Hardware-Assisted Cooperative Integration of Wear-Leveling and Salvaging for Phase Change Memory

LEI JIANG, YU DU, BO ZHAO, YOUTAO ZHANG, BRUCE R. CHILDERS,
and JUN YANG, University of Pittsburgh

Phase Change Memory (PCM) has recently emerged as a promising memory technology. However, PCM's limited write endurance restricts its immediate use as a replacement for DRAM. To extend the lifetime of PCM chips, wear-leveling and salvaging techniques have been proposed. Wear-leveling balances write operations across different PCM regions while salvaging extends the duty cycle and provides graceful degradation for a nonnegligible number of failures.

Current wear-leveling and salvaging schemes have not been designed and integrated to work cooperatively to achieve the best PCM device lifetime. In particular, a noncontiguous PCM space generated from salvaging complicates wear-leveling and incurs large overhead. In this article, we propose LLS, a Line-Level mapping and Salvaging design. By allocating a dynamic portion of total space in a PCM device as backup space, and mapping failed lines to backup PCM, LLS constructs a contiguous PCM space and masks lower-level failures from the OS and applications. LLS integrates wear-leveling and salvaging and copes well with modern OSes. Our experimental results show that LLS achieves 31% longer lifetime than the state-of-the-art. It has negligible hardware cost and performance overhead.

Categories and Subject Descriptors: B.8 [Hardware]: Performance and Reliability; C.5 [Memory Structures]

General Terms: Design, Reliability, Performance

Additional Key Words and Phrases: Salvaging, wear-leveling, hard faults, phase change memory

ACM Reference Format:

Jiang, L., Du, Y., Zhao, B., Zhang, Y., Childers, B. R., and Yang, J. 2013. Hardware-Assisted cooperative integration of wear-leveling and salvaging for phase change memory. *ACM Trans. Archit. Code Optim.* 10, 2, Article 7 (May 2013), 25 pages.

DOI: <http://dx.doi.org/10.1145/2459316.2459318>

1. INTRODUCTION

As technology scales, the core count of modern Chip MultiProcessors (CMPs) is increasing fast (e.g., Nvidia's 480-core GTX-480 GPU [NVIDIA 2010] and Intel's 80-core TeraFlops [Intel 2009]). With more threads executing concurrently, there is an increasing demand for large main memory. Unfortunately, traditional charge-based DRAM, despite its wide use for over 30 years, now faces severe scalability and leakage

This work was supported in part by the National Science Foundation under grants CNS-CAREER 0747242, CNS-1012070, CCF-0811295, CCF-0811352, and CNS-0702236.

Authors' addresses: L. Jiang, Electrical and Computer Engineering Department, University of Pittsburgh, Pittsburgh, PA; email: lej16@pitt.edu; Y. Du, Department of Computer Science and Engineering, University of Pittsburgh, Pittsburgh, PA; email: fisherdu@cs.pitt.edu; B. Zhao, Electrical and Computer Engineering Department, University of Pittsburgh, Pittsburgh, PA; email: boz6@pitt.edu; Y. Zhang and B. R. Childers, Department of Computer Science and Engineering, University of Pittsburgh, Pittsburgh, PA; email: {zhangyt, childers}@cs.pitt.edu; J. Yang, Electrical and Computer Engineering Department, University of Pittsburgh, Pittsburgh, PA; email: juy9@pitt.edu.

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 1544-3566/2013/05-ART7 \$15.00

DOI: <http://dx.doi.org/10.1145/2459316.2459318>

problems. A recent ITRS report [ITRS 2009] indicated that there is no known path forward to scale DRAM below 22nm. To overcome this looming crisis, it is vital to exploit novel memory technologies to satisfy memory capacity requirements of future high-performance computing systems.

Phase Change Memory (PCM) has emerged as one of the most promising new memory technologies. A PCM cell consists of phase change material (e.g., $\text{Ge}_2\text{Sb}_2\text{Te}_5$ or GST) and its peripheral logic. While PCM has many advantages, such as scalability beyond 9nm [ITRS 2009], nonvolatility, and fast read access, it also has limitations. One major drawback is poor write endurance; a PCM cell can be reliably written only a restricted number of times. It has been reported that PCM chips can survive only 10^7 to 10^9 write cycles [Burr et al. 2010; ITRS 2009; Yeung et al. 2005]. This write endurance is significantly worse than DRAM, which promises at least 10^{15} write cycles. Without proper protection and wear-leveling, a PCM chip can fail in as little as several hours [Seong et al. 2010a]. Recent studies proposed several techniques to prolong the lifetime of PCM devices. The average number of writes to each PCM cell can be reduced through the DRAM write buffer [Lee et al. 2009; Qureshi et al. 2009b], differential-write [Zhou et al. 2009], and flip-N-write [Cho and Lee 2009]. And uneven writes to different PCM regions can be balanced with Start-Gap [Qureshi et al. 2009a], security refresh [Seong et al. 2010a], and table-driven segment swapping [Ferreira et al. 2010; Zhou et al. 2009]. Here the lifetime of a PCM chip is defined as the duty cycles until the appearance of the first failed cell (or the $(m+1)$ -th line failure when using a small m -entry spare line buffer [Qureshi et al. 2009a]).

When millions to billions of PCM cells are integrated into PCM arrays and chips, these cells show nonnegligible variations; some fail much earlier than others, even under the same write conditions. To mitigate the problem that the lifetime of a PCM chip is determined by weak cells and lines, several recent proposals studied how to salvage PCM chips after a significant number of cell failures (e.g., up to 50% of total cells). Ipek et al. proposed to pair-up two pages that have failed cells [Ipek et al. 2010]. A usable page is constructed with healthy cells from these two pages. Schechter et al. proposed to use Error Correcting Pointers (ECP) instead of traditional Error Correction Code (ECC) to replace failed cells in a memory line [Schechter et al. 2010]. Seong et al. proposed a mechanism to dynamically group broken bits to further extended PCM chip lifetime [Seong et al. 2010b]. Yoon et al. utilized fine-grained remapping pointers to correct memory line hard faults [Yoon et al. 2011]. These techniques degrade both performance and usable memory space.

The past work on wear-leveling and salvaging clearly demonstrates that *both* are necessary for PCM main memory. However, the simple integration of these approaches results in a noncontiguous PCM space [Ipek et al. 2010; Schechter et al. 2010; Seong et al. 2010b]. This characteristic is due to the “marking page retirement” mechanism adopted in salvaging. When failures are propagated from device level to the Operating System (OS), the OS marks corresponding pages as unavailable and retires them by not allocating them to the kernel or user applications in the future. This mechanism has two limitations. First, exposing device-level failures to the OS implicitly binds the mappings between Physical Address (PA) and PCM Device Address (DA). It complicates wear-leveling schemes that use randomized PA-DA mapping. It often requires frequent data movement at runtime. Second, a marked page may still contain a few healthy memory lines. Retiring the whole page misses the opportunity to exploit the remaining endurance of these lines. Among previous works, the only one that is close to our design is FREE-p [Yoon et al. 2011], a recent work that was developed in parallel with ours. To support hardware-assisted wear-leveling on contiguous memory space, FREE-p would require similar adaptation as ours at hardware and OS levels.

Fig. 1. PCM basics.

In this article we propose LLS (Line-Level mapping and Salvaging) to effectively integrate wear-leveling and salvaging. LLS divides the PCM space into a main PCM space and a backup space. Only the main PCM space is visible to the OS and user applications. Instead of marking failed lines and corresponding pages, LLS maps failed lines in main PCM to healthy lines in backup PCM such that a contiguous main PCM space is constructed. The size of the main PCM is reduced in proportion to the number of failed cells. We adopt intra-line salvaging (e.g., ECP or ECC). A line failure is encountered and handled by our scheme only if intra-line salvaging cannot correct all failed cells in a line. We describe low-cost LLS address translation hardware and illustrate how to integrate LLS with wear-leveling techniques that use randomized mappings. In addition to providing a contiguous PCM space, simulation results show that LLS extends the lifetime of a PCM chip by 24% on average.

To summarize, our contributions include the following.

- We identify the limitations when integrating existing line-level wear-leveling and salvaging techniques, and illustrate the importance of providing a contiguous memory space for PCM main memory.
- We propose *LLS*, a novel hardware design that smoothly integrates wear-leveling and salvaging. LLS transparently maps failed lines to backup lines and relieves the OS from managing failed pages at runtime.
- We describe a hardware design that effectively implements LLS with low cost. Our experimental results show that LLS not only provides contiguous memory space but also extends chip lifetime.

In the rest of the article, background material is described in Section 2. Our design is motivated in Section 3, while details about LLS are given in Section 4. The integration of our scheme with wear-leveling is presented in Section 5. Our schemes are evaluated in Section 6. Section 7 concludes the work.

2. BACKGROUND

2.1. Phase Change Memory (PCM)

Phase Change Memory (PCM) is an emerging type of nonvolatile memory. Figure 1(a) shows a PCM array example. A PCM cell utilizes a small portion of phase change material ($Ge_2Sb_2Te_5$, a.k.a GST) to represent a bit (“0” or “1”) with two reversible states that have a significant resistance difference.

As shown in Figure 1(b), there are two kinds of PCM write (or *programming*) operations [ITRS 2009]. A *SET* operation heats the phase change material, GST, above the crystallization temperature (300°C) but below the melting temperature (600°C) using a long but small current. A *SET* operation writes the cell into a logic “1” (crystalline state). In contrast, with a short and large current, the GST is melted and quenched

quickly by a *RESET* operation, which writes the cell into a logic “0” (amorphous state). Due to nondeterministic characteristics of GST, different cells, even the same cell at different times, respond differently to the same programming pulse [Cabrini et al. 2009; Qureshi et al. 2010; Mantegazza et al. 2008]. Therefore, PCM programming typically uses an iterative Program and Verify (P&V) write scheme. Particularly, writing a “1” is presented in Figure 1(c). Here, a *RESET* operation is always done first to put the cell in an initial state. A series of *SET* and verify (read) operations then follow until the target resistance level is reached. Iterative P&V can precisely control the written resistance in each PCM cell.

2.2. Errors in Phase Change Memory

Resistance drift and cross-talk are neglected in traditional Single-Level Cell (SLC) PCM as described in Burr et al. [2010]. For example, a SLC PCM cell, if it is written reliably, can retain data for more than 10 years at 85°C. As such, the PCM failures considered in this article are hard errors. Detecting a hard error is performed by read-after-write; each line write is followed by a line read to confirm if the data was correctly written. Recent work [Zhang and Li 2011] noticed that resistance drift may produce soft errors on Multiple-Level Cell (MLC) PCM. However, by applying antidrift encoding and changing MLC pages to single-level cell pages, the resistance drifting problem can be controlled [Zhang and Li 2011].

Unlike traditional SRAM, PCM is resilient to radiation-induced transient faults. However, because of the large area of periphery circuitry in a PCM chip, soft errors can still occur on the periphery, such as row buffers and interface logic. FREE-p [Yoon et al. 2011] deployed a 61-bit 6 bit-error correcting 7 bit-error detecting (6EC–7ED) BCH code for each 64-byte memory line to prevent potential soft errors. The device and circuit communities have a more straightforward way to eliminate soft errors in periphery circuitry through ECC or parity before buffering the data [Rodgers et al. 2008], exchanging parity bits during bus transmissions, and reissuing requests in the case of an error. Therefore, no additional storage overhead within the PCM data array is required for protection against soft errors.

To change state, a bit cell is heated and cooled by applying different currents (*RESET* and *SET* current). Due to repeated heating, a PCM cell can be reliably written only a limited number of times, which is referred to as *write endurance*. While an individual PCM cell can handle 10^{12} write cycles [Lai and Lowrey 2001], experiments with PCM arrays and chips have shown much lower endurance in the range of 10^7 – 10^9 writes [Burr et al. 2010; ITRS 2009; Yeung et al. 2005]. Write endurance is a significant obstacle that restricts PCM from serving as an immediate and widespread replacement for DRAM.

A PCM cell encounters hard faults, if the GST material cannot be reliably programmed between amorphous and crystalline states. Over the years, researchers have identified two major failure types: *stuck-reset* and *stuck-set* [Burr et al. 2010; Gleixner et al. 2009]. *Stuck-reset* happens when the GST material and the heater physically separate such that the PCM cell permanently loses its switching ability. The failure mechanism of *stuck-set* involves degradation and changes in internal material structure [Burr et al. 2010; Gleixner et al. 2009]. When the *RESET* current is not effective in creating an amorphous portion in the GST material, the cell sticks in the low-resistance *SET* state. By increasing the amplitude of the *RESET* current, the *stuck-set* cells can be overwritten into *RESET* state. However, within a small number of write cycles, these resurgent cells die in *stuck-reset* form [Burr et al. 2010; Gleixner et al. 2009]. Therefore, the *RESET* current is an important factor in determining PCM cell endurance [Kim and Ahn 2005].

Fig. 2. Conceptual view of wear-leveling, built-in replacement, and salvaging.

2.3. Process Variation

For PCM chips with billions of cells, some cells tend to fail earlier than others. One variation source is the difficulty in controlling physical feature size at nanoscale [Zhang and Li 2009]. Due to these variations, different cells have different *optimal RESET* current values. A cell suffers from *OVER-RESET* if a current higher than its optimal value is used. An early report showed that every $10\times$ increase in pulse energy results in $1000\times$ lower endurance [Kim and Ahn 2005; Lai 2003]. Recent measurements of failure rates on fabricated PCM chips showed similar results; $10\times$ more failures were observed when a cell is 60% overheated [Gleixner et al. 2009]. While strong systematic Process Variations (PV) might be mitigated through circuit design, for example, current provision [Jiang et al. 2011; Zhang and Li 2009] or customized write circuit [Lee et al. 2008], there are still nonnegligible variations at chip level.

To model PCM failures, we take the same approach as Ipek et al. [2010], Schechter et al. [2010], Seong et al. [2010b], and Yoon et al. [2011], which built PV and variance models with the help of Numonyx engineers (a manufacturer of PCM). These works adopted a normal distribution of cell failure with 10^8 nominal write cycles and cell-level variances in the range of 0.2 to 0.35. Their model was built to be a good match to industrial observations of significant random variations at chip level.

2.4. Wear-Leveling, Built-in Spare-Line Replacement, and Salvaging

Based on differences in how cell failures are handled, we divide current PCM endurance techniques into three categories: *wear-leveling*, *built-in spare-line replacement*, and *salvaging*. Figure 2 presents a conceptual view of the desired stages to apply these techniques and their impact to system-visible PCM space and access latency.

Wear-leveling aims to postpone the appearance of cell failures by spreading and balancing write operations [Qureshi et al. 2009a; Seong et al. 2010a; Zhou et al. 2009] among all usable cells/lines. Early table-driven wear-leveling techniques [Zhou et al. 2009] require OS management to periodically swap data stored in different regions based on write activity. To achieve better trade-offs, write frequencies are often recorded at a coarse granularity in the table. Recently proposed wear-leveling techniques build a randomized mapping between Physical Address (PA) and PCM Device Address (DA) [Qureshi et al. 2009a; Seong et al. 2010a]. In these designs, one PA may be mapped to different DAs at different times. The mapping is managed by simple hardware (including several registers and control circuit) and is hidden from the OS and user applications.

To accommodate relatively high cell failures, PCM devices include spare lines and use built-in hardware support to automatically remap failed lines to spares early in a chip's lifetime (i.e., with a small number of failures). Two types of hardware designs may be adopted. One design rewires the address decoding logic (similar to a large-capacity cache design [Ansari et al. 2009]) and the other uses a small remapping table. Both designs incur large hardware overhead, and thus can only support remapping a small number of failed lines. For example, Qureshi et al. [2009a] incorporate a

Fig. 3. Start-Gap wear-leveling on a memory containing 8 device lines.

spare-line buffer that can remap 5% of total lines. The benefits of built-in spare-line replacement are: it is transparent to upper-level designs, user-visible PCM space is contiguous, and access latency is little affected.

Salvaging techniques [Ipek et al. 2010; Schechter et al. 2010; Seong et al. 2010b; Yoon et al. 2011] try to extend the duty cycle of PCM chips that have a significant number of failed cells, for example, Ipek et al. [2010] can tolerate up to one half of all pages failing. Salvaging techniques gracefully degrade in accordance with the number of failed cells, which is a significant difference to built-in spare-line replacement that masks failures. To study the salvaging result in the later stage of the lifetime of the PCM chip, we adopt ECP [Schechter et al. 2010] as our salvaging baseline. Given a 512-bit (64B) line, ECP saves six 9-bit pointers and corresponding 1-bit data in extra storage that was traditionally used to hold ECC information. Each pointer can fix any failed cell within a 64B line. ECP significantly improves PCM lifetime over ECC and other error correction techniques. SAFER [Seong et al. 2010b] dynamically divides broken bits into different groups within one 64-byte memory line. SAFER achieves around 2% extra lifetime improvement compared to ECP. In this article, we also compare LLS with FREE-p [Yoon et al. 2011], a recent memory line-level salvaging scheme.

2.5. Memory Line-Level Wear-Leveling

2.5.1. Start-Gap. Start-Gap uses a simple linear formulation, instead of a large table, to evenly distribute write traffic across the entire device address space. Figure 3 is an example of Start-Gap. In Figure 3, the numbers on the left side are device addresses, while letters in the boxes are physical addresses. Start-Gap has two registers: one is *start*, which records the device address of the wear-leveling start point; the other is *gap*, which stores the position of a nonwritable spare line in the device. Write operations cannot happen on *gap*, so that each device line can have a nonwritable period by moving *gap*.

- Figure 3(a) is the initial state, where *gap* points to a spare line (device line 7) and *start* contains the device address of the physical line, A. No write request can reach device line 7 in Figure 3(a), since only device lines 0 to 6 are visible to the OS and device line 7 is invisible.
- *gap* is reduced by 1 (curve (1)) in Figure 3(b). Now, device line 6 becomes the spare line. And physical line G is mapped to device line 7. In Figure 3(c), when *start* and *gap* overlap, a gap round is finished.
- In the next gap round, as Figure 3(d) shows, *start* is increased by 1 and *gap* is decreased by 1 (mod the number of all device lines). The mappings between device address and physical address have been shuffled in one gap round. When *start* comes back to device line 0 again, a complete Start-Gap round is finished.

Fig. 4. Security Refresh wear-leveling on a memory containing 8 device lines.

To accelerate address randomization, Start-Gap can move *gap* by a random number, instead of 1, just like Figure 3(b) curve (2) shows. Feistel Network and Random Invertible Binary Matrix (RIB) are integrated into Start-Gap system to realize a random move on *gap*. Since a Feistel Network and RIB are static random address generators, a malicious attack may cause a PCM chip to fail with Start-Gap within several minutes.

Each gap move in Start-Gap [Qureshi et al. 2009a] is triggered by a threshold of cumulative write traffic. If this threshold is too high, the effect of wear-leveling cannot be significant. On the other hand, if this threshold is too low, frequent gap movement brings a lot of extra write operations into the PCM chip. Therefore, Start-Gap adopts a hierarchical wear-leveling design: Regional-Based Start-Gap (RBSG). In Figure 3(b), a PCM chip is divided into several memory regions, where Start-Gap does as regional wear-leveling. Another copy of Start-Gap with independent start and gap registers works across all the regions as a chip-level wear-leveling scheme. When a write operation happens in a region, both the chip-level Start-Gap and the regional Start-Gap move a gap step. Only with a small extra write overhead does the hierarchical design make write traffic balanced across the entire chip as fast as possible.

2.5.2. Security Refresh. The static remapping engine (e.g., Feistel Network or RIB) used in Start-Gap is vulnerable to the Birthday Paradox Attack (BPA) [Seong et al. 2010a]. To achieve higher reliability under malicious attack, Security Refresh was proposed to dynamically generate randomized addresses [Seong et al. 2010a]. Figure 4 illustrates an example of Security Refresh using a XOR operation to randomly and dynamically swap two lines in a memory block.

- Figure 4(a) shows the initial state. Physical Addresses (PA) are mapped to Device Addresses (DA) using a random key value key_0 ($K_0=4$). For example, PA 0 (0000) containing A is remapped to DA 4 (0100), that is, $0000 \oplus 0100 = 0100$. A *Current Refresh Pointer* (CRP) (the shaded box in the figure) records the position of the next line to be remapped. CRP increments after each Security Refresh step.
- In Figure 4(b), a new refresh round starts by using a different key value key_1 ($K_1 = 6$). PA 0 (0000) containing A needs to be swapped to DA 6 (0110) as $0000 \oplus 0110 = 0110$. The old data C in DA 6 is from PA 2. Given PA 2 (0010) should be swapped into DA 4, that is, $0010 \oplus 0110 = 0100$, the data in DA 4 and 6 are swapped in this step. CRP increments and points to PA 1. Taking another refresh step, the scheme swaps DA 5 and 7 in Figure 4(c).
- In Figure 4(d), CRP points to PA 2. At this time, PA 2 is in DA 4, which has been swapped in this round. By comparing $PA \oplus K_0 \oplus K_1$ and CRP, whether a PA has been swapped in the current round can be identified. Since $2 \oplus 4 \oplus 6 = 0$, and now CRP is 2, we know that PA 2 has been swapped and can be skipped. By applying the same rule, the scheme finishes one complete round in Figure 4(e).

Fig. 5. The 6 ECPs in one memory line.

Security Refresh can be performed hierarchically to accelerate the distribution of malicious write traffic [Seong et al. 2010a]. For the Start-Gap scheme, an online malicious application detection circuit [Qureshi et al. 2011b] was developed to identify and prevent runtime attacks by adjusting the wear-leveling frequency.

2.6. Salvaging Techniques

2.6.1. Dynamically Replicated Memory. Dynamically Replicated Memory (DRM) [Ipek et al. 2010] combines two broken pages into one healthy page to extend PCM service time. Within one memory line, DRM configures one parity bit to check for the appearance of a hard fault. When a broken page occurs in the PCM chip, the OS seeks a compatible page having no fault bits in the same positions as the broken page.

2.6.2. ECP. Due to natural immunity to soft errors, ECP [Schechter et al. 2010] replaces Hamming (72, 64) ECC Code on PCM. In Figure 5, an ECP entry consists of a 9-bit pointer field and 1-bit replacement cell. The pointer field records “2”, which is the position of the fail bit in the memory line. The replacement cell stores data of the hard fault bit. The storage overhead of 6 ECPs is 60 bits. Typically, there are 6 ECPs in one memory line. A 1-bit *FULL* field indicates whether these 6 ECPs are all used. The total storage overhead for one memory line is 61 bits.

2.6.3. FREE-p. To further extend PCM lifetime, FREE-p [Yoon et al. 2011] was developed to remap broken lines to healthy lines in other pages. Since the hard faulty bit in a broken memory line can still be read, the addresses of backup healthy lines are recorded in the broken lines. In one memory line, FREE-p utilizes 6-error correctable 7-error detectable (6EC7ED) BCH code to prevent both hard faults and soft errors. Four hard faults or fewer can be corrected by the BCH code with low overhead. When the number of hard faults in one memory line exceeds five, this memory line will be rewritten into the backup line address, so that the next memory traffic is redirected to the backup line. One bit is used for data/poiner flag.

FREE-p was developed to extend PCM chip lifetime. With adaptation of allocating backup lines in higher address space, it is possible to construct a contiguous PCM space. However, to integrate with recently proposed hardware-assisted wear-leveling schemes such as Start-Gap and Security Refresh, we still need hardware and OS support as discussed in Section 5.

3. THE MOTIVATION FOR A CONTIGUOUS PCM SPACE

As wear-leveling should be used throughout a PCM chip’s lifetime, it is important to achieve compatibility with both built-in spare-line replacement and salvaging. Wear-leveling and built-in spare-line replacement are compatible as the latter is a transparent hardware design. Unfortunately, current wear-leveling and salvaging techniques are not optimized to cooperatively work together.

Consider a naïve solution to combine wear-leveling and salvaging. A salvaging scheme, such as ECP, cannot mask all failures. When progressively more cells fail,

Fig. 6. Start-Gap prefers a contiguous PCM space (black block indicates the failed line).

ECP encounters uncorrected failures, and thus, it must mark the corresponding OS memory page as *nonusable*. ECP relies on the OS to retire the page from further allocation to either the kernel or user applications. Since the OS sees Physical Addresses (PA) while the hardware sees Device Addresses (DA), this simple marking scheme implicitly binds a failed DA with an OS-marked PA.

Figure 6 illustrates the problem of combining Start-Gap and ECP. Start-Gap does randomized address mapping in two steps. In step 1, PAs are randomized with a pseudorandom function, such as random invertible binary matrix (RIB) [Qureshi et al. 2009a]. In step 2, the Randomized PAs (RPAs) are mapped to DAs based on the current start and gap locations. Step 1 uses static mapping while step 2 changes mapping at runtime. In Figure 6, we assume, in step 1, PA-100 and PA-500 are randomized to RPA-1 and RPA-(N-1) where N is the maximum size of the PCM memory. In step 2, RPA-1 and RPA-(N-1) are mapped to DA-1 at time 1 and 2, respectively. That is, at time 1, PA-100 maps to RPA-1 and then to DA-1; at time 2, PA-500 maps to RPA-(N-1) and then to DA-1.

If DA-1 fails at runtime (solid black block), then the OS needs a DA/PA mapping table to expose the DA failure to the PA level. Since PA/DA mapping depends on start and gap locations, it requires frequent updates to ensure correctness. For example, from time 1 to 2, the mapped device address of PA-500 changes from *usable* to *nonusable*, that is, from DA-(N-1) to DA-1. If PA-500 belongs to a user application, then user data cannot be stored in failed DA-1 at time 2. The OS needs to explicitly relocate data in PA-500 to another physical address, say, PA-600. The complication is Start-Gap prefers a contiguous memory space such that PA-DA mappings can be freely built and changed at runtime without the costly involvement of the OS. Start-Gap works well with ECP when all in-line errors are masked and no page is marked as nonusable.

Next, we show that integrating Security Refresh [Seong et al. 2010a] and salvaging has similar problems. Given a physical address for a line, Security Refresh performs XOR operations to determine the actual device address for the line to be accessed. To dynamically change the mapping keys, Security Refresh utilizes XOR properties such that each address has a unique peer address. By maintaining a refresh pointer, Security Refresh amortizes cost by swapping at most one pair during each invocation (for one-level Security Refresh). Intrinsically, an XOR operation prefers a contiguous 2^n -sized usable space. This approach is problematic with marked failed lines since a usable line may be mapped to a usable line with one key, but to a failed line with a different key. It also incurs a large cost to maintain mappings, similar to combining Start-Gap and salvaging as discussed earlier.

We compare our scheme to a strawman solution that extends current salvaging schemes with a hardware-managed mapping table, similar to the one used in built-in

Fig. 7. Splitting the PCM memory to main and backup space (to achieve graceful degradation, each chunk contains data from all banks).

spare-line replacement. For a 8GB PCM space, a 21-bit table entry is required for each 4KB page if remapping is done at page level. A 27-bit entry is required for each 64B line if remapping is done at line level. Due to space constraints, suppose the table is created at page level. Whenever a 4KB-sized page A fails, the hardware maps it to the last healthy page B in the whole space. Page B, instead of page A, is marked as *nonusable*. Future accesses to page A are redirected to page B with the mapping table. In this way, a contiguous usable PCM space is created in the lower address space while contiguous higher addresses are marked as nonusable. The difference between this solution and LLS is that LLS does remapping at line level, which helps to achieve better lifetime as shown later in our experiments.

4. LLS: LINE-LEVEL MAPPING AND SALVAGING

4.1. Overview

An overview of our Line-Level mapping and Salvaging scheme (LLS) is shown in Figure 7. The whole PCM device space M is divided into 2^s chunks. A contiguous subset of these chunks, starting from address 0, are used as the main PCM. The rest of memory space is the backup PCM. Each chunk consists of PCM arrays from all banks, and thus, allocating a subset of chunks to backup PCM does not bias PCM accesses to certain banks. Only main PCM is visible to the OS. The backup PCM is not visible. Initially the whole visible address space is in main PCM. Although no chunk is in backup PCM at this stage, a small number of cell failures can be corrected by line-level (64B) ECP, or built-in spare-line replacement hardware.

Eventually, with enough write cycles, a PCM line may fail as it has more failed cells than the cells that can be rescued by ECP. The system then activates LLS which moves the last chunk from the main space to backup space such that failed lines are marked and remapped to healthy lines in backup PCM. Future accesses to failed lines will be automatically redirected by hardware to the mapped lines. The failure details (e.g., the total number and the exact locations of failed lines) are hidden from the OS and user applications. LLS implicitly binds backup PCM to a higher address space at device address level. However, as we show next, LLS does not restrict any particular PA-DA address mapping. Instead, it supports randomized PA-DA mappings at runtime.

As more lines fail, the backup PCM space will eventually become insufficient. LLS then dynamically resizes main PCM and moves up to half of all chunks to backup PCM. In this way, the main PCM is resized in a step-down fashion to accommodate more failed lines. At any given time, the OS and user applications can only see and access a contiguous physical address space whose size is equal to main PCM's.

Fig. 8. A salvaging group contains 2^{s+t} lines (i.e., 2^t line from each of 2^s chunks).

4.2. Mapping Failed Lines to the Backup PCM

Figure 8 illustrates the mapping to smoothly resize PCM memory. LLS first constructs a global bitmap using one bit per line to indicate line status: “0” represents a healthy line and “1” represents a broken line. If a broken line is in main PCM, it will be remapped. The bitmap uses SRAM structure and is protected by (72,64) ECC. Its failure rate of bitmap is lowered to an ignorable level [Yoon and Erez 2009]. We show the details in Section 6.3.1. If a broken line is in backup PCM, it cannot be used to rescue other failed lines. The bitmap is organized as a two-dimensional array. One row (Figure 8) in the bitmap records the status of a *salvaging group* that is constructed as follows.

Suppose the PCM space is divided into 2^s chunks. We choose consecutive 2^t lines from each chunk such that the bit vector for a salvaging group has $2^{(s+t)}$ bits. As an example, if we divide a 8GB PCM into 128 chunks, and choose 4 lines from each chunk, then there are $128 \times 4 = 512$ lines in a salvaging group. In total, we have $8GB/64B/(128 \times 4) = 256K$ groups. Given a bit vector for a salvaging group, all its bits belong to main PCM when the system was first built. As the PCM is resized, the bit vector is split. Each PCM resizing moves 2^t bits to backup PCM. The offset is recorded in a *space split register* R_{loc} . In this example, initially R_{loc} is 512. After moving one chunk to backup PCM, R_{loc} is 508 (=512-4).

LLS adopts *in-group* sequential mapping to map failed lines to backup PCM. That is, the first broken main PCM line is mapped to the first healthy backup line, the second broken main PCM line is mapped to the second healthy backup line; and so on. In main PCM space, we count from low to high address. In backup space, we count in the reverse direction, that is, from higher to lower addresses. We use the reverse direction in backup PCM to avoid data movement during PCM resizing. Since there are more lines in main PCM than in backup PCM, and there might be failed lines in backup PCM, the address mapping involves two subtasks. Given a broken line X , LLS will do the following.

- (1) *Determine the broken line rank Y in X 's salvaging group.* The broken line rank Y is 1 plus the number of preceding broken lines in X 's salvaging group. Y means that the broken line should be mapped to the Y -th healthy line in backup PCM of the salvaging group.
- (2) *Determine the mapped address W for X in backup PCM.* With the computed rank Y , if there is no broken line in backup PCM, then the Y -th healthy line is the W -th line in backup PCM. If there are broken lines between 0 and Y , then the mapped line might

Fig. 9. Address mapping in three steps.

be different. In Figure 8, the 2nd broken line is mapped to the 3rd line (i.e., the 2nd healthy line) in backup PCM. Note we count in reverse order in backup PCM.

In LLS, each line stores a status bit in the PCM array. The line status information is distributed in both the line cell array and the centralized bitmap. This redundancy removes bitmap access from the critical path. When the memory controller gets a Device Address (DA), it is sent directly to the PCM bank if it is the next access to be scheduled. This speculation introduces no performance penalty for accesses to healthy lines. If the line is broken, then the bank access fails based on the line status bit in the PCM array, which enables the bitmap access to compute the mapped address. The second PCM access is then sent to access the (healthy) mapped line. Due to speculation penalty, LLS enables speculation at early stages when most lines are healthy and disables speculation when more than 30% of chunks have been moved to the backup PCM. When speculation is disabled, the bitmap is accessed before all bank accesses.

Hardware-assisted address translation is designed to solve the following problem: *given a broken line X in main PCM, how to quickly identify its mapped line W in the same group in backup PCM?* Since address translation is always performed in one group, in the following description, we use group offset to indicate a line. The task is find Y such that, in the corresponding group, X is the Y -th broken line in the main PCM while W is the Y -th healthy line in backup PCM.

We show the address translation logic in Figure 9 and Figure 10. We divide the PCM space into 128 chunks and choose 4 lines from each chunk. The PCM fails if more than half of the space moves into the backup PCM. Therefore, each salvaging group has $128 \times 4 = 512$ bits, and at most, the last 256 bits belong to the backup space. In the discussion, we assume five chunks are currently in the backup space.

Figure 9 shows the translation steps. Given a broken line $X=465$ (group offset), do the following.

- Step 1: Compute X 's broken line rank Y by counting the preceding broken lines. Assume $Y=15$.
- Step 2: Split the bits from backup space into 16-bit subgroups and identify which subgroup holds the line. Here, we have $Z=30$ indicating the mapped backup line is in the 2nd subgroup in reverse order.
- Step 3: Return the location after identifying the backup line in a 16-bit subgroup. $W=494$ is referred in this example.

We need three three bit-operations to assist address translation. As shown in Figure 10(a), one operation is about mapping from a PCM device address to its salvaging group (group id, offset). The other two operations generate two 512-bit

Fig. 10. The details of address translation logic.

bit-masks: A-mask and B-mask. Given a group offset $X=465$, bits 0 to 465 of A-mask are set, indicating that the broken line rank only counts 1s of these positions. B-mask is generated from R_{loc} to differentiate the bits in main PCM and backup spaces. Given $R_{loc}=492$ (i.e., 5 chunks in backup PCM), we set the last $5 \times 4 = 20$ bits of the B-mask.

Given an address X in the first step (Figure 10(b)), we fetch X 's bit vector for its salvaging group and filter the result with A-mask. We then divide 512 bits into 32 16-bit subgroups. A 32-way parallel 16-bit population counter is used to count the number of 1s in each subgroup. We use the fast population counting logic proposed in Ramanarayanan et al. [2008]. Ramanarayanan et al. used a 3:2 compressor and Wallace-tree structure to implement a 64-bit population counter in a single cycle on a 2.1 GHz low-power CPU using 65nm technology [Ramanarayanan et al. 2008]. The 32 5-bit results are again summarized using 3:2 compressor and Wallace-tree structure to get the broken line rank Y .

Figure 10(c) shows the second step. We apply an inverter to the bottom half of the bit vector such that healthy line bits are 1s, and select the subgroups to count using

B-mask. We reuse the population counting logic to count the number of healthy lines in each subgroup. A 4-stage circuit is used to calculate the prefix sum [Beaumont-Smith and Lim 2001], which indicates the range of healthy lines that will be covered by each subgroup. We subtract these prefix sums from Y and select the minimum positive residue from the results. In this step, we also produce a 16-bit group index. Step 2 can be skipped if backup PCM contains less than 5 chunks—only one group exists.

Figure 10(d) shows the third step. To identify the backup line in a 16-bit range, we compute the prefix sum inside a group. We generate 16 16-bit fixed masks. Each mask has a different length and each 16-bit subgroup uses one mask. The 1's in each masked 16-bit vector are counted. The first one matching the residue indicates the desired location within the subgroup. The final address W is a combination of subgroup index from step 2 and in-group offset from step 3.

4.3. Dynamic PCM Resizing

In this section, we describe when and how to perform PCM resizing, (i.e., moving chunks to backup PCM). In LLS, PCM resizing is triggered by a write operation, either to main PCM or to backup PCM. When a write operation fails to save data in a line and the failure cannot be rescued by ECP, then the line status bit (in the global bitmap) is set. When there are more broken lines in main PCM than healthy lines in backup PCM, PCM resizing is triggered.

In most cases, PCM resizing is triggered by write operations to the main PCM. Assume one salvaging group already has 9 broken lines in main PCM and a new write operation fails in another main PCM line. If the newly broken line takes the rank 6, then broken lines with old ranks 6, 7, and 8 need to change their ranks to 7, 8, and 9, respectively. The change requires line shifting in the backup space such that the new broken line can take the 7th healthy line in backup space. As there are now 10 broken lines, there may not be enough healthy lines in backup PCM. An exception is raised in this situation to pause the system to resize the PCM.

In the other cases, writes to backup PCM may also fail. If there are still healthy lines left in backup PCM, then only the affected lines are shifted. Otherwise, the failed writes will trigger PCM resizing. These writes include: (1) writes to a broken line in main PCM (this write is redirected to backup PCM); (2) writes generated from maintenance such as line shifting in backup PCM.

To perform PCM resizing, we need to consider its impact at the Physical Address (PA) and Device Address (DA) levels. At PA level, enforced by the space split register R_{loc} , the OS and user applications cannot access any physical address beyond the main PCM boundary after resizing. For this reason, any data allocated in the affected (moved) chunk needs to be reallocated to other locations. In the worst-case, the OS needs to move a full chunk with data. For example, given a 8GB PCM divided into 128 chunks, the worst-case data movement is 64MB (the chunk size). While it is relatively expensive, PCM resizing is done rarely relative to device lifetime. If a PCM chip is discarded after 50% of its capacity has failed, then LLS does only 64 resize operations during the chip's lifetime.

At DA level, the space split register R_{loc} identifies which bits can be used to salvage failed main PCM lines. Thus, the corresponding device space should not contain any useful data. If a direct map is used between PA and DA, then it is straightforward; reallocating OS pages moves useful data out of the affected chunk at DA level automatically. However, when wear-leveling with randomized address mapping is adopted, the PA-DA mapping is randomized such that the affected chunk may contain useful data that cannot be expunged by physical address reallocation. Reallocating OS pages cannot clean up this chunk because the mapping is invisible to the OS. Therefore, a

Fig. 11. Integrating Start-Gap and LLS with the support for PCM resizing.

way to ensure correctness is necessary. We describe the details of how LLS ensures correct operation in the next section.

5. INTEGRATION OF WEAR-LEVELING AND SALVAGING

By providing a contiguous PCM space, LLS hides lower-level line failures from the OS and user applications. When salvaging (LLS) is integrated with wear-leveling that is based on randomized address mappings, the only support that LLS needs to provide is a one-to-one PA-DA mapping over the contiguous space. The mapping should maintain consistency before and after a PCM resizing. We next describe how LLS supports Start-Gap.

5.1. Integrating LLS and Start-Gap

To defend against repeated address attacks, a variation of baseline Start-Gap, called Region-Based Start-Gap (RBSG), was proposed. RBSG divides the whole PCM into 64MB or smaller regions in the second step of baseline Start-Gap, and does wear-leveling in each region independently. Address randomization is performed in the first step. RBSG is still vulnerable to specially designed attacks, such as the birthday paradox attack [Seong et al. 2010a; Seznec 2010]. Qureshi et al. [2009a] later proposed enhancements to defend against such attacks [Qureshi et al. 2011a].

To integrate with RBSG, LLS slightly modifies address randomization in RBSG's first step. The PCM space is divided into two halves, the first half of PA is randomized to the second half of RPA, and vice versa (Figure 11(a)). A chunk in LLS is equal to or larger than a region in RBSG. At physical address level, a chunk is failure free such that RBSG can be performed without any modification. If the chunk is smaller than 64MB, for example, a 32MB chunk size results after dividing 4GB into 128 chunks. Performing RBSG on each 32MB region slightly increases overhead: it doubles the number of start/gap registers and control logic. However, this overhead is very low [Qureshi et al. 2009a].

When there is a need to resize PCM, LLS pauses execution and notifies the OS with an interrupt. After moving one chunk to the backup PCM, the system cleans up the data in the affected chunk at both physical and device levels. At physical address level, the OS needs to reallocate pages in the affected chunk to other locations as discussed in Section 4.

At device level, LLS needs to move useful lines in the affected chunk to new locations. Figure 11(b) illustrates that the new locations can be safely identified by applying another round of RIB mapping. Given a DA address DA-7800, the reverse mapping can be used to find the RPA address RPA-7200 and DA address DA-100. For PA-7200,

Fig. 12. Integrating LLS and Security Refresh with a new mapping.

its RPA address can be identified as RPA-1100. Since PA-7200 will not be used (due to reduced main PCM space), RPA-1100 and its associated DA address DA-1500 are the ideal new locations to hold the user data in DA-7800. We use DA-1500 as otherwise it is an unused line (due to PA-7200 no longer being used).

Given a PA address after resizing, if its RPA address is within the main PCM space, the original RBSG mapping is used to access the DA address in the corresponding region. If its RPA is beyond the main PCM space, we perform another round of randomization, that is, $RIB(RIB(PA-100)) = RPA-1100$, to find the mapped RPA address. We need at most two rounds of randomization due to the half-to-half mapping. In this way, LLS integrates LLS and Start-Gap.

5.2. Integrating LLS and Security Refresh

Integrating LLS with Security Refresh is more involved. Because the outer-level XOR operation performed in Security Refresh maps a line address in one chunk globally to an address in another chunk, two issues must be addressed in LLS: (1) The XOR operation used in Security Refresh intrinsically works on a 2^n -sized space. However, the PCM size after a sequence of PCM resizings usually does not satisfy this requirement. (2) When LLS moves a chunk to the backup PCM, due to the global mapping mechanism, the chunk at DA level may contain useful data from other main PCM chunks.

To support Security Refresh, LLS requires an enhanced mapping scheme as shown in Figure 12(a). Given an 8GB PCM space divided into 128 chunks, the chunk offset includes bit 0 to bit 25 and the chunk index includes bit 26 to bit 32. In the new mapping scheme, bit 6 to bit 24 (excluding bank ID and rank ID) are XOR-ed with a random key in the same way as in the original Security Refresh scheme. This helps to prevent inter-bank communication and achieves the same swapping overhead as Security Refresh. Bit 25 to bit 32 are remapped using a small 256-entry table. Each entry in this table corresponds to a Half-Chunk (HC) space while each chunk is 64MB, that is, the default size to perform RBSG. As an example, value “8” in entry 5 indicates “mapping PA HC-5 to DA HC-8”, that is, physical addresses in the second half of the 3rd chunk (or PA HC-5) are mapped to device addresses of the first half of the 5th chunk (or DA HC-8). Initially, all 256 entries in the table are active when the entire space is used as the main PCM. As chunks are gradually removed from the main PCM, a smaller number of entries are active. In particular, removing one chunk disables two active entries in the table.

To construct a new mapping, we: (1) choose a new random key for chunk offset bits 6 to 24 (excluding bits 14 to 17); and (2) determine a new mapping table for bits 25 to 32. In the following discussion, we focus only on the half-chunk bits (bits 25 to 32). The offset bits are XOR-ed in the same way as in the original proposal.

Fig. 13. A table-driven mapping to enable progressive refresh. PA-HC-2 refers to physical addresses whose half-chunk index (bit 25 to bit 32) is 2; DA-HC-2 refers to device addresses; A mapping table entry, for example, 0→4 indicates lines in PA-HC-0 are mapped to lines in DA-HC-4.

Security Refresh employs an important property of XOR to enable progressive refresh. Given an old key K_1 and a new key K_2 , an address X and its peer address $X \oplus K_1 \oplus K_2$ form a pair such that in order to switch the mapping from using the old key to using the new key, only the contents in X and $X \oplus K_1 \oplus K_2$ are swapped. This feature enables progressive refresh such that the key update overhead can be amortized. Unfortunately, this valuable property can be lost if a mapping table is generated without restriction. As shown in Figure 12(b), assume the old table mapping is $M_1(x) = (x+3) \bmod 255$ and the new mapping is $M_2(x) = (x+4) \bmod 255$, and we want to relocate a physical line from half-chunk 0, that is, PA-HC-0. The line currently stays in DA-HC-3 and needs to be moved to DA-HC-4. Since DA-HC-4 is occupied by PA-HC-1, we need to move PA-HC-1 first. Similarly, before moving PA-HC-1, we need to move PA-HC-2, and so on. These moves are chained which causes the refresh operation to be done in one step; no progressive refresh is possible.

Instead, a mapping table is constructed that supports progressive refresh as follows. First, all active entries are randomly paired in the old table (using a new randomization key, not the one for Security Refresh) and, second, the stored contents of each pair are swapped. Pairing can always be done since we have an even number of active entries; moving one chunk to backup space deactivates two entries in the table. To assist in finding the peer, the swap is recorded in a peer table. For example, in Figure 13(a), we swap entry 0 and 5 such that PA-HC-0 and PA-HC-5 are peers. Each red arrow in Figure 13(a) links the two peers in one pair. Note DA-HC-0 and DA-HC-5 are not peers.

Progressive refresh can be done at runtime by swapping only peers. Since we convert key-based refresh to table-based refresh, all these tables (the old and new mapping tables, and the peer table) are needed to enable finding the peers at runtime, in particular, before and after resizing. In Figure 13(b), assume we need to increment refresh pointer from PA-HC-0 to PA-HC-1, and thus, need to swap lines in PA-HC-0 to their destination locations based on the peer table. Since PA-HC-0 is mapped to DA-HC-4 and DA-HC-1 based on the old and new mappings, respectively, the action taken is to swap lines in these two chunks (i.e., each refresh swaps two lines) as shown in

Fig. 14. Resizing cases 1 to ensure consistency at DA level. Here, C1/B2 indicates that the half-chunk contains partial contents of both C1 and B2.

Fig. 15. Resizing cases 2 to ensure consistency at DA level. Here, E1/X2 indicates that the half-chunk contains partial contents of both E1 and X2.

Figure 13(c). Consider what may happen at this stage if there is a memory access. Given a physical address PA-HC-0, it needs to use the new mapping as 0 is smaller than the refresh pointer (=1 at this time) (Figure 13(f)(i)). For the second access PA-HC-2, 2 is bigger than the refresh pointer, as shown in Figure 13(d). Since its peer is PA-HC-1, and 1 is not smaller than the refresh pointer, PA-HC-2 still uses the old mapping (Figure 13(f)(ii)). For the third access PA-HC-5 as shown in Figure 13(e), we use the new mapping since its peer (PA-HC-0) has “0 < the refresh pointer” (Figure 13(f)(iii)).

PCM resizing during Security Refresh. PCM resizing may be triggered in the middle of refresh, that is, changing from an old mapping to a new mapping. Figure 14 illustrates the actions that are done to ensure mapping consistency. Assume PCM resizing is triggered when the refresh pointer references a location within PA-HC-1 (Figure 14(a)), and we need to reduce the mapping table from 6 to 4 active entries (this is for discussion; in practice, we allocate at least 128 entries to the main PCM).

The actions taken to relocate useful lines in the affected chunk at device address level (i.e., DA-HC-4 and DA-HC-5) are based on the current memory layout at the time PCM resizing is activated. If a DA-HC contains useful data, then we move it to a DA-HC that contains junk data. In our example, the data in DA-HC-5 is copied to DA-HC-3. We chose DA-HC-3 because it stores the content of PA-HC-5 (i.e., “E” as shown in Figure 14(b)). Due to resizing, the OS expunges “E” and “F” such that the data stored in DA-HC-3 is no longer needed. The relocation is recorded such that in future refreshes, for example, when there is a need to swap a line from DA-HC-2 and DA-HC-5 (Figure 14(c)), we swap DA-HC-2 and DA-HC-3 instead. In another case before PCM resizing (Figure 15(a)), the destination DA-HC may be partially refreshed. DA-HC-5

CPU core	4-core CMP, 3.2GHz
L1 cache	private, separate I/D- caches, 32K, 4-way, 2-cycle hit latency
L2 Cache	private, 4MB, 8-way, LRU, writeback, 12-cycle hit latency
DRAM L3 cache	64MB, shared, 16-way, LRU, 64B linesize, writeback, 15ns hit latency
Main Memory	8GB PCM, 4 ranks of 8 banks each
PCM latency	read: 50ns, write: 1000ns

Fig. 16. Baseline configurations.

is relocated into two DA-HC places to ensure consistency. At the end of refresh, the mapping is reconstructed as shown in Figure 15(b).

Mapping within memory controller. After integrating wear-leveling and salvaging, a physical address goes through two levels of mappings before the actual device address is found and accessed, that is: (1) a mapping from PA to DA for wear-leveling and (2) a mapping from DA to Real DA (if a failure is found). Due to the relatively high cost of the second mapping, LLS is integrated in the memory controller instead of in each PCM chip/bank. Therefore the first mapping (for wear-leveling) is extracted from PCM chips [Seong et al. 2010a] and placed in the memory controller as well. This modification ensures security as the mapping tables and random keys are kept in the memory controller and inaccessible to the OS and user applications.

6. EVALUATION

6.1. Experimental Methodology

We evaluate our design with a twofold approach. For performance, we evaluate LLS using Simics [2008]. We simulate a four-core 3.2 GHz CMP; the parameters are summarized in Figure 16. Each core has private L1 and L2 caches and a shared DRAM L3 cache. Each L2 cache is 4MB and 8-way set associative. The shared DRAM cache is 64MB and 16-way set associative. As shown in Qureshi et al. [2009a, 2009b], a large DRAM cache is essential as it reduces the number of writes to PCM and enables practical use of PCM as main memory. Only data evicted from DRAM are stored in PCM. We evaluate a subset of benchmark programs (the programs that compile in our setting) from SPEC 2006. These programs have good coverage; the programs include ones with intensive memory accesses (e.g., mcf) and light memory accesses (e.g., gcc) [Bird et al. 2007]. We evaluate performance after different portions of memory cells fail. At each point, a billion instructions are simulated after a billion warmup instructions. Check-points are set after skipping the warmup phase in each program. We use latencies from Numonyx; PCM read and write latencies are 50ns and 1000ns, respectively.

Since it is impractical to simulate the whole lifetime of PCM chips of this size, we follow the same approach as Schechter et al. [2010]. We assume perfect uniform wear-leveling that evenly distributes write operations to all lines in the usable memory space. Each write alters 50% cells in one line. When distributing a fixed number of writes to PCM, the number of writes to each line is slightly higher after resizing. Therefore, we report the total number of write operations rather than the number of writes to each line.

6.2. Lifetime Study

To evaluate the effectiveness of LLS, we compared it to ECP-M and Page-Ideal, which are enhanced versions of two existing salvaging schemes. The enhancements were added to support a contiguous usable PCM space, and have no impact on lifetime. ECP-M was enhanced from ECP [Schechter et al. 2010] with a mapping table at page level (discussed in Section 3). Page-Ideal is an ideal version of Ipek et al. [2010] to support contiguous PCM space. In this implementation, when there is a need to match

Fig. 17. Lifetime comparison of different salvaging schemes with different variances.

two pages with failed cells, we optimistically assume that one of them always has the biggest address of all usable pages. We use the lower page address as the one to identify the page pair. Therefore, the visible PCM space is contiguous before and after pairing pages. Moreover, we also compared LLS to FREE-P, a line-level salvaging scheme that was developed in parallel with our scheme.

Figure 17 summarizes the lifetime comparison of different salvaging schemes. We chose the same cell variances as Schechter et al. [2010] (discussed in Section 2.3). The x-axis shows the total number of write operations. We normalized this number to the setting in which all cells have the same 10^8 write endurance, that is, no PV for $x=1$. The y-axis shows the percentage of pages that survived over the time.

In the figure, we show the result from an oracle that writes each line exactly w times to cause its six weakest cells to fail. All failed cells are then rescued by ECP. Clearly w varies across different lines due to process variations. Oracle gives the upper bound of perfect PV-aware wear-leveling, perfect line salvaging, and perfect cooperation among both. The gap between oracle and ECP motivates our line-level salvaging scheme to work with wear-leveling.

In these experiments, LLS divides the space into 128 chunks and selects 4 lines from each chunk to form a salvaging group. From the figure, we observed that LLS shrinks more space than ECP for the first batch of failures. To handle the first line failure, LLS requires one PCM resizing and removes 64MB from the main PCM; ECP-M only

marks one page as nonusable. LLS has smaller usable space at this stage. However, LLS quickly overtakes ECP-M by exploiting line-level salvaging opportunities, and thus achieves a longer lifetime. On average, LLS achieves 31% extra lifetime compared to baseline ECP: 14%, 24%, 41%, and 65% for four variances, respectively.

Figure 17 also shows that LLS wins 11% extra lifetime over FREE-p: 5%, 7%, 13%, and 22.6% under different variations. FREE-p was developed in parallel with LLS. Focusing on extending PCM chip lifetime, FREE-p corrects 4 or fewer errors with low overhead and remaps the block if there are more than 5 errors [Yoon et al. 2011]. LLS achieves better chip lifetime due to the stronger intra-line error correcting ability; it adopts ECP to correct up to 5 errors. A design alternative is to replace the BCH correction in FREE-p with ECP, which helps to achieve nearly the same lifetime as that of LLS.

In Figure 17, LLS-Ideal shows the total usable space at line level, which gives an upper bound of all line-level salvaging techniques. oracle assumes PV-aware wear-leveling and LLS-Ideal assumes a wear-leveling scheme that evenly distributes writes. The difference between LLS and LLS-Ideal are the idle lines in the backup, that is, no broken main PCM lines are mapped to them. From the figure, idle lines account for a small percentage. The loss from exploiting their available endurance is small.

Projected lifetime in months. The preceding results are the normalized number of writes. The actual PCM lifetime in months depends on many factors. As an example, if we assume each bank has 256MB (as in Figure 16), a cache line has 64B, PCM experiences stream write traffic, each write alters half of a cache line, and the cell variance is 0.25, then the projected lifetime with ECP is about 28 months before we see many failed cells, or $256\text{MB} \div 64\text{B} \times 10^8 \times 0.18 \times 1000\text{ns} = 7.2\text{e8 ns} = 28\text{ months}$. 0.18 represents that the exploited PCM endurance when PV is 0.18 of the 10^8 no-PV chip (from Figure 17(b)). In other scenarios, attack traffic to a subset of addresses may shorten lifetime [Seong et al. 2010a] while normal traffic with fewer writes will prolong the lifetime.

6.3. Hardware Cost

We next study the hardware cost to enable LLS. The hardware cost includes bitmap storage, and the control logic to enable fast address translation from main PCM to backup PCM.

6.3.1. Bitmap Storage. Each 64B line has one status bit that indicates if the line is broken or not. This bit is needed by ECP and LLS. This accounts for 0.2% off-chip storage, or 18MB (16MB bitmap and 2MB (72, 64) ECC) for a 8GB PCM memory in our setting. In addition to storing each status bit with a line, LLS redundantly gathers all bits into a global bitmap as shown in Figure 8. Due to low modification frequency, the bitmap can be stored in PCM and protected using ECP.

In comparison, to support contiguous PCM space, current salvaging schemes [Ipek et al. 2010; Schechter et al. 2010] also need a mapping table. A simple page-level mapping needs $8\text{GB}/4\text{KB} = 2\text{M}$ entries. Given a 21-bit page index, the overhead is about 4.2MB. While LLS has more metadata storage, the overhead is modest compared to the PCM space saved from exploiting line-level endurance.

Since the bitmap is stored in PCM, it is slow to access. Therefore, it is beneficial to integrate a small on-chip bitmap cache to store frequently used entries. We chose 256KB as a good trade-off between cost and performance. Since the need to access the bitmap cache varies with the percentage of failed cells, we measured the hit rate under different percentages of survived lines. The results are summarized in Figure 18. The y-axis is the percentage of accesses to failed lines that hit in the bitmap cache. We observed

Fig. 18. The effectiveness evaluation of an on-chip bitmap cache.

Fig. 19. Performance comparison of ECP, L-ECP, and LLS.

slightly higher cache hit rates due to more reuse of fetched bit vectors with more accesses. We also evaluated larger cache sizes, but observed no significant improvement.

6.3.2. Translation Logic. We did a custom design of the proposed logic using PTM 45nm technology [Cao 2008]. Our design needs 55K transistors and $13\text{K}\mu\text{m}^2$ die area. The total latency is 2.68ns ($=0.98\text{ns} + 1.24\text{ns} + 0.46\text{ns}$) if step 2 is enabled, and 1.44ns if step 2 is skipped. The translation logic is activated only if the line is broken.

Energy consumption overhead comes from accesses to the bitmap cache and translation logic. From CACTI, it costs about 0.4nJ per access for a 256K DRAM bitmap cache. The translation logic consumes $<0.1\text{nJ}$ per access. However, note that PCM energy consumption is dominated by writes, which incurs 1nJ per bit accessed. PCM read energy consumption is $<<0.1\text{nJ}$ per bit accessed, and thus this cost can be omitted. Given that a cache line contains 512 bits, and the read/write ratios of normal benchmarks are less than 10, energy overhead from the bitmap cache access and address translation is a modest 2% of total energy consumption: $(0.4\text{nJ} + 0.1\text{nJ})$ out of $(1\text{nJ} \times 512 \times 0.5 + \delta)/11$. δ is the omitted read energy consumption.

6.3.3. Sensitivity Study. To study the sensitivity in forming salvaging groups, we chose different group sizes and different chunk sizes. Figure 20 summarizes the results on different Salvaging Group Sizes (SGS) for variance 0.2 and 0.3. We found that using 128 chunks with 4 lines per chunk gives a slightly better trade-off between lifetime

Fig. 20. Comparing the lifetime of different grouping choices.

Fig. 21. Comparing the lifetime of different backup bunch sizes from each group.

and overhead. However, the difference between 64 and 128 chunks is small. Figure 21 presents the results with different numbers of lines per chunk. With larger variation, that is, 0.3, larger number of lines per chunk, for example, 32, has worse chip lifetime. This is because more healthy lines are imported into backup PCM which is invisible to OS. On the other hand, it resizes smaller number of times over the lifetime.

6.4. LLS and Layered-ECP

We next compare LLS with layered-ECP (L-ECP), a page-level salvaging scheme proposed in Schechter et al. [2010]. L-ECP reserves one 64B line for each 4KB page, that is, 64 lines. If a line contains more failed cells than what ECP can fix per line, then the reserved line is used to fix these cells. To mitigate energy overhead, a bit is associated with each line, which indicates if there are more failed cells in the extra line (i.e., if the reserved line needs to be accessed).

LLS can be built on L-ECP to gain additional salvaging opportunities at the cost of hardware overhead. The integration has two benefits. First, line failures appear late. Second, when L-ECP marks a line as broken and activates LLS to remap it to backup PCM, the line can release its occupied cells in the reserved L-ECP line such that these cells may be used to rescue other lines. While the integration is transparent to upper levels, it needs additional hardware to ensure correctness.

In this section we only evaluated and compared their stand-alone implementations, that is, no integration of L-ECP and LLS. Figure 20 shows that L-ECP extends 10–49.6% extra lifetime from ECP while LLS achieves about 10.3% more lifetime beyond L-ECP. L-ECP's performance overhead comes from the extra access to the reserved line. Figure 19 compares the performance of ECP-M, L-ECP, and LLS with different percentages of surviving memory space. On average, L-ECP has a 10% and 18% performance overhead when 90% and 60% pages survive. Instead, LLS introduces only 0.5% and 5% overhead, respectively.

7. CONCLUSION

In this article, we proposed LLS, a line-level mapping and salvaging scheme that integrates state-of-the-art wear-leveling and low-level salvaging techniques. LLS helps provide a contiguous PCM space such that lower-level line failures are hidden from the OS and user applications. In addition, LLS extends PCM lifetime by 31% on average with modest hardware cost and performance overhead.

REFERENCES

- Ansari, A., Gupta, S., Feng, S., and Mahlke, S. 2009. Zerehcache: Armoring cache architectures in high defect density technologies. In *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*. 100–110.
- Beaumont-Smith, A. and Lim, C.-C. 2001. Parallel prefix adder design. In *Proceedings of the IEEE Symposium on Computer Arithmetic*. 218–255.
- Bird, S., Phansalkar, A., John, L. K., Mericas, A., and Indukuru, R. 2007. Performance characterization of spec cpu benchmarks on intel's core microarchitecture based processor. In *Proceedings of the SPEC Benchmark Workshop*.
- Burr, G., Breitwisch, M., Franceschini, M., Garetto, D., Gopalakrishnan, K., Jackson, B., Kurdi, B., Lam, C., Lastras, L., Padilla, A., et al. 2010. Phase change memory technology. *J. Vac. Sci. Technol. B. Microelectron. Nanometer Struct.* 28, 223.
- Cabrini, A., Braga, S., Manetto, A., and Torelli, G. 2009. Voltage-driven multilevel programming in phase change memories. In *Proceedings of the IEEE International Workshop on Memory Technology, Design, and Testing*. 3–6.
- Cao, Y. 2008. Predictive technology model. <http://www.eas.asu.edu/simptm>.
- 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*. 347–357.
- Ferreira, A., Zhou, M., Bock, S., Childers, B., MELHEM, R., and Mosse, D. 2010. Increasing pcm main memory lifetime. In *Proceedings of the Design, Automation Test in Europe Conference Exhibition (DATE)*. 914–919.
- Gleixner, B., Pellizzer, F., and Bez, R. 2009. Reliability characterization of phase change memory. In *Proceedings of the European Phase Change and Ovonic Symposium*.
- Intel 2009. Intel tera scale. http://en.wikipedia.org/wiki/Intel_Tera-Scale.
- 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 International Conference on Architectural Support for Programming Languages and Operating Systems*. 3–14.
- ITRS 2009. The international technology roadmap for semiconductors report. <http://www.itrs.net/>.
- Jiang, L., Zhang, Y., and Yang, J. 2011. Enhancing phase change memory lifetime through fine-grained current regulation and voltage upscaling. In *Proceedings of the 17th IEEE/ACM International Symposium on Low-Power Electronics and Design*. 127–132.
- Kim, K. and Ahn, S. J. 2005. Reliability investigations for manufacturable high density pram. In *Proceedings of the 43rd IEEE International Reliability Physics Symposium*.
- Lai, S. 2003. Current status of the phase change memory and its future. In *Proceedings of the IEEE International Electron Devices Meeting (Technical Digest)*. 1–10.
- Lai, S. and Lowrey, T. 2001. A 180nm nvm cell element technology for stand alone and embedded applications. In *Proceedings of the IEEE International Electron Devices Meeting (Technical Digest)*.

- 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*. 2–13.
- Lee, M., Breitwisch, M., and Lam, C. 2008. Phase change memory program method without over-reset. US patent application. 12/266,222.
- Mantegazza, D., Ielmini, D., Pirovano, A., Lacaita, A., Varesi, E., Pellizzer, F., and Bez, R. 2008. Explanation of programming distributions in phase-change memory arrays based on crystallization time statistics. *Solid-State Electron.* 52, 4, 584–590.
- NVIDIA. 2010. Nvidia GeForce GTX-480 GPU Specification. <http://www.nvidia.com/>.
- 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*. 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*. 24–33.
- Qureshi, M., Franceschini, M., and Lastras-Montano, L. 2010. Improving read performance of phase change memories via write cancellation and write pausing. In *Proceedings of the 16th International Symposium on High Performance Computer Architecture*. 1–11.
- Qureshi, M., Seznec, A., Lastras, L., and Franceschini, M. 2011a. Practical and secure pcm systems by online detection of malicious write streams. In *Proceedings of the 17th International Symposium on High Performance Computer Architecture (HPCA)*. 478–489.
- Qureshi, M. K., Seznec, A., Lastras, L. A., and Franceschini, M. M. 2011b. Practical and secure pcm systems by online detection of malicious write streams. In *Proceedings of the International Symposium on High-Performance Computer Architecture*. 478–489.
- Ramanarayanan, R., Mathew, S., Erraguntla, V., Krishnamurthy, R., and Gueron, S. 2008. A 2.1 ghz 6.5 mw 64-bit unified popcount/bitscan datapath unit for 65nm high-performance microprocessor execution cores. In *Proceedings of the International Conference on VLSI Design*. 273–278.
- Rodgers, J., Maimon, J., Storey, T., Lee, D., Graziano, M., Rockett, L., and Hunt, K. 2008. A 4-mb non-volatile chalcogenide random access memory designed for space applications: Project status update. In *Proceedings of the 9th Non-Volatile Memory Technology Symposium (NVMTS)*. 1–6.
- 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*. 141–152.
- 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*. 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 43th ACM/IEEE International Symposium on Microarchitecture*.
- Seznec, A. 2010. A phase change memory as a secure main memory. *IEEE Comput. Archit. Lett.* 9, 5–8.
- Simics 2008. Simics simulator. <http://www.simics.com>.
- Yeung, F., Ahn, S., Hwang, Y., Jeong, C., Song, Y., Lee, S., Lee, S., Ryoo, K., Park, J., Shin, J., et al. 2005. Ge2sb2te5 confined structures and integration of 64mb phase-change random access memory. *Japan. J. Appl. Phys.* 44, 4B, 2691–2695.
- Yoon, D. H. and Erez, M. 2009. Memory mapped ecc: Low-cost error protection for last level caches. In *Proceedings of the 36th Annual International Symposium on Computer Architecture*. 116–127.
- 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*. 466–477.
- Zhang, W. and Li, T. 2009. Characterizing and mitigating the impact of process variations on phase change based memory systems. In *Proceedings the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*. 2–13.
- Zhang, W. and Li, T. 2011. Helmet: A resistance drift resilient architecture for multi-level cell phase change memory system. In *Proceedings of the 2011 IEEE/IFIP 41st International Conference on Dependable Systems Networks (DSN)*. 197–208.
- 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*. 14–23.

Received June 2012; revised October 2012; accepted December 2012