. Yang, S.-C. Yu, and C.-Y. M. Wang. 2014. NVM duet: Unified working memory and persistent store architecture. In *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'14)*. ACM, New York, NY, 455–470. DOI:http://dx.doi.org/10.1145/2541940.2541957

L. Long, D. Liu, J. Hu, S. Gu, Q. G. Zhuge, and E. H.-M. Sha. 2014. A space allocation and reuse strategy for PCM-based embedded systems. *Journal of Systems Architecture* 60, 8, 655–667. DOI:http://dx.doi.org/10.1016/j.sysarc.2014.07.002

N. Lu, I.-S. Choi, S.-H. Ko, and S.-D. Kim. 2012. An effective hierarchical PRAM-SLC-MLC hybrid solid state disk. In *IEEE/ACIS 11th International Conference on Computer and Information Science*. 113–118.

S. Mittal, J. S. Vetter, and Dong Li. 2015. A survey of architectural approaches for managing embedded DRAM and non-volatile on-chip caches. *IEEE Transactions on Parallel and Distributed Systems* 26, 6, 1524–1537. DOI:http://dx.doi.org/10.1109/TPDS.2014.2324563

J. C. Mogul, E. Argollo, M. Shah, and P. Faraboschi. 2009. Operating system support for NVM+DRAM hybrid main memory. In *Proceedings of the 12th Workshop on Hot Topics in Operating Systems (HatOS'09)*. 18–20.

I. Moraru, D. G. Andersen, M. Kaminsky, N. Tolia, P. Ranganathan, and N. Binkert. 2013. Consistent, durable, and safe memory management for byte-addressable non volatile main memory. In *Proceedings of the 1st ACM SIGOPS Conference on Timely Results in Operating Systems (TRIOS'13)*. ACM, New York, NY, Article 1, 17 pages. DOI:http://dx.doi.org/10.1145/2524211.2524216

D. Narayanan and O. Hodson. 2012. Whole-system persistence. In *Proceedings of the 17th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'12)*.

S. Oikawa. 2013. Integrating memory management with a file system on a non-volatile main memory system. In *Proceedings of the 28th Annual ACM Symposium on Applied Computing (SAC'13)*. ACM, New York, NY, 1589–1594. DOI:http://dx.doi.org/10.1145/2480362.2480660

Oracle. 2010. Oracle Berkeley DB downloads. Retrieved March 30, 2016 from http://www.oracle.com/technology/products/berkeley-db/index.html.

Y. Park, S.-H. Lim, C. Lee, and K. H. Park. 2008. 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 (SAC'08)*.

Y. Park, S. K. Park, and K. H. Park. 2010. Linux kernel support to exploit phase change memory. In *Linux Symposium*.

S. Pelley, T. F. Wenisch, B. T. Gold, and B. Bridge. 2013. Storage management in the NVRAM era. *Proceedings of the VLDB Endowment* 7, 2.

M. K. Qureshi. 2011. Pay-as-you-go: Low-overhead hard-error correction for phase change memories. In *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'11)*.

M. K. Qureshi, J. Karidis, M. Fraceschini, V. Srinivasan, L. Lastras, and B. Abali. 2009. Enhancing lifetime and security of phase change memories via start-gap wear leveling. In *Proceedings of the International Symposium on Microarchitecture*.

M. K. Qureshi, M. F. Michele, and A. L.-M. Luis. 2010. Improving read performance of phase change memories via write cancellation and write pausing. In *HPCA'10*.

M. K. Qureshi, A. Seznec, L. A. Lastras, and M. M. Franceschini. 2011. Practical and secure PCM systems by online detection of malicious write streams. In *IEEE 17th International Symposium on High Performance Computer Architecture (HPCA'11)*, Vol. 37. 478–489.

M. K. Qureshi, V. Srinivasan, and J. A. Rivers. 2009. Scalable high performance main memory system using phase-change memory technology. In *Proceedings of the 36th Annual International Symposium on Computer Architecture (ISCA'09)*. New York, NY, 24–33.

L. Ramos, E. Gorbatov, and R. Bianchini. 2011. Page placement in hybrid memory systems. In *ICS'11*. 85–95.

P. Ranganathan. 2011. From microprocessors to nanostores: Re- thinking data-centric systems. *IEEE Computer* 44, 1, 39–48.

S. Raoux, G. W. Burr, M. J. Breitwisch, C. T. Rettner, Y.-C. Chen, R. M. Shelby, M. Salinga, D. Krebs, S.-H. Chen, H.-L. Lung, and C. H. Lam. 2008. Phase-change random access memory: A scalable technology. *IBM Journal of Research and Development* 52, 4.5, 465–479.

S. Raoux, F. Xiong, M. Wuttig, and E. Pop. 2014. Phase change materials and phase change memory. *MRS Bulletin* 39, 8, 703–710. DOI:http://dx.doi.org/10.1557/mrs.2014.139

D. A. Roberts. 2011. *Efficient Data Center Architectures Using Non-Volatile Memory and Reliability Techniques)*. Ph.D. Dissertation. The University of Michigan, Ann Arbor, MI.

S. Schechter, H. L. Gabriel, K. Strauss, and D. Burger. 2010. Use ECP, not ECC, for hard failures in resistive memories. In *ISCA*.

R. Sears and E. Brewer. 2006. Stasis: Flexible transactional storage. In *OSDI'06: Proceedings of the 7th Symposium on Operating Systems Design and Implementation*. USENIX Association, Berkeley, CA, 29C–44. DOI:http://dx.doi.org/10.1109/IEDM.2009.5424263

D. Sengupta, Q. Wang, H. Volos, L. Cherkasova, J. Li, G. Magalhaes, and K. Schwan. 2015. A framework for emulating non-volatile memory systems with different performance characteristics. In *Proceedings of the 6th ACM/SPEC International Conference on Performance Engineering (ICPE'15)*. ACM, New York, NY, 317–320. DOI:http://dx.doi.org/10.1145/2668930.2695529

N. H. Seong, D. H. Woo, and H.-H. S. Lee. 2010a. Security refresh: Prevent malicious wear-out and increase durability for phase-change memory with dynamically randomized address mapping. In *ISCA'10*, Vol. 37.

N. H. Seong, D. H. Woo, V. Srinivasan, J. A. Rivers, and H.-H. S. Lee. 2010b. SAFER: Stuck-at-fault error recovery for memories. In *Proceedings of the 43rd Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE Computer Society, 115–124.

N. H. Seong, S. Yeo, and H.-H. S. Lee. 2013. Tri-level-cell phase change memory: Toward an efficient and reliable memory system. In *Proceedings of the 40th Annual International Symposium on Computer Architecture (ISCA'13)*. ACM, New York, NY, 440–451. http://dl.acm.org/citation.cfm?id=2485960.

A. Seznec. 2010. A phase change memory as a secure main memory. *IEEE Computer Architecture Letters* 9, 1, 5–8.

G. Sun, D. Niu, J. Ouyang, and Y. Xie. 2011. A frequent-value based PRAM memory architecture. In *Proceedings of the 16th Asia and South Pacific Design Automation Conference (ASPDAC'11)*. IEEE Press, Piscataway, NJ, 211–216. http://dl.acm.org/citation.cfm?id=1950815.1950867.

S. Venkataraman, N. Tolia, P. Ranganathan, and R. H. Campbell. 2011. Consistent and durable data structures for non-volatile byte-addressable memory. In *Proceedings of the 9th USENIX Conference on File and Stroage Technologies (FAST'11)*. USENIX Association, Berkeley, CA, 5–5. http://dl.acm.org/citation.cfm?id=1960475.1960480.

H. Volos, A. J. Tack, and M. M. Swift. 2011. Mnemosyne: Lightweight persistent memory. In *SIGARCH Computer Architecture News*, Vol. 39.

J. Wang, X. Dong, G. Sun, D. Niu, and Y. Xue. 2011. Energy-efficient multi-level cell phase-change memory system with data encoding. In *IEEE 29th International Conference on Computer Design (ICCD'11)*. IEEE, 175–182.

T. Wang, D. Liu, Y. Wang, and Z. Shao. 2015. Towards write-activity-aware page table management for non-volatile main memories. *ACM Transactions on Embedded Computing Systems* 14, 2, Article 34, 23 pages. DOI:http://dx.doi.org/10.1145/2697394

X. Wu, J. Li, L. Zhang, E. Speight, and Y. Xie. 2009a. Hybrid cache architecture with disparate memory technologies. In *Proceedings of the 36th Annual International Symposium on Computer Architecture (ISCA'09)*. New York, NY, 34–45.

X. Wu, J. Li, L. Zhang, E. Speight, and Y. Xie. 2009b. Power and performance of read-write aware hybrid caches with non-volatile memories. In *Proceedings of Design Automation and Test in Europe (DATE'09)*. 737–742.

X. Wu and A. L. N. Reddy. 2011. SCMFS: A file system for storage class memory. In *Proceedings of International Conference for High Performance Computing, Networking, Storage and Analysis (SC'11)*. ACM, New York, NY, Article 39, 11 pages. DOI:http://dx.doi.org/10.1145/2063384.2063436

N. Yamada, E. Ohno, N. Akahira, K. Nishiuchi, K. Nagata, and M. Takao. 1987. High speed overwritable phase change optical disk material. *Japanese Journal of Applied Physics Supplement* 26, 8, 61C66.

B. Yang, J. Lee, J. Kim, J. Cho, S. Lee, and B. Yu. 2007. A low power phase change random access memory using a data comparison write scheme. In *ISCAS'07*.

J. Yang, Q. Wei, C. Chen, C. Wang, K. L. Yong, and B. He. 2015. NV-tree: Reducing consistency cost for NVM-based single level systems. In *13th USENIX Conference on File and Storage Technologies (FAST'15)*. USENIX Association, Santa Clara, CA, 167–181. https://www.usenix.org/conference/fast15/technical-sessions/presentation/yang.

D. H. Yoon, N. Muralimanohar, J. Chang, P. Ranganathan, N. P. Jouppi, and M. Erez. 2011. FREE-p: Protecting non-volatile memory against both hard and soft errors. In *HPCA'11*.

H. Yoon, J. Meza, R. Ausavarungnirun, R. A. Harding, and O. Mutlu. 2012. Row buffer locality aware caching policies for hybrid memories. In *Proceedings of the 30th IEEE International Conference on Computer Design (ICCD'12)*. Montreal, Quebec, Canada, 1–8.

H. Yoon, N. Muralimanohar, J. Meza, O. Mutlu, and N. P. Jouppi. 2013. *Techniques for Data Mapping and Buffering to Exploit Asymmetry in Multi-Level Cell (Phase Change) Memory*. SAFARI Technical Report 2013-002. Computer Architecture Lab (CALCM) at Carnegie Mellon University.

J. Zhao, S. Li, D. H. Yoon, Y. Xie, and N. P. Jouppi. 2013. Kiln: Closing the performance gap between systems with and without persistence support. In *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'13)*. ACM, New York, NY, 421–432. DOI:http://dx.doi.org/10.1145/2540708.2540744

M. Zhao, L. Jiang, Y. Zhang, and C. J. Xue. 2014a. SLC-enabled wear leveling for MLC PCM considering process variation. In *Proceedings of the 51st Annual Design Automation Conference (DAC'14)*. ACM, New York, NY, Article 36, 6 pages. DOI:http://dx.doi.org/10.1145/2593069.2593217

M. Zhao, L. Shi, C. Yang, and C. J. Xue. 2014b. Leveling to the last mile: Near-zero-cost bit level wear leveling for PCM-based main memory. In *Proceedings of the 2014 32nd IEEE International Conference on Computer Design (ICCD'14)*. IEEE, 16–21.

M. Zhao, Y. Xue, C. Yang, and C. J. Xue. 2015. Minimizing MLC PCM write energy for free through profiling-based state remapping. In *Proceedings of the 20th Annual Asia and South Pacific Design Automation Conference (ASP-DAC'15)*. IEEE, 502–507.

K. Zhong, D. Liu, L. Long, X. Zhu, W. Liu, Q. Zhuge, and E. H.-M. Sha. 2015. nCode: Limiting harmful writes to emerging mobile NVRAM through code swapping. In *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition (DATE'15)*. EDA Consortium, San Jose, CA, 1305–1310. http://dl.acm.org/citation.cfm?id=2755753.2757117.

K. Zhong, T. Wang, X. Zhu, L. Long, D. Liu, W. Liu, Z. Shao, and E. H.-M. Sha. 2014. Building high-performance smartphones via non-volatile memory: The swap approach. In *Proceedings of the 14th International Conference on Embedded Software (EMSOFT'14)*. ACM, New York, NY, Article 30, 10 pages. DOI:http://dx.doi.org/10.1145/2656045.2656049

P. Zhou, B. Zhao, J. Yang, and Y. Zhang. 2009. A durable and energy efficient main memory using phase change memory technology. In *36th International Symposium on Computer Architecture (ISCA'09)*. 14–23.

Received July 2015; revised November 2015; accepted February 2016