Phase-Change Memory: An Architectural Perspective

OMER ZILBERBERG, SHLOMO WEISS, and SIVAN TOLEDO, Tel-Aviv University

This article surveys the current state of phase-change memory (PCM) as a nonvolatile memory technology set to replace flash and DRAM in modern computerized systems. It has been researched and developed in the last decade, with researchers providing better architectural designs which address the technology's main challenges—its limited write endurance, potential long latency, high energy writes, power dissipation, and some concerns for memory privacy. Some physical properties of the technology are also discussed, providing a basis for architectural discussions. Also briefly shown are other architectural alternatives, such as FeRAM and MRAM. The designs surveyed in this article include read before write, wear leveling, write cancellation, write pausing, some encryption schemes, and buffer organizations. These allow PCM to stand on its own as a replacement for DRAM as main memory. Designs for hybrid memory systems with both PCM and DRAM are also shown and some designs for SSDs incorporating PCM.

Categories and Subject Descriptors: C.1 [**Computer Systems Organization**]: Processor Architectures

General Terms: Design, Performance

Additional Key Words and Phrases: Computer architecture, phase change memory, main memory, hybrid memory systems

ACM Reference Format:
Zilberberg, O., Weiss, S., and Toledo, S. 2013. Phase-change memory: An architectural perspective. ACM Comput. Surv. 45, 3, Article 29 (June 2013), 33 pages.
DOI: http://dx.doi.org/10.1145/2480741.2480746

1. INTRODUCTION

The purpose of this architectural survey is to provide newcomers to the field of PCM research with an introduction to PCM, its applications and challenges, and some key ideas already published by researchers. It also provides veteran researchers with a quick index of PCM architectural designs—all researchers affiliated with PCM may benefit from this article as a quick reference to other, more extensive works focusing on a desired aspect of PCM.

There are quite a few published surveys regarding PCM. Some offer only a glimpse of the current state of PCM development [Atwood 2010; Lai 2003; Lam 2007], while others offer a more extensive review of PCM technology [Burr et al. 2010; Li and Lam 2011; Ohta 2011; Wong et al. 2010]. However, these surveys mostly cover the physical properties of the PCM cell, such as the crystallization process of the chalcogenide glass, fabrication challenges, and the resistivity and endurance of the PCM. To the best of our knowledge, this is the first survey to cover architectural aspects of incorporating PCM in common applications, such as PCs, laptops, and SSDs.

Corresponding author's address: S. Weiss, Department of Electrical Engineering - Systems, Tel-Aviv University, Tel-Aviv, 69978, Israel; email: weiss@eng.tau.ac.il.
Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies show this notice on the first page or initial screen of a display along with the full citation. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, to redistribute to lists, or to use any component of this work in other works requires prior specific permission and/or a fee. Permissions may be requested from Publications Dept., ACM, Inc., 2 Penn Plaza, Suite 701, New York, NY 10121-0701 USA, fax +1 (212) 869-0481, or permissions@acm.org.
© 2013 ACM 0360-0300/2013/06-ART29 $15.00
DOI: http://dx.doi.org/10.1145/2480741.2480746

Computer systems are present today in every home and business, and are under constant development to satisfy growing demands for more applications with better performance. Such systems, whether in desktop, laptop or handheld computers, whether in large business machines or tiny embedded systems, mostly rely on memory devices either as their main storage system or as a faster access point to slower hard disk drives (HDDs).

While there are several technologies involved in memory systems, depending on the overall systems requirements and limitations, any such technology has both a cost and physical limitation when scaled down or pushed to some performance limit. It is always up to researchers to improve upon the existing technology's limitations, but ultimately realize that at some point the costs of minimizing and optimizing the existing technology may surpass its benefits. That is the exact time the industry may be open for new ideas and innovative technologies that have a prospect of answering the industry's demands for the upcoming years.

phase-change memory (PCM) is a reborn nonvolatile memory technology, initially researched in the 60s and abandoned, only to be picked up again at the turn of the century, and is now considered a promising technology that may replace the existing trusted but aging technologies of both flash memory and DRAM. While it is of some controversy whether flash or DRAM are actually nearing their technological limits, and both have been developed and improved further than some have foreseen, it is likely that at some point in the future they too will reach their respective limits [Freitas and Wilcke 2008], and it is that prospect which motivates PCM researchers to bring the technology to a stable, cost-effective, and competitive state when compared with the standard technologies used in modern computerized systems.

The remainder of this survey is organized as follows. Section 2 offers a short technological survey of leading memory technologies, such as DRAM and flash, while also providing a short survey of so-called competing technologies. Section 3 provides a brief historical introduction of PCM up until commercial developments in recent years. Section 4 surveys PCM cell attributes—its physics, cell designs, and quantitative parameters. Section 5 discusses PCM applications as flash or DRAM replacement. Section 6 explains the challenges in adopting PCM as either flash or DRAM replacement, and gives the basic questions any researcher must strive to answer. Section 7 surveys some key papers attempting to provide architectural designs as solutions for main memory designs. Section 8 complements the architectural designs with a survey of PCM in SSDs. Section 9 concludes the article.

2. OTHER MEMORY TECHNOLOGIES

Prior to elaborating on PCM technology, this section highlights other key memory technologies. Some of these, such as DRAM and flash, are the dominant technologies in today's market, while others, such as FeRAM and MRAM, are aspiring new technologies that are still in prototype or early manufacturing stages, but are still considered as promising alternatives to existing technologies and are therfore constantly under development.

2.1. Quantitative Parameters

Since PCM is constantly developing, it is difficult to present accurate data and measurement for its physical properties, as different works contain varying parameter values. Additionally, any comparison with other technologies should be examined carefully, as some published parameters originate in commercial advertising, while other up-to-date parameters are difficult to obtain because the industry does not publish some of its datasheets. Table I attempts to offer key parameters of the main technologies mentioned here.

Table I. Quantitative Parameters of PCM

Parameter	DRAM	NAND Flash [Boboila and Desnoyers 2010] [Javanifard et al. 2008] [MicronFlash 2008] [Nobunaga et al. 2008]	PCM [Atwood 2010] [Pirovano et al. 2004b]
Scalability	3X nm [Samsung 2011]	2X nm	<1X nm
Read Latency	60ns	25–200us	50–100ns
Write Speed	~1Gb/s	2.5 MB/s	~100MB/s
Endurance	N/A	10^3 to 10^5	10^6 to 10^8 [Atwood 2010], 10^{11} [Pirovano et al. 2004b]

2.2. DRAM

Dynamic random access memory (DRAM) serves as the main memory of personal desktop computers, laptops, gaming consoles, and high-end phones. A DRAM cell consists of a single transistor and single capacitor, allowing great density (in the order of several Gb per chip [Samsung 2009]). DRAM is a volatile memory technology, meaning that it requires routine refreshing of its data every few milliseconds, although some of its stored data may still be recoverable after several seconds, depending on environmental parameters [Halderman et al. 2008].

2.3. Flash Memory

Flash memory is a nonvolatile memory technology whose cells use floating gate devices. Flash is organized in two variations. The first is NOR flash, which is not very dense, and therefore is used mainly for code storage, for example, as boot up software. Since it is not used for data storage, it is not discussed in this article. The other form is NAND flash, which is common as data storage in mobile devices, such as digital cameras and digital audio players and is also becoming widely used in laptop computers. Being nonvolatile, it requires no power to maintain the data stored in it, unlike DRAM. Additionally, it has faster access times and more shock resistance than magnetic disks (HDDs), is considered more durable, and can sustain higher pressures. Its main limitations are the need to erase whole blocks of it (resetting the cell's values to 0), which forces the memory controller to rewrite some of the data after it has been erased, and its limited number of program-erase cycles. Typically, flash devices can withstand about 1,000–10,000 program-erase cycles, with ongoing research to increase that number by a factor [MicronFlash 2008]. It should be noted that the number of program-erase cycles is not an accurate criteria for flash endurance, since flash is typically written iteratively, and its value is checked after several such cycles. Should the value be incorrect after a certain threshold, the operation is deemed as a failure. But by increasing this threshold, it is possible to increase the number of program-erase cycles supported, at the expense of write operation latency.

2.4. Other-Solid-State Memory Alternatives

Besides PCM and NAND flash, there are other alternative memory technologies considered for solid-state memories. Two such leading technologies are FeRAM and MRAM, though PCM has been demonstrated to be feasible with smaller device dimensions than these technologies [Burr et al. 2010].

FeRAM. Ferroelectric RAM, or FeRAM, is a RAM with a similar construction to DRAM. However, FeRAM substitutes DRAM's dielectric layer with a ferroelectric layer. This causes the FeRAM to be a nonvolatile memory. FeRAM was first proposed in the

1950s but has been developed mainly by Ramtron in the mid 2000s. Small scale FeRAM is now commercially available and has been used instead of NOR flash in some chips.

FeRAM has been suggested for a hybrid flash and FeRAM memory architecture in an SSD called Chameleon [Yoon et al. 2008].

MRAM. Magnetoresistive RAM, or MRAM, is a nonvolatile memory technology that does not use electric charge to store data, but instead uses magnetic properties [Huai 2008]. It consists of two ferromagnetic plates and an insulating layer. One layer has a constant polarity, while the other can be switched between polarities, effectively storing a data bit. A 32Mbit RAM device has been demonstrated, and smaller 4Mbit devices have been marketed. MRAM has also been considered as an alternative in CMOS designs [Guo et al. 2010]. This technology is less mature than PCM and FeRAM and is still under development.

3. PCM HISTORY AND CURRENT STATE

Early Years. The initial research on the usage of phase-change materials in memory application is attributed to Stanford Ovshinsky, whose research into certain glassy materials having the property of switching between two phases in a stable manner eventually led to the recognition that some of these materials exhibited a change of phase between an ordered state and a disordered phase. In the late 60s, it was recognized that these two phases had very distinct resistivity, an effect which could be harnessed for both optical and electronic memories. The research culminated in September 1970 with a collaboration between Ovshinsky's company, Energy Conversion Devices, and Intel's Gordon Moore, which presented the first PCM array of 256 bits [Ovshinsky 1968]. However, the technology had not been further developed until recent years, because material quality and high power consumptions had rendered it non-cost-effective and thus impractical when compared with the other popular technologies of that time.

Recent Years. Research into phase-change Materials and their applications as memory devices has picked up significantly in the last decade, as the growing demands on memory devices and scalability issues prod scientists and engineers to look for alternative technologies.

Modern development of PCM began in the late 90s with the forming of Ovonyx, Inc[1], a corporation which licensed all intellectual properties and patents. from Stanford Ovshinsky's Energy Conversion Devices in order to commercialize Ovshinsky's original technology. The 2000s showed other major players entering the field of PCM R&D, such as Intel, Lockheed Martin, and STMicroelectronics. Intel and STMicroelectronics showed a 128Mb PRAM device in 2008. Such devices were released to customers in 2009 and mass produced in 2010 by Numonyx, later purchased by Micron, Inc. These devices were all based on 90nm technology [MicronNumonyx 2010]. Numonyx has announced a 1Gb device based on 45nm technology, but is yet to ship it to customers. Another key player, Samsung, announced in June 2009 that they have joined forces with Numonyx to further develop and market PCM technology. In April 2010, Samsung announced a 512Mb PCM based on 65nm technology [Samsung 2010].

4. ATTRIBUTES

4.1. Physics

Phase-change memory utilizes the special characteristics of chalcogenide glass, which can switch between two distinct states—*amorphous* and *crystalline*. Phase-change material can be switched between its states by applying heat using electrical pulses. The

[1]`http://oronyx.com/corporate`.

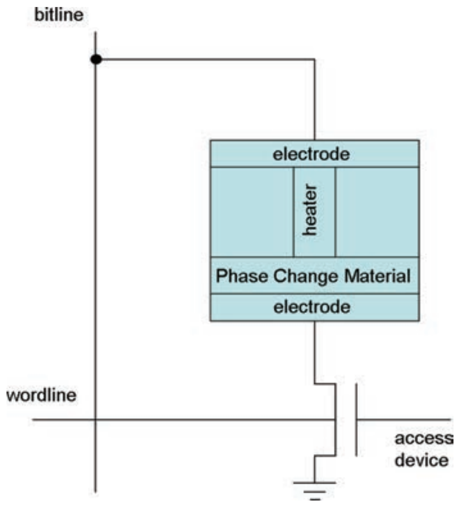

Fig. 1. PCM cell structure with electrodes connecting the phase-change material and the heater with the bitline and wordline.

important distinction between the two states is in their electrical resistivity—the amorphous state is characterized by its high resistivity, and the crystalline state by its low resistivity. Fast crystallizing materials, such as $Ge_2Sb_2Te_5$ (GST), which can crystallize in less than 100ns, can have practical uses as fast memory devices. Moreover, these devices can be fabricated in smaller dimensions than other industrial memory technologies (e.g., DRAM and flash memory) [Burr et al. 2010; Lee et al. 2009; Numonyx 2008; Qureshi et al. 2009a, 2009b; Zhou et al. 2009].

A PCM device generally consists of phase-change material located between two electrodes. Between the bottom electrode and the phase-change material itself, there is a heating element present (see Figure 1). The PCM is characterized by two key temperatures—crystallization temperature and melting temperature. Injecting current into the contact of the PCM and the heating element, thus heating the PCM above the first threshold temperature but below the second one, sends the material into the *crystallized* state and is called the SET operation. Alternatively, applying high voltage (and consequently high power) to the crystallized PCM increases its conductivity, and shutting down the current when a certain threshold voltage has been achieved sends the material back to the *amorphous* state. This is called the RESET operation (see Figure 2). 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 order to read the data stored inside the PCM, low power is applied to it, thus sensing its resistivity. After a SET operation, when the device is in the crystallized states, its resistance is low and its data is equivalent to a logical 1. Accordingly, after a RESET operation, the device is in its high-resistance amorphous state, which constitutes a logical 0.

It has also been shown that PCM can retain its stored resistance value for ten years at a temperature of 110°C, estimating some 300 years of data retention at a normal working temperature of 85°C [Pirovano et al. 2004b].

4.2. Single-Level Cells and Multilevel Cells

The PCM memory cell has so far been shown to store one of two possible values, that is, a single bit. This is the traditional use of a memory cell, called single-level cell (SLC).

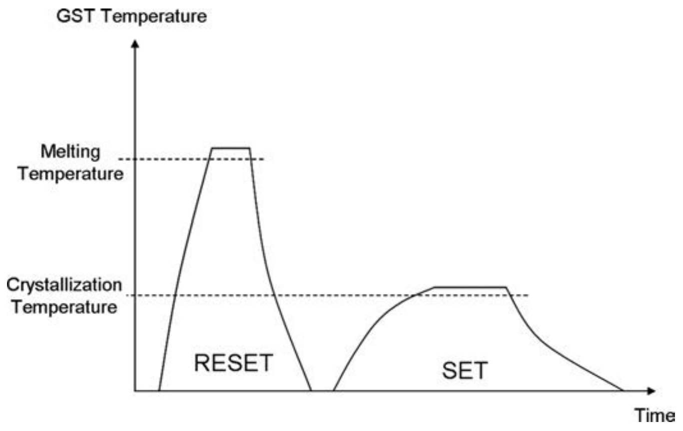

Fig. 2. RESET current heats the PCM above the melting temperature and puts it in the amorphous state. SET current heats it above the crystallization temperature, but below the melting temperature, and puts it in the crystallized state.

There is ongoing research into increasing the storage capabilities of a single PCM cell by utilizing its physical properties to switch between several distinct states. A single PCM cell can be put in four distinct physical states [Bedeschi et al. 2009; Dong and Xie 2011], while sustaining reasonable latency and write endurance limitations. A specially designed programming algorithm allows the PCM cell to be put in intermediate states in terms of resistivity, by exposing the cell to a set of accurately designed electrical pulses. Such design is called a multilevel cell (MLC), and in this design, two bits are stored in a single MLC, effectively doubling the storage capacity of the memory for the same area cost [Lin et al. 2009].

In order to put the PCM cell in the correct state when it is used as an MLC, it is often necessary to use iterative writes [Nirschl et al. 2007; Qureshi et al. 2010a]. This technique stems from the fact that different PCM cells have different responses to electrical pulses, and this physical behavior is unpredictable. It is therefore impossible to program a PCM cell to a desired state with a single electrical programming pulse without some probability for errors. As shown in Figure 3, the iterative writes method calculates the properties of the electrical pulse required to program the PCM cell, but after it has been applied to the cell, the cell's state (resistivity) is sampled to ensure it has been set to the correct state. If that is not the case, a new electrical pulse is calculated and applied, iteratively setting the cell until it reaches the desired state, that is, its resistivity is in the desired range.

5. APPLICATIONS

This section raises the main possible applications for PCM memory, should it arise above the challenges facing it—that of flash memory and of DRAM memory, both core technologies in computerized storage systems, each with its predicted limitations. These applications offer fertile ground for development of PCM, since both flash and DRAM technologies are found abundantly in various systems, and should PCM replace either of them, it could become a key technology in the computerized memory industry.

5.1. An Alternative for Flash

An important market in which flash has grown to be dominant is the solid-state drive (SSD) market. SSDs are data storage devices which store persistent data. Unlike HDDs, they have no moving parts and are therefore more resistant to physical shocks, are

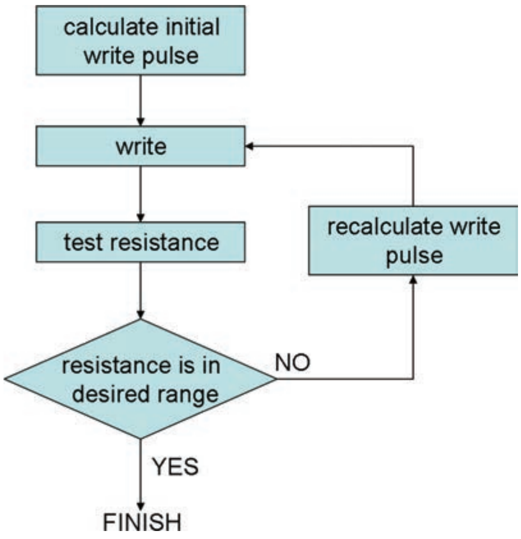

Fig. 3. Iterative writes algorithm for PCM MLC writes.

quieter, and have lower access times and latencies than HDDs. These factors have led SSDs to be used as a replacement for HDDs in some laptop computers, with a maximal capacity of up to 512GB [Toshiba 2009]. While these advances speed up development of flash memory, they also increase the demands on flash technology, ever bringing it closer to its cost-effect limit. Even as multilevel cells are researched for PCM, they exhibit lower resistance and write speed, and ultimately, flash memory may reach its limit or at least reach a critical point where other technologies, such as PCM, may prove to be more effective for the same cost [Burr et al. 2010]. When flash memory would reach this critical point is a controversial issue, but growing research into alternative technologies brings that point nearer.

PCM, being a nonvolatile memory, which at least theoretically exhibits superior endurance and latency when compared with flash memory, offers a viable alternative to flash memory [Bez et al. 2010]. If the challenges in producing it in lower scales and fully integrating it as a memory chip in portable devices or as an SSD in laptop computers are met, it may someday replace flash memory or at least some of its uses. Until such a time, PCM could also be used in hybrid flash-PCM architectures to counter some of the disadvantages of flash memory. Such designs are further explained in Section 8.

5.2. An Alternative for DRAM

Modern computer systems usually employ several processor cores on a single chip, with many systems also consisting of several such chips. This ever-growing computational power allows for more concurrent threads and processes, which naturally leads to an increased demand for fast, available data. This burden falls mainly on the main memory of the computer system, which must meet demands of both quantity and availability—more concurrent processes require more data, and going to the hard disk drive for data too often cancels out any increase in computational power, since it is too slow. The preferred choice for the system's main memory is almost always DRAM (see Figure 4(a)), and has been such for several decades.

However, in recent years, DRAM has approached its limits in scalability and performance. It has already fallen behind other technologies when scaling technology down to 4x nm, and even more so when incorporated into 3x nm technology [Atwood 2010; Burr et al. 2010]. The time may soon come when DRAM can no longer fulfill the increasing demands of multicore processors with respectively increasing workloads. These limits of DRAM in the one transistor one capacitor cell design are due to both capacitor scaling and transistor scaling. Future process scaling depends on the ability to manufacture a small enough capacitor that still stores a sufficient charge for reliably sensing the bitline, and also on the access transistor that, as it keeps scaling down, also becomes more difficult to ensure DRAM retention times, owing to the transistor's increased subthreshold leakage. It is already uncertain whether DRAM could be manufactured beyond the 40nm technology, while PCM is projected to extend to 9nm [ITRS 2007; Lee et al. 2009].

In order to prepare for that time, PCM is considered for replacing DRAM in the role of the computer system's main memory. There is, of course, quite a lot of PCM research to do before PCM can be declared a suitable successor and DRAM is abandoned. First, it must be noted that PCM requires great energy for a single write operation, owing to the need to heat it in order to switch its state. On the other hand, with PCM it is possible to write a single bit, while in DRAM multiple banks are accessed for every write operation. Additionally, PCM is a nonvolatile memory, while DRAM, a volatile memory, requires periodic rewrites of its data. Next, unlike DRAM, PCM has a limited write endurance, and if not used wisely, may become faulty after a shore period of time. Finally, when incorporating PCM into main memory, one must consider PCM access times. PCM is about four times slower than DRAM [Qureshi et al. 2009b], a problem which must be addressed before it is used as main memory, or overall system performance will suffer.

The challenges involving PCM in its various applications are further developed in the following sections. Should these challenges be met efficiently, PCM could well prove to be DRAM's successor as the main memory of future computer systems. In the meantime, PCM is also used in hybrid DRAM and PCM architecures, in which PCM is used where DRAM is disadvantageous. Such designs are further explained in Section 7.2. Finally, PCM is also suggested as a replacement for DRAM as the preferred device for hybrid checkpointing in massively parallel processing (MPP) systems [Dong et al. 2011].

6. CHALLENGES

6.1. Finite Write Endurance - Hard Errors

Writing the phase-change memory is the primary contributor to its wear, reducing its lifetime. The current injection required to write a PCM cell has a degrading thermal effect on the contact between the electrode and storage area. This in turn increases current variance, hence increasing resistance variability. The immediate consequence is that PCM can only endure a limited amount of writes before its stored data can no longer be read reliably. The exact endurance differs between manufacturers and manufacturing techniques, but is in general on the order of 10^7 to 10^8 writes. The lifetime of the PCM varies with its applications, but as shown in Zhou et al. [2009], it may be as short as a few days, seemingly rendering the PCM an inappropriate replacement for main memory. Such a device failure is considered to be a *hard error*, as it is a final fault in the device, and the device itself is no longer functional.

Studying statistic properties of main memory write accesses shows that these accesses suffer from the locality property. Memory writes are not evenly distributed among the entire address space, but tend to cluster around *hot* cells. This in turn decreases the lifetime of PCM, because these hot cells use up their available writes quickly, losing their reliability. Once a single PCM cell can no longer be considered reliable, it may render an entire page or even the entire memory unusable, depending on whether some error recovery scheme is in use. If the PCM or some external mechanism were to distribute write operations more evenly across the available memory, they would also evenly distribute the wear, increasing PCM lifetime considerably.

The finite write endurance, which may lead to device failure, provides another challenge which must be dealt with. PCM architectural studies usually evaluate their proposed works with typical memory write profiles. These may also include profiles which make great demands on memory, and some even suffer from the localization property, which is a greater hazard to PCM endurance. But an important aspect of PCM design should give an answer to the possibility of a malicious attack on the memory system [Qureshi et al. 2009a]. Knowing the internal structure of the memory system and possibly the architectural design, which is supposed to compensate for the short PCM lifetime, may allow a potential adversary to implement a memory-demanding application that exploits such design flaws and continuously writes to the same area of PCM, or even to a specific memory line. If unprepared for, such an attack may lead to device failure in several seconds.

6.2. Long Latency

A key parameter of any memory architecture in almost any computerized application is the latency of the memory system. A vast amount of studies and analyses have been made on the effect of memory latency on the general performance of a computer system, and it is beyond the scope of this article to survey these. However, it is clear that increased latency in servicing a memory read or write request eventually leads to either increased latency in servicing an application's memory read request, or a growing occupancy of write buffers. Architectural solutions to these problems, whether in the form of improving existing architectural designs or inserting new hardware devices to mitigate the costs of increased latency, are bound to increase both manufacturing costs and energy requirements of the associated devices, limiting their scalability and incurring a further cost in hardware to sustain the increased energy costs.

In order for PCM to be a viable alternative for either flash memory or DRAM, its latency must be comparable with that of its predecessors, both in read and write operations, while also considering different behaviors for SLC and MLC PCM designs.

6.3. High-Energy Writes and Power Dissipation

With the growing demands on the memory system in terms of both quantity and latency, the power required to support these functionalities is taking an ever-increasing portion of the total power required by the system. In some servers, as much as 40% of total power is required by the memory system [Lefurgy et al. 2003]. In typical memory systems, which employ volatile memory technology as their main memory, the main memory dissipates both leakage energy and dynamic energy, and since the leakage energy dissipation grows with the memory capacity, it can reach the level of the dynamic energy dissipation [Thoziyoor et al. 2008]. New architectural designs for main memory should therefore focus on decreasing the amount of leakage memory dissipated. One solution for this problem is to use nonvolatile memory technologies such as PCM.

When using a nonvolatile memory technology such as PCM to suppress the dissipation of leakage energy, it is imperative to check its consumption of dynamic energy. PCM experimentations show that write operations take as much as 150–300 times more power than DRAM, depending on the bit value written [Zhou et al. 2009]. Failure to suppress dynamic energy dissipation in PCM main memory may result in the consumption of all power saved on leakage energy, and the total energy dissipated may be the same as DRAM, or even worse.

6.4. Privacy

Typical computer systems use DRAM as their main memory, which is volatile. Even so, it has been demonstrated that data can be retrieved from a DRAM memory chip even after power has been cut off to the chip, allowing a malicious retrieval of data from the chip, such as disk encryption keys that are kept in memory for easy access [Halderman et al. 2008]. While there are methods to protect volatile memories from such malicious attacks, PCM is a nonvolatile memory, which can retain its stored data for several years [Pirovano et al. 2004b]. Without some protection scheme, this could prove to be a serious security breach, as anyone with physical access to the machine can reboot it or cut its power, relaunch it with some modified kernel, or otherwise remove the PCM chip entirely and install it in some other machine, and with any of those techniques restore critical data, which has been stored in memory for temporary usage [Seong et al. 2010a; Seznec 2010]. While processors and disks may be secured, any encryption key or other private and sensitive data may be stored in the main memory by an application or the operating system, rendering processor and disk securities useless.

6.5. Soft Errors

Soft errors in the PCM device represent phenomena that, over time, lead to a wrong value read result, but unlike hard errors, do not represent some intrinsic failure in the PCM device and can be remedied.

Resistance Drift. As described in Section (4), a PCM cell can be in either one of two states—crystalline and amorphous. It has been observed [Wong et al. 2010] that device conductivity remains fairly constant in the crystalline state. However, the device resistivity changes substantially over time in the amorphous state. While this is relatively tolerable in single-level cells (SLC), it may prove hazardous for multilevel cell (MLC) designs, as resistance drift could eventually lead a read operation into misreading the stored value. This physical aspect of the PCM device is somewhat beyond the scope of this architectural survey, but should be addressed by anyone employing PCM in an MLC architecture. Further explanations of the phenomenon and attempts to cope with it can be found [Burr et al. 2010; Wong et al. 2010; Ielmini et al. 2007; Pirovano et al. 2004a].

Process Variation. The main attraction of PCM as a substitute for existing technologies in future memory applications resides in its increased scalability. However, as the devices become smaller, they are more susceptible to variations in the process of fabrication. Since it is impossible to have exact control over the fabrication of an entire wafer of PCM devices, target parameters are accepted within some predetermined error ranges. Consequently, PCM devices may vary in thermal and electrical properties, such as set and reset currents and resistivity in both amorphous and crystalline states. It would be prudent of anyone designing PCM-based architectures to be aware of these variations, and to design solutions that are robust to such variations. Further explanations and solutions to this problem are beyond the scope of this architectural survey and can be found in other physically-oriented surveys (such as [Burr et al. 2010; Wong et al. 2010]).

7. MAIN MEMORY DESIGNS

7.1. PCM Main Memory

Conventional main memory design usually employs DRAM as its main memory (see Figure 4(a)). This section describes architectural design solutions to problems which arise when using PCM as the main memory (see Figure 4(b)).

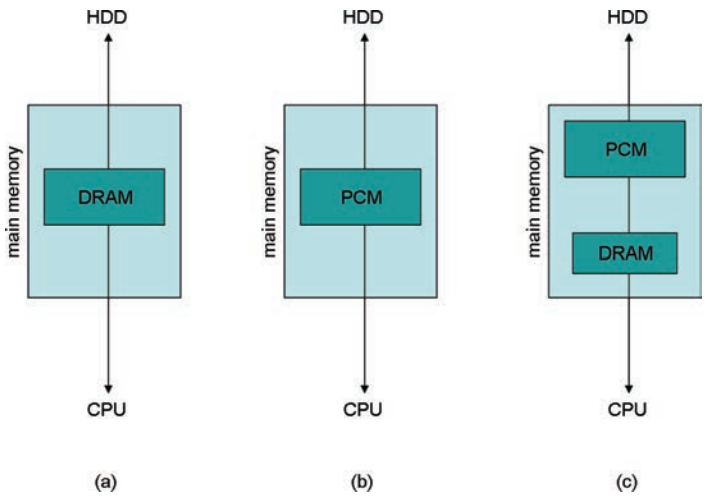

Fig. 4. (a) Conventional main memory design with DRAM. (b) PCM used as main memory. (c) Hybrid PCM and DRAM memory.

7.1.1. Bit-Level Read Before Write

PCM Design Opportunities. In conventional DRAM and flash, each write operation updates an entire page or block, and write operations on a bit granularity level are unavailable. Furthermore, the read and write latencies for DRAM memory access operations are similar. PCM exhibits different attributes. Depending on the device manufacturer, it may be bit-accessible (though it is sometimes addressable on a page-level granularity, to emulate NOR flash [NumonyxOmneo 2010]), and its read latency is far smaller than its write latency. These characteristics provide an opportunity for new design schemes focused on reducing PCM power consumption and increasing its lifetime [Cho and Lee 2009; Yang et al. 2007; Zhou et al. 2009].

Redundant Writes. A DRAM or flash write operation affects an entire page or block. Owing to the value locality property, this means that individual bits are often rewritten with the same value stored in them. While this is of little importance in DRAM, it provides an opportunity for increasing PCM lifetime, by removing these redundant writes on a bit-level. Since read operations in PCM are much faster than write operations, it is relatively cheap to precede each write operation with a read operation, as depicted in Figure 5. It is thus possible to write a cell only when its value actually changes, instead of on any write request. The removal of these redundant write operations on a bit-level may increase PCM lifetime by a factor of 4.5 [Zhou et al. 2009]. It should be noted that this method is new to PCM and is inappropriate for DRAM, because unlike PCM, the time required for a read operation in DRAM is roughly the same as for a write operation, rendering the read-before-write technique costly in terms of performance. This basic and simple reduction in memory write operations is common to many designs and can be found under several names, including Read-Before-Write [Zhou et al. 2009], Flip-N-Write [Cho and Lee 2009], and Data-Comparison-Write Scheme [Yang et al. 2007].

Flip-N-Write. It has been shown how to improve upon memory write energy, bandwidth, and endurance by inspecting the currently written bit value and comparing it with the new required value. These parameters can be further improved if the

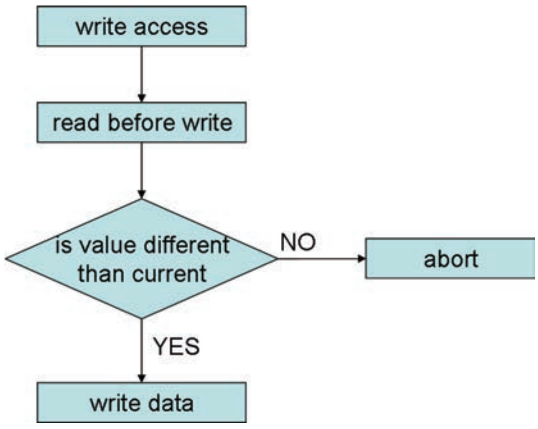

Fig. 5. Typical read before write algorithm.

comparison is done for an entire memory word and adding a *flip* mechanism. The key idea here is adding a single bit for every memory word stored in the system. If, on a write access, more than half of the word's bits are about to be changed, then the flipped word is written instead, and the flip flag is set. This effectively limits the amount of bits written on each write operation to half of the available bits at most, and further decreases harmful write operations and the accompanying power consumption. In order to decide which word to write—the original or the flipped version—the Hamming distance between the currently stored word and the two optional new words is calculated, including the effect on the flip bit. The word which required less bit changes is chosen and is written. Note that this design requires an additional bit per every memory word, which for a 16-bit PCM word requires an increase of 7% in area cost.

7.1.2. Wear Leveling

Row Shifting. The locality of write operations causes *hot* cells to be written more frequently than other cells. A possible compensation for this property is the shifting of memory cells inside a single memory row. Shifting allows memory writes to be distributed more evenly across the cells in a row and avoids peaks in memory writes statistics. There are several factors to consider when implementing row shifting. First is the shift frequency. It is possible to shift memory after every write or keep a counter for write operations and shift only after every *n* write operations. Shifting after every write may very well worsen the wear of the PCM, because it incurs additional bit changes and may cancel the advantages of reading before writing, as previously described. An additional factor to consider is the shift granularity. A shift may be on a single-bit granularity or any courser granularity. As studied in Zhou et al. [2009], too fine a granularity is not advantageous, since hot cells tend to be clustered together (e.g., least significant bits are usually written more often than most significant ones), and shifting by a single bit may cause one hot cell's write operations to affect another hot cell, which misses the point of the shift operation.

Segment Swapping. When observing memory write statistics, it is noticeable that in some write profiles entire, pages may be considered hot. In these pages, a local mechanism to vary write operations does not solve the endurance problem, since there are not enough cold cells to distribute the writes. However, studying the memory writes on a coarser granularity may show that there are entire memory pages which are scarcely written. Thus, swapping between such pages may prove advantageous in

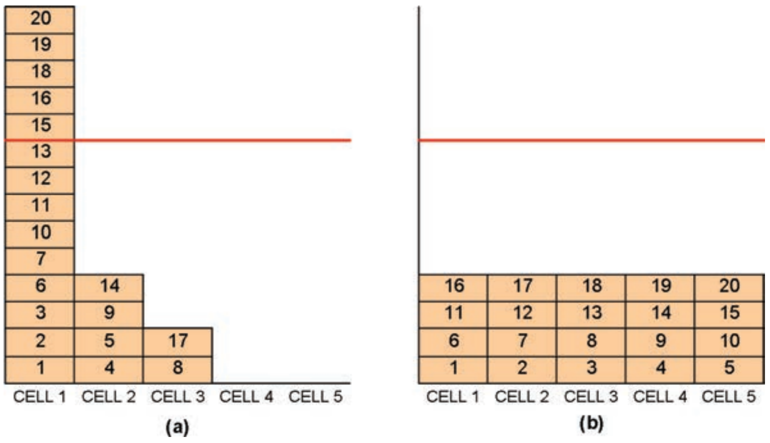

Fig. 6. Illustration of wear leveling. (a) Without wear leveling, the endurance limit is reached for cell 1 rendering the chip unusable, while other cells are barely used. (b) With an even usage of all cells, the limit is not reached, and the chip is functioning.

terms of memory writes distribution. The swap operation has several key properties to consider. First, the size of the swapped segment. If the segment is too small, there would be many segments. In order to identify segments that have been written more frequently than other segments, some write counter should be kept for each segment. A large amount of segments would incur many such write counters, and the overhead in terms of both area and performance (latency caused by updating the counters and comparing their values to find a relatively unused segment) would render the entire segment swapping mechanism too costly and inefficient. On the other hand, choosing to swap a segment whose size is too large may incur overhead in the form of additional writes—a large segment, which may be considered hot, might actually contain some parts which are relatively cold and would be swapped needlessly. Implementation wise, Zhou et al. [2009] suggest implementing the logic required for segment swapping inside the memory controller, while keeping a write counter for each segment, along with an additional register to hold the last time the segment was swapped. This is required to prevent a cold segment from being swapped too frequently.

Start-Gap. Most wear-leveling techniques keep track of write operations on a per-line statistic, with the aid of dedicated tables. This incurs an additional linear cost in space for maintaining these tables, and furthermore takes an additional toll in the form of increased latency, for every read and write memory access operation requires the additional table lookup to translate the required logical address to its physical location. This is translation is performed in the memory controller and is in addition to any such translation performed by virtual memory mechanisms. The concept of Start-Gap [Qureshi et al. 2009a] sets out to eliminate these costly overheads, or at least to bring them to a bearable minimum. The key idea is to add an algebraic mapping between logical and physical memory addresses, which is simple and cheap to maintain, and to consult on every memory access.

Start-Gap is implemented by the addition of two designated registers—Start and Gap—with both holding indices that allow calculation of the address mapping. Gap holds the index of a memory line which is currently not in use. When calculating the physical address, Gap should always be skipped. At first, Gap points to an index just outside of the currently available memory range, for example, for a 16-line memory (with lines 0–15 in use), Gap would point to line 16, which is unused, as shown in

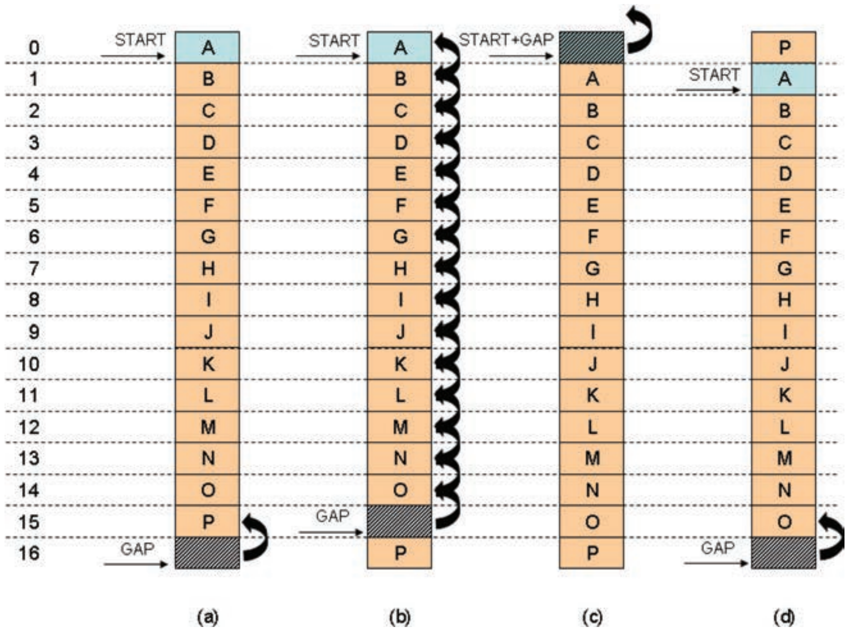

Fig. 7. Illustration of Start-Gap. (a) Initial state; (b) Gap index is decremented by 1; (c) Gap reaches the first memory line; (d) Gap is reset to 16 and Start is incremented by 1.

Figure 7(a). After several write operations, Gap is decremented by 1, and the data it now points to is stored instead in the previous Gap address. In terms of this example, should Gap be decremented from 16 to 15, the data stored in line 15 would be copied to address 16 and would henceforth be read from that address, as shown in Figure 7(b). Gap continues to be decremented throughout the entire memory range until it reaches the first memory line, as shown in Figure 7(c). The next memory write operation, which should further decrement Gap according to the Start-Gap algorithm, would instead reset the Gap address to its original value, for example, 16. Additionally, the Start register, which initially holds the value of 0, is now incremented by 1, as shown in Figure 7(d). This means that the memory addresses should be translated as if starting from the Start address, instead of 0.

Note that the Start-Gap technique requires additional spare lines to function properly and assumes that the memory design already has such spare lines. Otherwise they should be added, at an additional cost. Furthermore, the Gap register is not incremented after every write, because that would effectively double write operation (i.e., register write on every memory write). Therefore, Gap is updated only every ϕ write operations. For a choice of $\phi = 100$, the additional costs in latency and power consumption are only taken for 1% of write operations. However, this requires counting write operations using some 7-bit counter. Whenever the counter reaches ϕ, the Gap value is incremented according to the aforementioned algorithm, and the write-operations counter is reset.

Start-Gap, in its given form, is ignorant of the spatial correlation of heavily written memory addresses. It is typical in memory write operation profiles for certain memory areas to be "hotter" than others, and the Start-Gap wear-leveling mechanism is localized and on its own is not sufficient to handle such write profiles. It would simply switch one heavily written memory line with another. It is therefore necessary to upgrade the Start-Gap wear-leveling mechanism with some randomization mechanism.

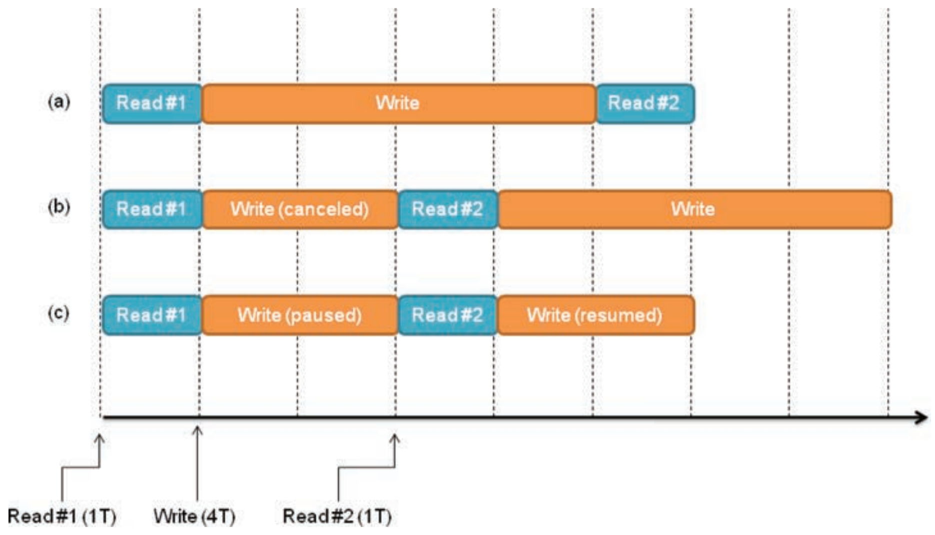

Fig. 8. (a) First-come-first-serve scheduling; (b) write cancellation; (c) write pausing.

The enhanced version of Start-Gap defines some randomization algorithm, using either a Feistel-Network based randomization, which is common in ciphers (DES), or a random invertible matrix (RIB). Either of these elements allows a pseudorandom memory of logical addresses to some intermediate address, with the added value that two subsequent logical memory addresses, whose write usage operations have great correlation, are now mapped to different unrelated addresses. So in effect, two nearby intermediate addresses are far less likely to have similar memory write profiles, thus removing the locality effect. After such a randomized translation is applied, Start-Gap can be used to level the wear of any locally hot memory line, with less likelihood that reducing wear of one critical line would increase the wear of another critical one.

Start-Gap also provides a potential solution for the possibility of an adversary attacking the memory system. Should a malicious adversary be knowledgeable in the internal design of the memory system, including the Start-Gap technique and its parameters, it would be possible for that adversary to bring the PCM to device failure in several seconds, due to continuous write operations to the same memory line. If, however, Start-Gap is separated to different regions of the memory system and Start-Gap registers are duplicated for each such region separately, then it is possible to ensure that the Gap would move the assaulted line before it reaches its memory write operations limit. The required overhead is still far less than that of other wear-level mechanisms, though it now depends linearly on the memory size and is no longer constant.

7.1.3. Write Cancellation and Write Pausing. PCM memory accesses, as opposed to DRAM memory accesses, typically have a much higher write latency than read latency. Additionally, once a write operation has begun for some memory bank, a subsequent read access request for the same bank must be delayed until the write request has been serviced, as can be seen in Figure 8(a). The increase in read requests' latency because of dominant write requests is 2.3x. In order to reduce the overall latency of read requests, which is performance critical in the system, it is possible to cancel or pause a write request even as it is being serviced [Qureshi et al. 2010a].

Write Cancellation. The key element of this design scheme is allowing the PCM cells the ability to cancel an ongoing write operation when a dedicated signal is sent. Thus, whenever a read request arrives for a bank which is currently being written to, the write operation is canceled and the read request is serviced, as can be seen in Figure 8(b). While such a cancellation may lead to inconsistency in the memory contents, as long as the desired written values are kept in the write queue and subsequent read requests are serviced from the write queue, the memory is coherent. To prevent an overflow of the write queue buffer, another mechanism monitors the queue's occupancy, and should it reach some near-limit threshold, its write operations would be performed regardless of subsequent read requests to the same bank (i.e., the write cancellation policy is withheld). Further improvement of this mechanism arises from monitoring each write request to keep track of how long it has been in operation. With this statistic it is possible to prevent the cancellation of write requests which are nearing completion, for example, a write request which is 90% complete should not be canceled. Further improvement to the mechanism could be achieved by an adaptive write cancellation scheme, which sets the cancellation threshold dynamically according to the number of entries in the write queue buffer. This way writes are forced (i.e., there is no write cancellation) if the queue is nearly full, but more writes are cancelled if the queue is relatively empty. The threshold-based designs reduce the number of writes cancelled, and thus the number of write operations which are redone, effectively decreasing the energy overhead of the write cancellation mechanism.

Write Pausing. Due to the iterative writes property of the PCM cell, its write procedure can be suspended and resumed. As seen in Figure 3, this allows a logical separation of the write operation to several distinct operations, which can be paused inbetween. This property is most beneficial for improving the read latency in PCM cells, because whenever a read request arrives for a memory bank which is now serving a write request, the write request can be paused, allowing the read operation to be serviced, after which the write operation can be resumed until completion without ever canceling the write process, as seen in Figure 8(c). Thus the time and energy already spent on the write operation need not be lost. In order to implement this policy, the iterative writes algorithm is edited to allow the insertion of a new stage. Before the newly calculated programming pulse is applied to the PCM cell, there is an additional check whether a read request is pending. If there is such a request, it is serviced before the pulse is applied. A final optimization for this technique could be achieved by combining it with write cancellation. Because write operation cannot be stopped at any time, but only in between iterative writes, an intra-iteration write cancellation scheme can be applied to cancel a write iteration (and not the whole write request) to allow servicing of read requests, if the read request arrives when there is no near option for pausing the write iteration.

7.1.4. Error Recovery, Encryption Schemes, and ECC. The limited write endurance of PCM can often lead to faulty memory cells. Even if mitigated with wear-leveling techniques, such as described in Section 7.1.2, it is likely that at some point one or more cells would become faulty, seemingly rendering the entire PCM chip unusable. The following section offers architectural techniques for contending with this scenario and effectively increasing PCM lifetime.

Dynamically Replicated Memory. Assuming that it is possible to identify faulty PCM cells in a memory page, such errors can be recovered from by adding another layer of indirection and pairing two real PCM pages to appear as one physical page for all purposes. Dynamically Replicated Memory [Ipek et al. 2010] does this by adding a global *physical* to *real* memory page translation table, which is stored in the PCM itself. Thus, every memory access to a physical page is directed to either of two real memory pages. Two pages are paired in the table if they are found to be compatible, that is, there is no single byte which is faulty in both of them. Since PCM is nonvolatile, this new memory translation table must also be kept on it, as it too must not be volatile. Since this method cannot ensure zero faults in the memory translation table itself, the table is stored in triplicate copies on the PCM. Faulty pages and the pairing of two real pages into a single physical page are handled by the OS. Finally, in order for this scheme to be feasible, the memory controller must be able to identify faulty bytes in a PCM page. This is achieved by a standard parity bit mechanism, which is common in DRAM, and assisted by an additional read operation after every write, which ensures that either the write operation was successful or that the parity mechanism noted the error. Should the controller note that the parity mechanism is oblivious to the byte error, another single bit would be intentionally inverted, prodding the parity mechanism into recognizing the error.

The Effect of Encryption on Wear-Leveling. In computer systems where privacy and security are demanded, it is common to use some encryption scheme on the main memory to protect its contents from being extracted and used maliciously [Kong and Zhou 2010; Seong et al. 2010a; Seznec 2010]. While this may help to increase the system's privacy, in some cases it has the side effect of seriously reducing the effectiveness of wear-leveling schemes. The basic problem of encryption in this context is its changing of the data stored in the memory. Encryption often works on block sizes, which are smaller then the block sizes of the cache level. Additionally, a key principle of most encryption schemes is the principle of diffusion, which dictates that a change to a single bit in the block affect the entire block, or at least most of it. This is completely orthogonal to the assumptions on which wear-leveling schemes below cache-line granularity are based. Such schemes, such as the removal of redundant bytes on a bit-level, are primarily based on the statistical properties of write operations which state that in most blocks written to the memory, an abundant amount of bits remain unchanged by the write operation. This assumption is naturally invalidated by the introduction of an encryption scheme, as even a change in a single bit may toggle a change in most of the bits of the encrypted block. Encryption also disables the partial writes scheme, since the entire cache line is changed and needs to be rewritten even if only a single word is dirty.

Block-Level Counters. In order to mitigate the negative effect of encryption schemes on the wear-leveling techniques described, it has been suggested to provide the encryption scheme with additional counters on a block level, on top of the single counter on a cache-line level. In this way, when a write-back occurs, only the counters of the written-back block are updated, and there is no need to rewrite the entire cache, except on the rare event that the block-level counter overflows, in which case the cache-line level counter is incremented and the entire cache-line is re-encrypted and rewritten to memory.

ECC Increases Lifetime. When considering the limited lifetime on the PCM, even if some architectural designs are adopted to lengthen it, the question of what to do with the device when it finally does fail still needs to be answered. In order to continue using the system in a stable and reliable manner, it is possible to introduce error-correction codes (ECCs) to mitigate PCM device failures by correcting bit errors in memory read data. Ideally, if enough space is allocated in favor of the ECC algorithm, it can make the PCM robust to bit failures. However, increased space obviously increases cost and reduces PCM effectiveness. Additionally, during PCM's initial lifetime, it has only a few bit errors at most, and space allocated in favor of ECC is wasted.

It would therefore seem advantageous to apply some adaptive ECC scheme to the PCM which takes into account the amount of write operations to the PCM, or specifically to any of its lines, and utilizes a sufficiently strong ECC to fix any expected bit errors. This can be done by dividing the memory addresses into several groups, with each group allocating some of its memory pages in favor of ECC. Each group also keeps some statistics on the write operations to it which are kept in the form of a counter, and when this counter reaches some predetermined factor, it causes an interrupt to the OS, which is aware of the adaptive ECC scheme and can reorganize the group so as to allocate more space for the ECC, until after some final threshold, the entire group is marked as faulty and no memory is ever allocated from it by the OS. It is claimed that using the PCM itself to store the ECC is beneficial because it saves the need for extra hardware and also because it is quite unlikely for the ECC pages themselves to reach the write limit threshold that is defined in this scheme.

Encryption and ECC Combined. As discussed, use of adaptive ECC requires counters to keep track of write operations. This can be done either by keeping a counter for each cache line, which is costly in terms of space, or by keeping a single counter for the entire page, which requires very little space but is far less accurate since writes to different lines in the same page are accumulated, instead of being considered a single write.

In order to improve the accuracy of this statistic without having to store additional counters on a cache-line level, the aforementioned encryption scheme can be utilized. Each cache line already has some counter for encryption purposes, which is updated by write operations to that line and in effect counts write accesses to that line. With the addition of some global counter on a page-level granularity, ECC can keep track of write accesses to each page with nearly no additional hardware cost other than costs already incurred by the encryption scheme.

Error-Correction Pointers (ECP). The usage of error-correction codes (ECC) is commonly employed to correct errors in DRAM-based memories, and has also been suggested for PCM. However, it should be noted that DRAM and PCM memories exhibit different behaviors regarding errors. The charge-based DRAM is commonly affected by soft errors, that is, leakage-related errors in a write operation, which affect only the results of that particular write operation and are reset on the next write operation. PCM, however, is prone to hard errors, that is, specific memory cells can fail completely when written too many times. It would therefore be more efficient to mark these cells as unusable and replace them with other functioning cells, rather than attempting to correct the error using ECC, especially since ECC requires additional writes and may be contradictory to other wear-leveling techniques employed. The ECP [Schechter et al. 2010] scheme stores pointers to defective memory cells and their corresponding data. Thus, whenever a read-write-read operation fails, a pointer is stored to the faulty cell with the required written data, and upon read operation, the correction pointer is considered prior to the actual data stored in the PCM cell. This scheme can also be used to mitigate errors in the pointers themselves.

Stuck-At-Fault Error Recovery (SAFER). A data block which contains faulty bits (i.e., bits stuck at a certain value) can be partitioned into several sub-blocks, where each such sub-block has at most one faulty bit [Seong et al. 2010b]. In this manner, it is possible to employ a simple technique to recover from a single-bit error on such sub-blocks, avoiding rendering the entire block or page unusable. The trick to overcoming a single fault is to keep a single inversion bit for every such sub-block. On every write operation, the faulty bit is checked. If it holds the required value (there is generally a 50% chance of that occuring), the sub-block can be read later without error. Otherwise, if the faulty bit holds the inverted value of the written bit, the entire sub-block can be inverted and the inversion bit set. Subsequently, a read operation would recognize that the sub-block is inverted and invert back the read data. In order to prevent duplicate write operations on every faulty sub-block, a dedicated cache can hold addresses of faulty bits and their stuck-at values, so that the value of the inversion bit and the written data could be determined before the actual write operation.

Fine-Grained Remapping with ECC and Embedded Pointers (FREE-p). Previous techniques have employed some dedicated memory storage area in order to either store some backup for faulty memory bytes or simply to keep track of the addresses of these bytes. FREE-p [Yoon et al. 2011] offers to use the faulty memory block itself in order to embed a pointer to the replacement memory block. In this manner, a faulty 64B block can hold a 64b address pointer to its replacement. Because of the high space redundancy used for the pointer, it is possible to use a powerful ECC to ensure correct encoding and decoding of the pointer, such as a 7-modular-redundancy (7MR) ECC which repeats the 64b address 7 times in the allocated 64B block. An additional bit is reserved for each 64B block to mark it as faulty. When a read operation occurs, the faulty bit is read first to determine whether the actual data should be read or an indirection followed. If errors are detected in the block pointed to, multiple indirection pointers could be used in a linked-list fashion, with an additional mechanism that updates the first pointer to point at the last segment in the chained loop, for added efficiency of read operations. If a 64B block becomes too faulty to store even the small 64b address, the entire page is rendered unusable and replaced, as in previously described schemes. An advantage of FREE-p over other error recovery schemes rests in its ability to recover from many local errors, such as a single cache-line failure.

7.1.5. Buffer Organization and Partial Writes. In using PCM as an alternative to DRAM main memory, it is beneficial to consider the memory array architectural organization. A PCM cell, very much like DRAM, is organized into different banks, blocks, and sub-blocks. These blocks share among them the peripheral circuitry required to operate the memory array, most notably the sense amplifiers and write drivers. These large devices take a heavy toll on PCM [Lee et al. 2009], and an efficient buffer organization should be adopted to reduce usage of these devices.

In this typical memory architecture, reading a row from the memory requires latching it to a memory buffer, from whence it is read and to which it is written. Any read or write access to an unbuffered row requires that the buffer would be evicted in order to make room for the next accessed row. In DRAM memory, every such row eviction requires that the data evicted from the row be rewritten to memory, owing to DRAM's destructive read operations. In a PCM-based architecture, however, it is seldom required that the data be rewritten, except when it has changed.

In order to compete with DRAM's superior energy write costs and access latencies while maintaining the same area costs, the described architecture should be altered. Using the same space occupied by the buffered row, along with its accompanying sense amplifiers, it is possible to maintain narrower buffers with multiple buffer rows. The amount of sense amplifiers required to operate the buffer is linearly dependent on the buffer width, hence the great cost reduction when narrowing the buffer. This cost reduction can be utilized to support several buffer rows with full associativity between them, which reduces the amount of memory write operations due to buffer eviction, greatly increasing average PCM access performance.

Narrower buffers reduce the PCM energy writes and help bring it to the same level as that of DRAM, but at a performance cost—a narrow buffer has poorer utilization of memory accesses spatial locality and provides fewer opportunities for memory write accesses coalescing. Mitigating these performance costs with a fully associative, multiple-row design with the same area cost as the original architecture can decrease PCM delay penalties, when compared with DRAM design, from 1.60x to 1.16x, and also decrease memory energy costs from 2.2x to 1.0x [Lee et al. 2009].

This design can be further developed to account for the limited PCM write endurance in the form of partial writes. This technique keeps track of the dirty data through all cache levels up until they are written to the memory banks. In this manner, evicted data from any cache level is only written to the main PCM memory if it has changed, reducing the amount of unnecessary write operations. This scheme is considered on two granularities. Lowest-level cache bitline size (64B) and word size (4B). It is not considered on a per-bit granularity, since this incurs too great an overhead in stored metadata or otherwise requires the use of comparators. Keeping track of memory writes on the suggested granularities is possible by keeping track of memory write instructions from the microprocessor pipeline. Using partial writes on a lowest-level cache bitline size requires a 0.2 percent increase in memory usage, while utilizing it on a word-size granularity has an increased 3.1 percent overhead [Lee et al. 2009].

7.1.6. Morphable Memory System. As described in Section 4.2, PCM cells can be either single-level (SLC) or multilevel (MLC). While MLCs provide higher density, the iterative algorithms required to accurately read and write data to these cells take a toll on the device latency. Depending on the load services, some systems may benefit from the use of MLCs with their higher density, while some workloads which are not capacity-intensive may result in degraded performance due to usage of MLCs. The purpose of the Morphable Memory System (MMS) is to benefit from both designs by incorporating both of them into the system, while dynamically setting the portion of PCM cells used as MLCs to account for varying workloads [Qureshi et al. 2010b].

In MMS, the memory pages are kept in either of two states. The high-density PCM (HDPCM) are pages in which the PCM cells are treated as MLCs, and the low-density low-latency PCM (LLPCM) are pages in which PCM is treated as SLC for reduced latency. Using a memory monitoring circuit, the MMS architecture is able to determine the portion of memory to be treated as HDPCM according to the capacity required by the varying workload. In order to correctly translate logical addresses to physical addresses and also provide current memory page allocations, the MMS collaborates with the OS which must be aware of pages in the LLPCM section of the memory, as these provide less memory capacity than is otherwise assumed by the software, which normally treats all memory addresses as belonging to HDPCM, thus providing greater capacity.

In order to estimate the memory capacity requirements of the system, a specialized memory monitoring circuit (MMON) is designed. This circuit estimates memory usage statistics using stack distance histogram analysis. The results of the MMON statistical algorithms are periodically read by the OS to determine subsequent handling of memory page allocations. Whenever the OS is informed of a lack of LLPCM pages, it evicts some HDPCM pages, enabling them to be used as LLPCM. Alternatively, LLPCM pages may be converted to HDPCM pages when they are evicted by caching write mechanisms or when the MMON detects that workload capacity requirements are on the rise, requiring increased PCM density.

7.1.7. Software Opportunities. Using PCM (or any other nonvolatile memory technology) in place of DRAM as main memory offers several conceptual advantages for software systems utilizing the nonvolatile property of the memory. While this is somewhat beyond the scope of PCM architecture, it does provide an extensive research field. Some recent key works are presented here.

Byte-Addressable Persistent File System (BPFS). Traditional file systems are designed for best performance with traditional block-accessed memory technologies. BPFS offers performance enhancements by utilizing byte-accessible nonvolatile memory technologies, such as PCM [Condit et al. 2009]. BPFS requires use of atomic write operations and write ordering mechanisms implemented in hardware. Using these capabilities, it allows for storing of simple data structures in persistent memory, while storing complex data structures in volatile memory. In order to ensure reliability, BPFS uses short-circuit shadow paging (SCSP). While most file systems use either logging or shadow paging (with shadow paging generally being less efficient), BPFS implements SCSP which allows for efficient commit operations to any location in the file system tree, due to memory byte-addressability and fast random write. The main drawback of BPFS resides in its need of dedicated hardware support and its nonstandard file system, which may require extensive adaptations of traditional computerized systems.

Nonvolatile Memory Heaps (NV-heaps). Architecting software for byte-addressable, nonvolatile memory systems has several advantages over traditional systems, but may also lead to some dangerous pitfalls. NV-heaps [Coburn et al. 2011] provide programmers with several primitives to ease work with nonvolatile memory while protecting them from common bugs. These primitives include objects, pointers, memory allocations, and atomic sections. They ensure pointer safety (e.g., protection against pointers in nonvolatile memory which point at locations in volatile memory), ACID transactions, traditional API (similar to volatile-memory based data structures), high performance, and scalability. Finally, NV-heaps provide the application with direct access to the nonvolatile memory, skipping operating system overheads.

Mnemosyne. The Mnemosyne architecture [Volos et al. 2011] provides programmers with an interface for direct access to nonvolatile memory, without any hardware constraints on PCM device design. Global data that should be stored in nonvolatile (persistent) memory is declared by programmers using the dedicated *pstatic* keyword. This direct nonvolatile memory access not only removes the need for data serialization into files in block-based file systems, but also does not require any special adaptation of the file system. Mnemosyne does not offers an alternative file system, but instead offer a fast mechanism for storing simple data which should be kept persistent. By providing user-mode access to persistent memory and offering the programmer mechanisms for consistent updates to the persistent memory, all the while requiring no hardware changes, Mnemosyne appears to be a feasible, simple-to-implement architecture, which is decoupled from PCM hardware architecural considerations.

7.2. Hybrid PCM and DRAM Memory

In most conservative main memory designs, the main memory homogeneously consists of a single type of memory, for example, DRAM or PCM. In order to overcome the challenges of implementing main memory using PCM while still making use of the advantages of DRAM, a hybrid design which consists of PCM-based main memory while using a small DRAM buffer is considered [Dhiman et al. 2009; Qureshi et al. 2009b; Ramos et al. 2011]. This allows for utilizing DRAM's superiority in terms of both latency and endurance to avoid those disadvantages in the PCM, while still building the majority of the main memory with PCM, with its superior density and scalability. This design is depicted in Figure 4(c). We also note that such hybrid designs may also benefit 3D die-stacking designs [Zhang and Li 2009], though these aspects are beyond the scope of this article.

Organization. Using PCM as the computer system's main memory is identical in its organization to traditional DRAM usage as main memory, with an operating-system aware page table. The addition in the hybrid main memory organization is a small DRAM buffer of which the OS is not aware [Dhiman et al. 2009; Qureshi et al. 2009b; Ramos et al. 2011]. As seen in Figure 9, it is managed internally in hardware by the main memory controller, similar to memory caches.

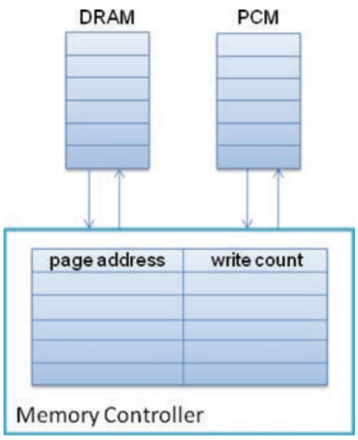

Fig. 9. A hybrid PCM and DRAM memory design with memory controller and write count statistics table.

Lazy-Write. The Lazy-Write organization technique [Qureshi et al. 2009b] uses the DRAM buffer to take advantage of the low latency of DRAM, compensating for the PCM main memory's inferior access and write speeds. Furthermore, it reduces the amount of actual write operations to the PCM memory, thus improving its limited lifetime.

The concept of this organization is that when a page fault occurs and a page must be brought into the main memory from the HDD, it is first inserted into the fast DRAM buffer instead of the slower PCM main memory. A page is allocated in both DRAM buffer and PCM memory, but the data is copied only to the buffer. Hence, read time from the HDD incurs relatively short DRAM latency instead of the longer PCM latency. Additionally, the memory controller holds two state bits for each page in the DRAM buffer—*present* and *dirty*. Pages which are marked with the *present* bit in the DRAM buffer are also present in the PCM main memory. This of course means that upon page fault, the page that is brought into the DRAM buffer has its *present* bit set to 0. When a page is replaced in the DRAM buffer, if it is dirty, it is written to the PCM memory first, to maintain consistency. This technique reduces redundant write operations to the endurance-wise limited PCM memory.

In addition to the DRAM buffer, a fast PCM write queue is used when actually writing to the PCM memory. This write queue, which is much faster than the PCM main memory, amortized the time spent for write operations. The DRAM buffer does not write directly to the PCM memory, and thus does not have to wait for the PCM's long latency. On the other hand, The write queue is large enough so that it does not get filled, and the written pages are transferred from it directly into the PCM, which incurs the long PCM latency cost, but since only the write queue is affected, the system as a whole is oblivious to this latency.

In short, the application of these two memory devices—the DRAM write buffer and the PCM write queue—in addition to the PCM main memory, helps reduce write operations to the PCM and compensate for its relatively high latency.

Line-Level Writes. It has already been suggested [Qureshi et al. 2009b] that the DRAM write buffer be used to write to the PCM main memory only pages which have been marked as *dirty*, that is, pages which have been written over since their last read from the main memory or HDD. In order to further improve upon this technique, the writing of the dirty pages into the PCM memory has to be considered on a finer granularity. This is done by considering chunks smaller than a whole memory page when deciding whether a certain datum is dirty and must be written back to the memory. The proposed granularity for consideration is the size of a cache line. This allows for only chunks which have been changed in the cache, and thus differ from their original values as stored in the main memory, to be written back. It incurs the greater cost of keeping track of each chunk's dirty state, instead of a unified dirty bit for the entire page, but can save a significant amount of redundant write-back operations which reduce PCM lifetime.

Fine-Grained Wear-Leveling in the Hybrid Architecture. Several techniques have been discussed here in order to reduce PCM wear in the DRAM and PCM hybrid memory organization [Dhiman et al. 2009; Qureshi et al. 2009b; Ramos et al. 2011]. While these techniques indeed decrease PCM write operations and help achieve the goal of increasing its lifetime, they are not complete without some wear-leveling mechanism. The granularity of the proposed wear-leveling mechanism is imposed by the line-level writes mechanism previously described. Analyzing memory write accesses inside a single page shows that the distribution of memory writes on a cache-line granularity inside a single page are nonuniform, and some cache lines are written more often than others. This nonuniform wear obviously reduces the memory's lifetime. Taking this uneven distribution into consideration, the idea here is to apply a cyclic shift on a cache-line scale, so that while some cache lines are logically written to more often than others, physically the wear is evened out over all cache lines in a page.

Like most wear-leveling mechanisms, it is imperative to hold some translation of logical addresses to physical addresses. In this proposed fine-grained wear-leveling mechanism, some register holds the size of the cyclic shift per memory page. For example, a 16-line memory page incurs a 4-bit register to hold its shifting value. Some random number generator is used to supply an initial value for the cyclic shift when the page is replaced. In this manner, each time the page is replaced, its shift register holds a varying evenly-distributed number, thus leveling the write wear across all cache lines in the page.

Page-Level Bypass. Memory usage profiling of various applications shows that some applications have poor memory reuse [Qureshi et al. 2009b]. An example of such applications is one which relies on streaming data. Such data are read once, used, and are then rendered irrelevant by the arrival of new data. As described earlier, memory pages read from the HDD are first stored in the DRAM buffer. When these pages are replaced by the new data, they are stored in the PCM. But the profiling shows that these pages are no longer needed by the application, and hence writing them to the PCM is not beneficial performance wise and only serves to wear the PCM cells, reducing its overall lifetime.

For such applications, it would prove beneficial to mark them for bypassing the PCM. Such a page-level bypass (PLB) flag would serve to dispose of memory pages replaced in the DRAM instead of writing them back to the PCM, saving redundant write operations. Such memory profiling is not done by the hardware itself—it can only be predefined, and the PLB set by the operating system.

Memory Controller and Page Manager. Hybrid PCM and DRAM architectures can also benefit from an OS memory handling policy, which is aware of the hybrid memory architecture [Dhiman et al. 2009]. Such a design incorporates architectural changes in both the memory controller (hardware) and the page manager (OS—software).

The memory controller routes memory access requests to the relevant hardware according to the requested address, whether it be in the DRAM section of the memory, or the PCM. It also keeps track of all PCM accesses on a page-level granularity (see Figure 9). This allows the controller, which is aware of PCM's limited endurance, to mark a PCM page as a "bad" page, once it has reached its theoretical limit for write accesses. This should prevent any future page allocations from reaching this hot page, while the rest of the PCM is still usable. Additionally, once the amount of memory writes to a single PCM page has reached some predetermined threshold, a page swap occurs, which is handled by the page manager and should transfer the hot virtual page to some other physical portion of the PCM, so as to provide wear leveling. These statistics, which are maintained by the memory controller, may be kept in a small dedicated SRAM cache so that this maintenance does not degrade PCM lifetime. Finally, since these statistics should be kept throughout the lifetime of the system, they are copied to the disk on shutdown and reloaded on startup, while also synchronizing the data with the disk periodically to account for system crashes.

The page manager is the software part of this design scheme, and with it, the OS ensures PCM wear leveling and an optimal use of the hybrid memory architecture. It consists of two key elements—the memory allocator and the page swapper. This design scheme assumes different behaviors for PCM page allocator and for DRAM page allocator. DRAM design is outside the scope of this work, and thus only the PCM allocator is discussed. The PCM allocator keeps four lists of memory pages: *free*, *used-free*, *threshold-free*, and *bad*. Initially, all pages are marked as free. After an allocated page has been freed, it is marked as used-free, to indicate that it is free for reallocations but has already been used recently, while other pages have not. Whenever the free list is exhausted, it is merged with the used-free. Also, as discussed regarding the memory controller, a page which has been written to more than a predetermined threshold amount of times is moved to the threshold-free list. This is done by the page manager by handling the page-swap generated by the memory controller, thus ensuring that this page will not be written to again until all memory pages have reached the threshold. This scheme provides an OS-based wear leveling for the PCM on a page-level granularity. The final list of bad pages contains pages which have reached their theoretical endurance limit and should never be used again, for fear of write or read failures.

When swapping pages, the page manager updates all relevant memory components to comply with memory coherency—page table entries, TLB entries, and the page contents themselves. It is also possible to choose from a simplistic memory swap policy, which always allocates pages from the PCM, or a hybrid policy. The simple policy has advantages of its own, as it provides some wear leveling for the PCM, independent of DRAM memory, and can be beneficial. However, the hybrid policy achieves the true goal of this design policy, as it provides a DRAM backup for the PCM endurance-limited cells. Such a policy not only reduces memory writes to the PCM but also allows heavily written memory pages to be placed in the DRAM, thus reducing the amount of page swap interrupts raised for these pages and removing a major bulk of PCM endurance costly writes.

8. SOLID-STATE DISK DESIGNS

8.1. Background

In recent years, advancements in solid-state disk (SSD) technology and nonvolatile memories (NVM) have led to NAND flash memory being used extensively in various applications, such as mobile devices, hard disk caches, and even HDD replacements [Sun et al. 2010]. While NAND flash is still a preferable choice for such applications, it has some limitations which need to be addressed in any future developments and applications of the technology.

One such shortcoming of NAND flash technology lies in the asymmetry between its read and write operations [Kim et al. 2008; Sun et al. 2010]. A NAND flash memory read request can access any of its pages directly. However, that is not the case for write

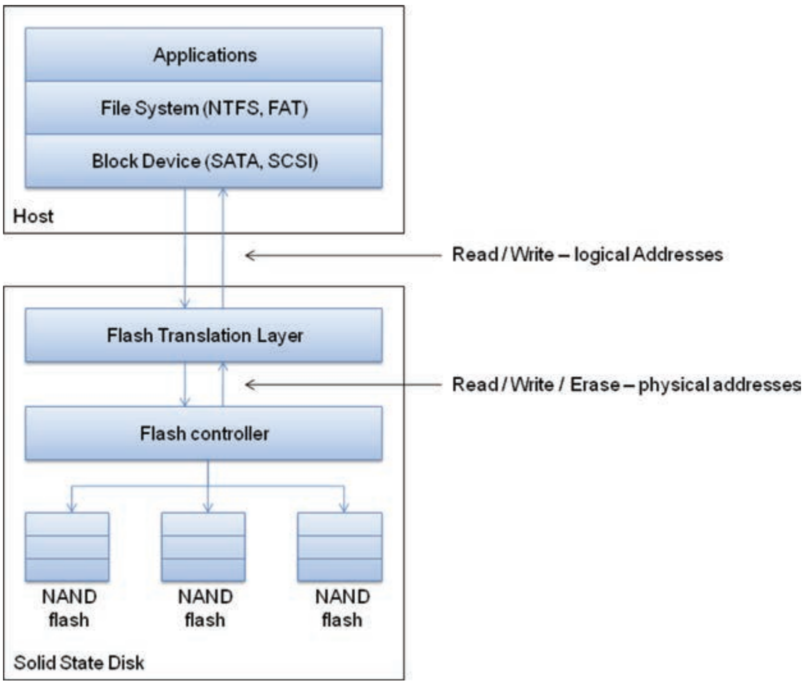

Fig. 10. Flash memory based SSD typical architecture with the flash tranlation layer.

operations. NAND flash pages cannot be written directly. Instead, before any write operation, the memory area being written has to be erased first. Furthermore, NAND flash cannot be erased on a single-page granularity. Instead, a single *erase unit* has to be erased. Such an erase unit typically consists of several adjacent pages, not all of which should be changed by the write operation. So, in effect, in order to write a NAND flash memory page, an entire erase unit has to be backed up, erased, and subsequently restored from the backup, with the updated version of the written page.

PCM has been shown to have superior endurance over NAND flash memory, but as it has not yet been manufactured on a large scale with a cost competitive to flash memory, it provides an opportunity for hybrid architectures which benefit from the advantages of both technologies [Boboila and Desnoyers 2010; Park et al. 2008; Yoon et al. 2008].

Flash Translation Layer. The most common solutions to the problem of erase-before-write in NAND flash memories have to do with using a flash translation layer (FTL) [Boboila and Desnoyers 2010; Park et al. 2006; Kim et al. 2008]. As seen in Figure 10, this abstraction layer translates between the file system and the physical flash pages, giving the flash device write operations the feel of regular HDD access operations. The extra layer redirects write requests by the file system to erase-free physical pages. Thus not all memory write requests result in immediate costly erase operations; instead, pages are written to physical memory areas which require no erase, while the original data page is marked as invalid. Of course, once all pages have been exhausted in this manner and there are no longer any erase-free memory pages, entire erase units have to be merged so that previously invalidated pages are erased, freeing up room for more write operations, while updated data from previous write operations are merged consistently. This method incurs a trade-off between faster write operations due to less frequent erase operations and the costly merge operations.

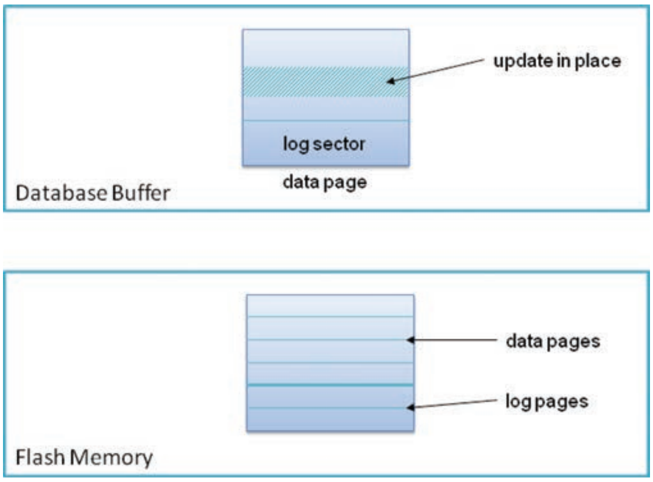

Fig. 11. IPL design.

It should be noted that the FTLs are mostly implemented as firmware and are proprietary of their respective manufacturers. One should therefore keep in mind that academic papers regarding FTL designs, some of which are surveyed here, may not necessarily be aware of the latest designs of industrial FTLs, as these designs are not publicly available.

FTL Mapping Schemes. FTLs can map logical addresses to physical addresses on varying resolutions [Park et al. 2006; Kim et al. 2008]. There are FTLs which translate on a page-level granularity, giving good performance for random access write patterns, but requiring a rather large amount of mapping information. Other FTLs map on a block-level granularity, requiring a significantly smaller amount of mapping information, but suffering a performance degradation owing to the extra operations required on the entire block, even when only a part of it has been written to. Finally, the log block mapping scheme combines both page-level mapping and block-level mapping by dividing each memory block into a data block and a log block. Block mapping is used for the data blocks and page mapping is used for the log blocks. Initially, every write operation is performed on the log block. Once it runs out of free space, the FTL merges both sections of the block. The size of the log block is fixed by design to avoid the large memory requirements of the page-level mapping.

8.2. PCM-Based Log Region

In-Page Logging. In-page logging (IPL) is a further improvement on the log block addressing method explained in the previous section [Lee and Moon 2007; Sun et al. 2010]. In IPL, instead of storing the entire updated page in the log section, the log section is divided into several log sectors. These log sectors are used to keep track of updated data only, and not the entire page. They can be allocated on demand and may also contain more than a single update record. These records are written back to the NAND flash memory when all log pages of a single erase unit are dirty or full, but the data page itself does not have to be written back, since all updates are stored in the log sectors. A schematic design can be seen in Figure 11. This method gives improved performance over traditional log block addressing, but still suffers performance degradation because no in-place updating can be done for the log pages, since that is still a limitation of NAND flash memory. Furthermore, a scenario of frequent writes to a single erase unit exhausts its log sectors quickly, triggering frequent costly merge operations between data and log sections.

PCM-Based IPL. In order to exploit the advantages of PCM over NAND flash memory, such as improved endurance and in-place updating, it is suggested to use hybrid architecture in which PCM is used for the log region [Sun et al. 2010]. This design utilizes a special log region controller which is responsible for address translation between the NAND flash data pages and PCM-based log pages.

A read operation accesses both memory types, where the data is first taken from the NAND flash data pages, and any further updates recorded for these pages are taken from the PCM log section, thus forming a complete, updated memory page. Write operations should first access the PCM log section. If no log records for the written page exist, then a new record should be allocated. If the page already has some log records associated with it, their contents are compared address-wise. Records which apply to the same memory address are overwritten (which could not have been done with NAND flash), and if no such records exist for the specific address, then a new record is allocated along with the other records for the same page. A merge operation is similar to previous schemes, where the log sector updates are applied to the data pages, while writing them to a different free erase unit.

Since the log section is smaller than the data section, the latency incurred by the peripheral access devices to it is smaller than the latency to access the data pages, and since these occur simultaneously, there is no extra latency incurred due to the usage of the hybrid memory scheme. Furthermore, PCM usage allows for in-place updating, which greatly reduces the amount of new log sectors allocated, and even though each write operation requires memory address comparison to check whether a log sector already exists, this overhead is dwarfed by the reduction in unnecessary writes, especially if keeping in mind that the PCM read operations are much cheaper than write operations. Finally, since PCM allows for read accesses with byte granularity, only the log records that are relevant to the read memory page are loaded, in contrast with NAND flash which requires loading an entire page or even several pages if the log records stretch out over several different memory pages.

Another aspect to consider if utilizing PCM for log pages is how to allocate the log region. Traditionally, IPL states that each erase unit has a fixed amount of log pages [Lee and Moon 2007]. This may be necessary when using the NAND flash itself for the log region, and its main advantage is in its simple design and easy address translation. However, in such a scheme, a merge operation occurs when the fixed size of the log region has been fully used for a given page, incurring a performance cost while other pages on the log region have yet to be utilized. This scheme is called a static log region assignment. In contrast, using PCM, it is possible to use the dynamic log region assignment [Sun et al. 2010]. In this scheme, the number of log sectors assigned for each erase unit is not fixed, but instead allocated based on the amount of updates for that particular unit. This allows for a significantly lower amount of merge operations, since more log sectors can be allocated for a frequently updated erase unit, which is particularly beneficial for asymmetric write profiles.

Finally, the hybrid architecture suggested improves the overall endurance of the memory, since most update operations do not affect the NAND flash, but only the PCM, which has much greater endurance. In order to optimize the endurance benefits, it is important to make sure PCM does not fail before the NAND flash, that is, the log region can be written to more often than the data region, but only by a factor determined by the ratio of PCM endurance to NAND flash endurance. Knowing this ratio beforehand, it can be set as a threshold to the amount of updates per page stored in the log region before a merge operation is forced, even if not all log pages are full. This way, the log pages and data pages wear out at the same pace. Further improvements for the log region lifetime can be obtained by utilizing some wear-level mechanisms, as previously elaborated in Section 7.1.2. It should be noted that while wear-leveling schemes are common in flash memory designs, such designs are not necessarily applicable for PCM. Since PCM is faster than flash memory in terms of both read and write accesses, its wear-leveling designs are under more limiting constraints, and it is less desirable to implement software-based solutions for wear-leveling. Thus PCM wear leveling designs may be different than flash memory designs.

8.3. Metadata Separation

Metadata and User Data. File systems generally keep track of user data and metadata. Every file updated incurs both a user data update and metadata update, which in NAND flash memory are both sent to the FTL, as previously described. While the size of metadata is usually much smaller than a file system block, it is nevertheless updated quite frequently, in about half of memory write operations [Kim et al. 2008; Park et al. 2008]. Additionally, analyses show that only a small portion of the metadata is usually changed upon a write operation. This incurs a rather expensive cost when writing to NAND flash memory, especially since the metadata and user data are not usually stored together in the file system, which means that such a write operation to NAND flash would access several memory blocks, breaking the sequential writes to the memory and increasing the amount of log records and merge operations.

FSMS and PFFS. File System Metadata Separation (FSMS) is a technique which separates metadata from user data [Kim et al. 2008]. The idea is to add a PCM device to store the metadata only. This requires a change in the block device driver to allow it to identify metadata writes according to their addresses. Once such a write has been identified, it is sent to a PCM filter device instead of to the NAND flash memory. This filter is responsible for carrying out only necessary write operations, using a read-before-write scheme.

Alternatively, the file system metadata itself could be altered to account for the hybrid architecture. This is done in the PCM-based Flash File System (PFFS) [Park et al. 2008]. The idea here is to maintain file system metadata, such as directory structure, in the PCM while enhancing the file system metadata to account for the specific structure of both NAND flash and PCM.

Hiding Address Translation. Demand-based Flash Translation Layer (DFTL) [Gupta et al. 2009] is a recent FTL design which is based entirely on page-level granularity address translation. It relies on the assumption that most workloads have temporal address locality, and therefore, there is no need to store the entire page mapping table for easy access, but instead, a caching mechanism is employed that allows quick address translation for hot pages. Non-cached address translations are stored in the flash itself, thus saving DRAM space.

Hiding address translation (HAT) is a hybrid memory architecture utilizing flash, a small portion of DRAM, and a PCM device [Park et al. 2006]. It is an upgraded FTL based on DFTL, but which stores the small cached address map on DRAM for quick access, while keeping the rest of the address translation map on a PCM device, utilizing its advantages (i.e., in-place updating, superior endurance), as previously detailed.

Chameleon. While not using PCM devices in itself, the Chameleon [Yoon et al. 2008] architecture is interesting in itself, since it is based on a hybrid of flash and FRAM technologies, which may also be applicable for PCM. This is another architecture which attempts to reduce the costly random access write operations on the NAND flash, since these are the most harmful to common FTL schemes' performance. This architecture separates any metadata from the actual written data and stores the metadata on the FRAM instead of on the flash device. The metadata mentioned here is not a file system generated metadata but, instead, the block mapping table of the FTL, pointers used by page-level write buffers, and so forth.

8.4. Hybrid Flash Translation Layer

Hybrid Flash Translation Layer (*h*FTL) [Kim et al. 2008] is a page-level granularity mapping based FTL, which incorporated PCM to store mapping information, thus reducing the main memory consumption due to mapping metadata. All metadata, including the mapping table, physical page status bitmaps, and the physical block information are kept on PCM, and only the data blocks are stored on the NAND flash memory.

The NAND flash memory blocks are also divided into data blocks, buffer blocks, and garbage blocks. Newly arrived data is always stored in the buffer blocks. Once a buffer block runs out of space, it is relabeled as a data block, and a new buffer block is allocated from the pool of garbage blocks. Should the amount of garbage blocks decline below a predetermined threshold, a merge operation occurs. As explained, all metadata is stored in the PCM. However, in order to account for slower PCM writes when compared with main memory, metadata which is related to the currently written buffer block, is temporarily stored in the main memory. Only when the buffer block is relabeled as data block is its metadata copied to the PCM.

In order to further reduce merge operations costs, *h*FTL implements the *logical-page-delete* function, which is called whenever the file system deletes a file. This function invalidates the physical pages which belong to the deleted file. The reason for this is that most file systems mark a file as deleted by changing only its metadata, while the user data itself remains seemingly valid, and may be redundantly copied during a merge operation. Using this function removed unneeded memory pages from costly merge operations.

A final improvement to merge-operation cost incorporated into *h*FTL is for the case when storage utilization is high and nearing 100%. In such a case, only a few garbage or free blocks are available for merge operations, rendering them less effective and more frequent. To account for this costly scenario, *h*FTL sets several reserve blocks aside, which are not counted in the total of logical space available, but are only used in such a scenario when more blocks are needed for merge operations.

9. CONCLUSIONS

This article surveyed the current state of PCM technology, its physics, the main challenges facing it as it attempts to provide a feasible alternative to DRAM main memory and to flash in SSDs, and some key architectural designs attempting to answer these challenges.

The field of PCM architectural research is constantly growing and provides attractive research opportunities. There are indications that existing omnipresent technologies, such as DRAM and flash, may soon reach the limit of their scalability, with their further development becoming non-cost-effective. In an attempt to be prepared for that day, the industry looks toward alternative memory-technology investing efforts and funding in designing such technologies. Among these, PCM is the most promising candidate to become the next commonly used memory technology.

PCM provides research opportunities because its advantages and limitations are different than those of DRAM and flash technologies. Therefore, while both flash and PCM suffer from limited endurance, the solutions for each of the technologies are different. That is also true for the technology's other limitations. However, PCM provides an opportunity for innovative design schemes that counteract these limitations by using some of PCM's unique advantages over other technologies, such as being accessible with higher granularity than flash, having faster read than write operations, and being nonvolatile, as opposed to DRAM.

Should PCM find its way to becoming a dominant main memory technology, it may also affect existing software architectures, which are based today on the fact that the main memory is always volatile. With the introduction of a nonvolatile memory design, software architectures may be less dependent on the storage system for saving durable metadata or the memory contents. Thus PCM-based designs may also trigger further research in software architectures and storage conceptions.

REFERENCES

Atwood, G. 2010. The evolution of phase-change memory. Micron's Innovations Blog, `http://www.micronblogs.com/2010/08/what's-the-future-for-pcm/`.

Bedeschi, F., Fackenthal, R., Resta, C., Donze, E., Jagasivamani, M., Buda, E., Pellizzer, F., Chow, D., Cabrini, A., Calvi, G., Faravelli, R., Fantini, A., Torelli, G., Mills, D., Gastaldi, R., and Casagrande, G. 2009. A bipolar-selected phase change memory featuring multi-level cell storage. *IEEE J. Solid-State Circuits 44,* 1, 217–227.

Bez, R., Bossi, S., Gleixner, B., Pellizzer, F., Pirovano, A., Servalli, G., and Tosi, M. 2010. Phase change memory development trends. In *Proceedings of the IEEE International Memory Workshop (IMW'10)*. 1–4.

Boboila, S. and Desnoyers, P. 2010. Write endurance in flash drives: Measurements and analysis. In *Proceedings of the 8th USENIX Conference on File and Storage Technologies* (FAST'10).

Burr, G. W., Breitwisch, M. J., Franceschini, M., Garetto, D., Gopalakrishnan, K., Jackson, B., Kurdi, B., Lam, C., Lastras, L. A., Padilla, A., Rajendran, B., Raoux, S., and Shenoy, R. S. 2010. Phase change memory technology. *J. Vacuum Sci. Technol. B: Microelectronics and Nanometer Structures 28,* 2, 223–262.

Cho, S. and Lee, H. 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)*. 347–357.

Coburn, J., Caulfield, A. M., Akel, A., Grupp, L. M., Gupta, R. K., Jhala, R., and Swanson, S. 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)*. 105–118.

Condit, J., Nightingale, E. B., Frost, C., Ipek, E., Lee, B., Burger, D., and Coetzee, D. 2009. Better I/O through byte-addressable, persistent memory. In *Proceedings of the ACM Symposium on Operating Systems Principles (SoSP'10)*.

Dhiman, G., Ayoub, R., and Rosing, T. 2009. PDRAM: A hybrid PRAM and DRAM main memory system. In *Proceedings of the 46th Annual Design Automation Conference (DAC'09)*. 664–469.

Dong, X. and Xie, Y. 2011. AdaMS: Adaptive MLC/SLC phase-change memory design for file storage. In *Proceedings of the 16th Asia and South Pacific Design Automation Conference (ASPDAC'11)*. 31–36.

Dong, X., Xie, Y., Muralimanohar, N., and Jouppi, N. P. 2011. Hybrid checkpointing using emerging nonvolatile memories for future exascale systems. *ACM Trans. Architect. Code Optim, 8*, 6:1–6:29.

Freitas, R. F. and Wilcke, W. W. 2008. Storage-class memory: The next storage system technology. *IBM J. Res. Develop. 52*, 439–447.

Guo, X., Ipek, E., and Soyata, T. 2010. Resistive computation: Avoiding the power wall with low-leakage, STT-MRAM based computing. In *Proceedings of the 37th Annual International Symposium on Computer Architecture (ISCA'10)*. 371–382.

Gupta, A., Kim, Y., and Urgaonkar, B. 2009. DFTL: A flash translation layer employing demand-based selective caching of page-level address mappings. In *Proceedings of the 14th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'09)*. 229–240.

Halderman, J. A., Schoen, S. D., Heninger, N., Clarkson, W., Paul, W., Calandrino, J. A., Feldman, A. J., Appelbaum, J., and Felten, E. W. 2008. Lest we remember: Cold boot attacks on encryption keys. In *Proceedings of the 17th Conference on Security Symposium*. 45–60.

Huai, Y. 2008. Spin-transfer torque MRAM (STT-MRAM) challenges and prospects. *AAPPS Bulle. 18,* 6, 33–40.

Ielmini, D., Lavizzari, S., Sharma, D., and Lacaita, A. 2007. Physical interpretation, modeling and impact on phase change memory (PCM) reliability of resistance drift due to chalcogenide structural relaxation. In *Proceedings of the IEEE International Electron Devices Meeting (IEDM'07)*.

Ipek, E., Condit, J., Nightingale, E. B., Burger, D., and Moscibroda, T. 2010. Dynamically replicated memory: Building reliable systems from nanoscale resistive memories. In *Proceedings of the 15th Edition of ASPLOS on Architectural Support for Programming Languages and Operating Systems (ASPLOS'10)*.

ITRS 2007. International Technology Roadmap for Semiconductors. Process integration, devices & structures. `http://www.itrs.net`.

Javanifard, J., Tanadi, T., Giduturi, H., Loe, K., Melcher, R., Khabiri, S., Hendrickson, N., Proescholdt, A., Ward, D., and Taylor, M. 2008. A 45nm self-aligned-contact process 1Gb NOR flash with 5MB/s program speed. In *Proceedings of the IEEE International Solid-State Circuits Conference. (ISSCC'08*. 424 –624.

Kim, J. K., Lee, H. G., Choi, S., and Bahng, K. I. 2008. A PRAM and NAND flash hybrid architecture for high-performance embedded storage subsystems. In *Proceedings of the 8th ACM international Conference on Embedded Software (EMSOFT'08)*. 31–40.

Kong, J. and Zhou, H. 2010. Improving privacy and lifetime of PCM-based main memory. In *Proceedings of the IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'10)*. 333–342.

Lai, S. 2003. Current status of the phase change memory and its future. In *Proceedings of the IEEE International Electron Devices Meeting (IEDM'03)*. 10.1.1–10.1.4.

Lam, C. 2007. Phase-change memory. In *Proceedings of the 65th Annual Device Research Conference*. 223–226.

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

Lee, S.-W. and Moon, B. 2007. Design of flash-based DBMS: An in-page logging approach. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD'07)*. 55–66.

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

Li, J. and Lam, C. 2011. Phase change memory. *Sci. China Inform. Sci. 54*, 1061–1072.

Lin, J.-T., Liao, Y.-B., Chiang, M.-H., Chiu, I.-H., Lin, C.-L., Hsu, W.-C., Chiang, P.-C., Sheu, S.-S., Hsu, Y.-Y., Liu, W.-H., Su, K.-L., Kao, M.-J., and Tsai, M.-J. 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)*. 525–528.

MicronFlash. 2008. Micron collaborates with Sun Microsystems to extend lifespan of flash-based storage, achieves one million write cycles. `http://news.micron.com/releases.cfm`.

MicronNumonyx. 2010. Numonyx introduces new phase change memory devices. `http://investors.micron.com/releasedetail.cfm?ReleaseID=466859`.

Nirschl, T., Phipp, J., Happ, T., Burr, G., Rajendran, B., Lee, M.-H., Schrott, A., Yang, M., Breitwisch, M., Chen, C.-F., Joseph, E., Lamorey, M., Cheek, R., Chen, S.-H., Zaidi, S., Raoux, S., Chen, Y., Zhu, Y., Bergmann, R., Lung, H.-L., and Lam, C. 2007. Write strategies for 2 and 4-bit multi-level phase-change memory. In *Proceedings of the IEEE International Electron Devices Meeting (IEDM'07)*. 461–464.

Nobunaga, D., Abedifard, E., Roohparvar, F., Lee, J., Yu, E., Vahidimowlavi, A., Abraham, M., Talreja, S., Sundaram, R., Rozman, R., Vu, L., Chen, C. L., Chandrasekhar, U., Bains, R., Viajedor, V., Mak, W., Choi, M., Udeshi, D., Luo, M., Qureshi, S., Tsai, J., Jaffin, F., Liu, Y., and Mancinelli, M. 2008. A 50nm 8Gb NAND flash memory with 100MB/s program throughput and 200MB/s DDR interface. In *Proceedings of the IEEE International Solid-State Circuits Conference (ISSCC'08)*. 426–625.

Numonyx. 2008. The basics of phase change memory (PCM) technology. Technical White Paper. `http://www.numonyx.com/Documents/WhitePapers/PCM_Basics_WP.pdf`.

NumonyxOmneo. 2010. Numonyx Omneo P8P PCM - 128-mbit parallel phase change memory (NP8P128A13BSM60E datasheet). `http://www.alldatasheet.com/datasheet-pdf/pdf/354464/NUMONYX/NP8P128A13BSM60E.html`.

Ohta, T. 2011. Phase change memory and breakthrough technologies. *IEEE Trans. Magnetics 47,* 3, 613–619.

Ovshinsky, S. R. 1968. Reversible electrical switching phenomena in disordered structures. *Physical Rev. Lett. 21,* 20, 1450–1453.

Park, C., Talawar, P., Won, D., Jung, M., Im, J., Kim, S., and Choi, Y. 2006. A high performance controller for NAND flash-based solid state disk (NSSD). In *Proceedings of the Non-Volatile Semiconductor Memory Workshop (IEEE NVSMW'06*. 17–20.

Park, Y., Lim, S.-H., Lee, C., and Park, K. H. 2008. PFFS: A scalable flash memory file system for the hybrid architecture of phase-change RAM and NAND flash. In *Proceedings of the ACM Symposium on Applied Computing (SAC'08)*. 1498–1503.

Pirovano, A., Lacaita, A., Pellizzer, F., Kostylev, S., Benvenuti, A., and Bez, R. 2004a. Low-field amorphous state resistance and threshold voltage drift in chalcogenide materials. *IEEE Trans. Electron Devices 51,* 5, 714–719.

Pirovano, A., Redaelli, A., Pellizzer, F., Ottogalli, F., Tosi, M., Ielmini, D., Lacaita, A., and Bez, R. 2004b. Reliability study of phase-change nonvolatile memories. *IEEE Trans. Device Materials Reliab. 4,* 3, 422–427.

Qureshi, M., Franceschini, M., and Lastras-Montano, L. 2010a. Improving read performance of phase change memories via write cancellation and write pausing. In *Proceedings of the IEEE 16th International Symposium on High Performance Computer Architecture (HPCA'10)*. 1–11.

Qureshi, M. K., Franceschini, M. M., Lastras-Montaño, L. A., and Karidis, J. P. 2010b. Morphable memory system: A robust architecture for exploiting multi-level phase change memories. In *Proceedings of the 37th Annual International Symposium on Computer Architecture (ISCA'10)*. 153–162.

Qureshi, M. K., Karidis, J., Franceschini, M., Srinivasan, V., Lastras, L., and Abali, B. 2009a. Enhancing lifetime and security of pcm-based main memory with start-gap wear leveling. In *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'09)*. 14–23.

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

Ramos, L. E., Gorbatov, E., and Bianchini, R. 2011. Page placement in hybrid memory systems. In *Proceedings of the International Conference on Supercomputing (ICS'11)*. 85–95.

Samsung. 2009. Samsung develops world's highest density DRAM chip (low-power 4Gb DDR3). `http://www.samsung.com/global/business/semiconductor/newsView.do?news_id=975`.

Samsung. 2010. Samsung ships industry's first multi-chip package with a PRAM chip for handsets. `http://www.samsung.com/us/news/newsRead.do?news_seq=18828`.

Samsung 2011. Samsung producing industry's highest density mobile DRAM, using 30nm-class technology. `http://www.samsung.com/global/business/semiconductor/newsView.do?news_id=1238`.

Schechter, S., Loh, G. H., Straus, K., and Burger, D. 2010. Use ECP, not ECC, for hard failures in resistive memories. In *Proceedings of the 37th Annual International Symposium on Computer Architecture (ISCA'10)*.

Seong, N. H., Woo, D. H., and Lee, H.-H. S. 2010a. Security refresh: Prevent malicious wear-out and increase durability for phase-change memory with dynamically randomized address mapping. In *Proceedings of the 37th Annual International Symposium on Computer Architecture (ISCA'10)*. 383–394.

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

Seznec, A. 2010. A phase change memory as a secure main memory. *IEEE Comput. Architect. Lett. 9*, 5–8.

Sun, G., Joo, Y., Chen, Y., Niu, D., Xie, Y., Chen, Y., and Li, H. 2010. A hybrid solid-state storage architecture for the performance, energy consumption, and lifetime improvement. In *Proceedings of the IEEE 16th International Symposium on High Performance Computer Architecture (HPCA'10)*. 1–12.

Thoziyoor, S., Ahn, J. H., Monchiero, M., Brockman, J. B., and Jouppi, N. P. 2008. A comprehensive memory modeling tool and its application to the design and analysis of future memory hierarchies. In *Proceedings of the 35th Annual International Symposium on Computer Architecture (ISCA'08)*. 51–62.

Toshiba. 2009. Toshiba announces world's first 512GB SSD laptop. `http://news.cnet.com/8301-17938_105-10241140-1.html`.

Volos, H., Tack, A. J., and Swift, M. M. 2011. Mnemosyne: Lightweight persistent memory. In *Proceedings of the 16th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'11)*. 91–104.

Wong, H., Raoux, S., Kim, S., Liang, J., Reifenberg, J., Rajendran, B., Asheghi, M., and Goodson, K. 2010. Phase change memory. *Proc. IEEE.*

Yang, B.-D., Lee, J.-E., Kim, J.-S., Cho, J., Lee, S.-Y., and Yu, B.-G. 2007. A low power phase-change random access memory using a data-comparison write scheme. In *Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS'07)*. 3014–3017.

Yoon, D. H., Muralimanohar, N., Chang, J., Ranganathan, P., Jouppi, N., and Erez, M. 2011. FREE-p: Protecting non-volatile memory against both hard and soft errors. In *Proceedings of the IEEE 17th International Symposium on High Performance Computer Architecture (HPCA'11)*. 466–477.

Yoon, J. H., Nam, E. H., Seong, Y. J., Kim, H., Kim, B., Min, S. L., and Cho, Y. 2008. Chameleon: A high performance flash/FRAM hybrid solid state disk architecture. *IEEE Comput. Architec. Lett. 7*, 17–20.

ZHANG, W. AND LI, T. 2009. Exploring phase change memory and 3d die-stacking for power/thermal friendly, fast and durable memory architectures. In *Proceedings of the 18th International Conference on Parallel Architectures and Compilation Techniques*. 101–112.

ZHOU, P., ZHAO, B., YANG, J., AND ZHANG, Y. 2009. A durable and energy efficient main memory using phase change memory technology. In *Proceedings of the 36th Annual International Symposium on Computer Architecture (ISCA'09)*. 14–23.

Received August 2011; revised November 2011; accepted January 2012