LLS: Cooperative Integration of Wear-Leveling and Salvaging for PCM Main Memory

Lei Jiang †, Yu Du ‡, Youtao Zhang ‡, Bruce R. Childers‡, Jun Yang †

† Electrical and Computer Engineering Department
University of Pittsburgh
Pittsburgh, PA 15261
†{lej16,juy9}@pitt.edu

‡ Computer Science Department
University of Pittsburgh
Pittsburgh, PA 15260
‡{fisherdu,zhangyt,childers}@cs.pitt.edu

***Abstract*—Phase change memory (PCM) has emerged as a promising technology for main memory due to many advantages, such as better scalability, non-volatility and fast read access. However, PCM's limited write endurance restricts its immediate use as a replacement for DRAM. Recent studies have revealed that a PCM chip which integrates millions to billions of bit cells has non-negligible variations in write endurance. Wear leveling techniques have been proposed to balance write operations to different PCM regions. To further prolong the lifetime of a PCM device after the failure of weak cell, techniques have been proposed to remap failed lines to spares and to salvage a PCM device that has a large number of failed lines or pages with graceful degradation.**

However, 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 non-contiguous PCM space generated from salvaging complicates wear leveling and incurs large overhead. In this paper, 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 seamlessly integrates wear leveling and salvaging and copes well with modern OSs, including ones that support multiple page sizes. Our experimental results show that LLS achieves 24% longer lifetime than a state-of-the-art technique. It has negligible hardware cost and performance overhead.

***Keywords*-Salvaging; Wear Leveling; Hard Faults; Phase Change Memory; Reliability;**

I. Introduction

As technology scales, the number of cores in modern chip multiprocessors (CMPs) is increasing fast (e.g., Nvidia's 480-core GTX-480 GPU [19] and Intel's 80-core TeraFlops [9]). With more threads enabled to run 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 problems due to today's small feature size. A recent ITRS report [8] indicates that there is no known path forward to scale DRAM below 22*nm*. 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., $Ge_2Sb_2Te_5$ or GST) and its peripheral logic. While PCM has many advantages, such as scalability beyond 9*nm* [8], non-volatility, fast read access, it also has limitations. One major drawback is poor write endurance — a PCM cell can be reliably written only a limited number of times. It has been reported that PCM chips can survive only 10^7 to 10^9 write cycles [1], [8], [27]. 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 2 minutes [23]. 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 DRAM write buffer [14], [18], differential-write [29], and flip-N-write [5]. And uneven writes to different PCM regions can be balanced with Start-Gap [16], security refresh [23], and table-driven segment swapping [29], [6]. 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 [16]).

When millions to billions of PCM cells are integrated into PCM arrays and chips, these cells show non-negligible 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, two 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 [10]. 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 [24]. 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-based main

978-1-4244-9233-6/11/$26.00 ©2011 IEEE

memory. However, the simple integration of these approaches results in a non-contiguous PCM space [10], [24]. This characteristic is mainly due to the "marking page retirement" mechanism adopted in salvaging. When failures are propagated from the 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 designs that use randomized PA-DA mapping, and often requires frequent data movement at runtime. Second, a marked page may still contain a few healthy memory lines. Retiring the whole page loses the opportunity to exploit the remaining endurance of these lines.

In this paper 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, such as ECP or ECC, in our baseline configuration. Thus, a line failure is encountered only if intra-line salvaging cannot correct all failed cells in the line. In the paper we describe the low-cost LLS address translation hardware and illustrate how to seamlessly integrate LLS with wear leveling technique that uses 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:

- 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-based main memory.
- We propose LLS, a novel hardware-based design that smoothly integrate wear leveling and salvaging. LLS transparently maps failed lines to backup lines and relieves the OS from managing failed pages at runtime.
- We elaborate the hardware design that effectively implements LLS with low-cost. Our experimental results show that LLS not only provides continguous memory space but also extends chip lifetime.

In the rest of the paper, we discuss background in Section II and motivate our design in Section III. We present LLS in Section IV and discuss how to integrate it with wear leveling in Section V. The evaluation is described in Section VI. Section VII concludes the paper.

II. Background

A. Phase Change Memory (PCM) and its Failure Model

A PCM cell represents a bit ("0" or "1") with two reversible states that have a significant resistance difference. 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 [12], experiments with PCM to arrays and chips have shown much lower endurance in the range of $10^7 - 10^9$ writes [1], [8], [27]. Write endurance is significant obstacle that restricts PCM from serving as an immediate and widespread replacement for DRAM.

B. 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 in a nano-scale regime [28]. Due to these variations, different cells have different *optimal* reset-set current values. A cell suffers from over-programming if a current higher than its optimal value is used. An early report showed that every 10× increase in pulse energy results in 1000× lower endurance [13], [11]. Recent measurements of failure rates on fabricated PCM chips showed similar results — 10× more failures were observed when a cell is 60% overheated [7]. While strong systematic process variations (PV) might be mitigated through circuit design, e.g., current provision [28] or customized write circuit [15], there are still non-negligible variations at the chip level.

To model PCM failures, we take the same approach as [10], [24], 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 0.3. Their model was built to be a good match to industrial observations of significant random variations at the chip level.

Other failure mechanisms, such as resistance drift and cross-talk, are neglected as described in [1]. For example, a PCM cell, if it is written reliably, can retain data for more than 10 years at 85°C. As such, PCM failures considered in this paper can be immediately detected with read-after-write. Each line write is followed by a line read to confirm if the data was correctly written.

C. 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 1 presents a conceptual view of the

desired stages to apply these techniques and their impact to system-visible PCM space and access latency.

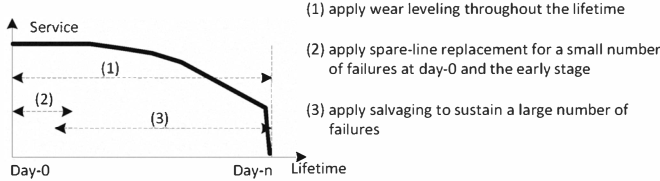

Figure 1. Conceptual view of wear leveling, built-in replacement, and salvaging.

Wear leveling aims to postpone the appearance of cell failures by spreading and balancing write operations [16], [23], [29] among all usable cells/lines. Early table-driven wear-leveling techniques [29] require OS management to periodically swap data stored in different regions based on write activity. To achieve better tradeoffs, 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) [16], [23]. 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 re-wires the address decoding logic (similar to a large capacity cache design [3]) 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.*[16] integrates a 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 [24], [10] try to continue the duty cycle of PCM chips that have even a significant amount of failed cells, e.g., Ipek *et al.*[10] 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 lifetime of PCM chip, we adopt ECP [24] 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.

III. START-GAP AND ECP

Start-Gap uses a simple linear formulation, instead of a large table, to evenly distribute write traffic across the entire device address space. Figure 2 is an example of Start-Gap. In Figure 2, 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 non-writable spare line in the device. Write operations cannot happen on *gap*, so that each device line can have a non-writable period by moving *gap*. Figure 2(a) is the initial state, where *gap* points to a spare line (device line 7) and *start* contains the device address of physical line, A. No write request can reach device line 7 in Figure 2(a), since only device line 0 to 6 are visible to the OS and device line 7 is invisible. *gap* is reducing 1 (curve (1)) in Figure 2(b). Now, device line 6 becomes the spare line. And physical line G is mapped to device line 7. In Figure 2(c), when *start* and *gap* overlap, a gap round is finished. In the next gap round, as Figure 2(d) shows, *start* is increasing 1 and *gap* is decreasing 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. To accelerate address randomization, Start-Gap can move *gap* by a random number, instead of 1, just like Figure 2(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 Feistel Network and RIB are static random address generators, malicious attack may fail PCM chip with Start-Gap within several minutes. Security refresh [23] dynamically generating randomized addresses is proposed to prevent malicious attack.

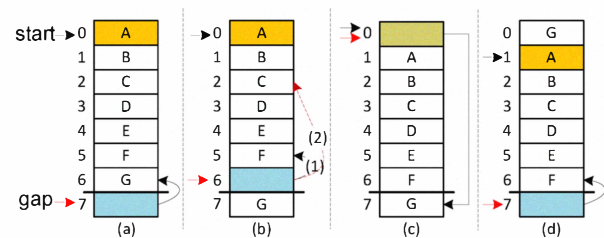

Figure 2. Start-Gap wear leveling on a memory containing 8 device lines.

Each gap move in Start-Gap is triggered by a threshold of cumulative write traffic. If this threshold is too high, the effect of wear leveling can not 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, 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 chip-level wear leveling scheme. When a write operation happens in a region, both the chip-level Start-Gap and the target-regional Start-Gap move a gap step. Only with a small extra write overhead, hierarchical design makes write traffic balanced across the entire chip as fast as possible.

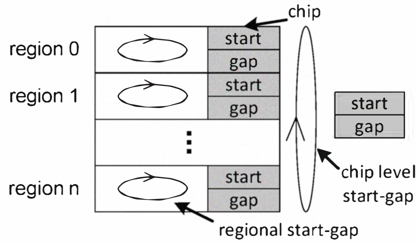

Figure 3. Region-Based Start-Gap wear leveling.

Due to natural immunity to soft errors, ECP replaces Hamming (72, 64) ECC Code on PCM. In Figure 4, a 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. 1 bit *FULL* field indicates whether these 6 ECPs are all used. The total storage overhead for one memory line is 61 bits.

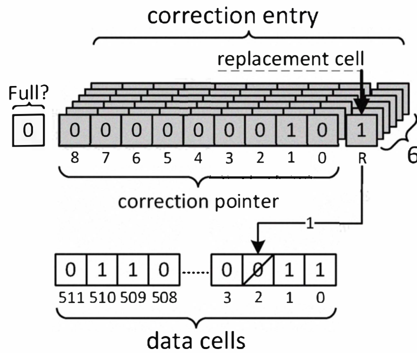

Figure 4. The 6 ECPs in one memory line.

IV. The Motivation for a Contiguous PCM Space

As wear leveling should be used throughout the whole lifetime of a PCM chip, 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, as we show next, current wear leveling and salvaging techniques are not optimized to cooperatively work together.

With an increasing number of failed cells, a salvaging scheme such as ECP cannot mask all failures. When there is a cell failure that cannot be corrected, ECP marks the OS memory page associated with the uncorrectable failure as *non-usable*. ECP relies on the OS to retire the page from further allocation to the kernel or user applications. This implicitly creates and binds a mapping between the physical address (PA) and the PCM device address (DA), and thus, it restricts the use of wear-leveling techniques based on randomized mapping. We illustrate the problem as follows.

Consider Start-Gap when failed addresses are marked by ECP. Start-Gap does randomized address mapping in two steps as shown in Figure 5. In step 1, PAs are randomized with a pseudo-random function, such as random invertible binary matrix (RIB) [16]. In step 2, the randomized PAs (RPAs) are mapped to DAs based on the current `start` and `gap` locations. Figure 5 shows that PA-100 and PA-500 are randomized to RPA-1 and RPA-(N-1) where N is the maximum size of the PCM memory. Assume PA-100 and PA-500 are mapped to DA-1 at time 1 and 2, respectively. If DA-1 fails at runtime (solid black block), then a (failure-aware) OS needs a DA/PA mapping table to expose the DA failure to the PA level. Managing this table incurs large overhead as the information is constantly updated with different `start/gap` combinations. For example, from time 1 to 2, PA-500 changes from *usable* to *non-usable*. If this address has already been allocated to a user application, then the data needs to be explicitly reallocated to a new location before this start/gap combination is used. The complication we face from this simple way to integrate Start-Gap and ECP is due to the fact that 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 non-usable. The non-contiguous memory space also limits normal operations of Security Refresh [23], which performs wear leveling by dynamically swapping two random memory lines across the entire device address space.

We discuss a strawman solution that extends current salvaging schemes with a hardware-managed mapping table, similar to the one used in built-in spare-line replacement. For a 8GB PCM space, a 21-bit table entry is required for each 4KB page if remapping is done at the page level. A 27-bit entry is required for each 64B line if remapping is done at the line level. Due to space constraints, suppose the table is created at the 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 *non-usable*. 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 non-usable. The difference between this solution and LLS is that LLS does remapping at the line level, which helps to achieve better lifetime as shown later in our experiments.

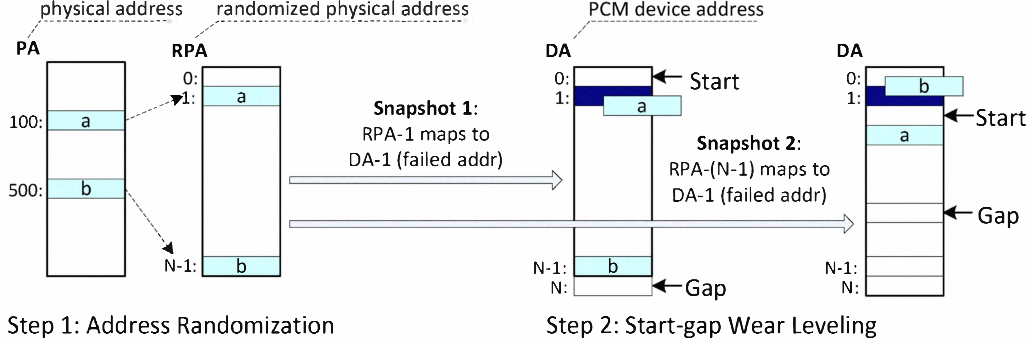

Figure 5. Start-Gap [16] prefers a contiguous PCM space (black block indicates the failed line).

V. LLS: Line-Level Mapping and Salvaging

A. Overview

An overview of Line-Level mapping and Salvaging scheme (LLS) is shown in Figure 6. 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, but backup PCM is not. Initially the whole visible address space is in main PCM. Even though 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, there are more cell failures and a line will evenutally fail that cannot 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 re-mapped 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 higher address space at the 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, 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.

B. Mapping Failed Lines to the Backup PCM

Figure 7 illustrates mapping to smoothly resize PCM memory. LLS first constructs a global bitmap using one bit per line to indicate current line status: "0" represents a healthy line and "1" represents a failed line. If a broken line is in main PCM space, then it needs to be remapped. If a broken line is in backup PCM, then it cannot be used to rescue other failed lines. The bitmap is organized as a two dimensional array. One row (Figure 7) 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 128×4=512 lines in a salvaging group. In total, we have 8GB/64B/(128×4)=256K groups. Given a bit vector for a salvaging group, all the 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, i.e., 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:

(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.*

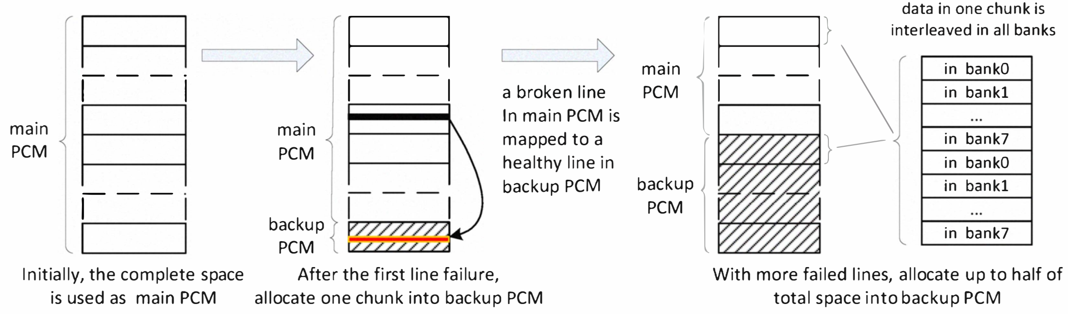

Figure 6. Splitting the PCM memory to main and backup space (To achieve graceful degradation, each chunk contains data from all banks).

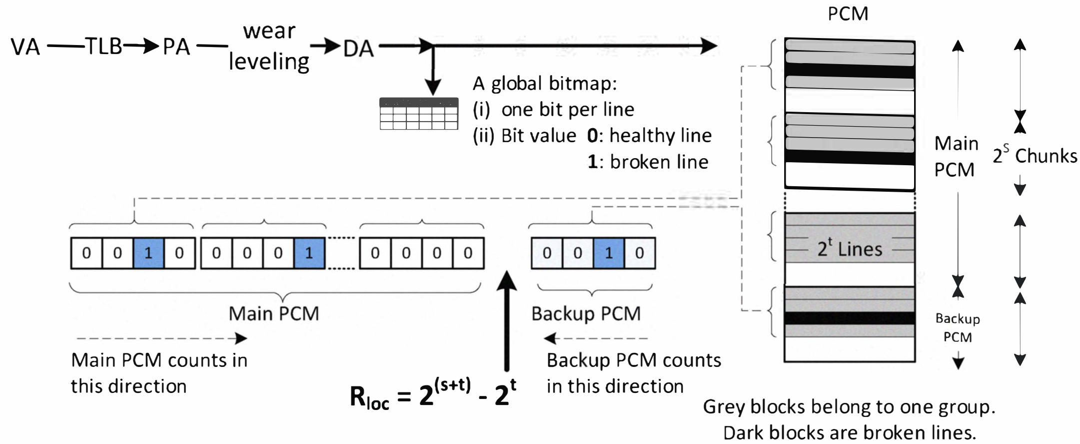

Figure 7. A salvaging group contains 2^{s+t} lines (i.e., 2^t line from each of 2^s chunks).

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 be different. In Figure 7, 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.

C. 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 above 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 the 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 re-allocated 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% capacity fail, then LLS only does 64 resize operations during the chip lifetime.

At the DA level, the space split register R_{loc} identifies what 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 the 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 way to to ensure correctness is necessary. We describe the details of how LLS ensures correct operation when we present the integration of techniques in Section 5.

VI. 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 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 elaborate on how LLS supports *Start-Gap.*

A. Integrating LLS and Start-Gap

To defend against repeated address attacks, a variation of the baseline Start-Gap, called Region-Based Start-Gap (RBSG), was proposed to enhance security. RBSG divides the whole PCM into 64MB or smaller regions in the second step of baseline Start-Gap, and performs the wear-leveling algorithm in each region independently. Address randomization is performed in the first step of RBSG. RBSG is still vulnerable to specially designed attacks such as birthday paradox attacks [25], [23]. The authors of [16] later proposed enhancements to defend against such attacks [17].

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 8(a)). A chunk in LLS is equal to or larger than a region in RBSG. At the 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, we may get a 32MB chunk size 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 as shown in [16].

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

At the device level, since we map the first half of PA to the second half of RPA, every line in the affected chunk is from the first half of PA. As shown in Figure 8(b), given a DA address DA-7800, we first use $RBSG^{-1}$ to find its RPA address RPA-7200. We then pick up PA-7200 and use the randomization function RIB to find PA-7200's RPA address, RPA-1100. The actual DA address of RPA-1100 is DA-1500. Since PA-7200 is invisible to the OS after PCM resizing, DA-1500 must be an unused line. Therefore, we can safely relocate DA-7800 to DA-1500.

Given a PA after resizing, if its randomized address RPA is within the PCM size, then we follow the original RBSG mapping to access the DA address in the corresponding region. If RPA is bigger than PCM size, then we perform another round of randomization, i.e., RIB(RIB(PA-100))=RPA-1100, to find the mapped RPA address. We need at most two rounds of randomization due to our half-to-half mapping. Our approach seamlessly integrates LLS and Start-Gap.

We also have developed a scheme for Security Refresh [23] to overcome the non-contiguous memory space problem in LLS. The main idea is to add a small mapping table to guide chip-level Security Refresh to perform swapping operations in a non-2^n size memory space. Due to limited space, we do not expand on this topic.

VII. Evaluation

A. Experimental Methodology

In this paper, we evaluate our design with a two-fold approach. For performance, we evaluate LLS using Simics [26]. We simulate a four-core 3.2GHz CMP; the detailed simulator parameters are summarized in Figure 9. Each core has private L1 and L2 caches and a shared DRAM L3

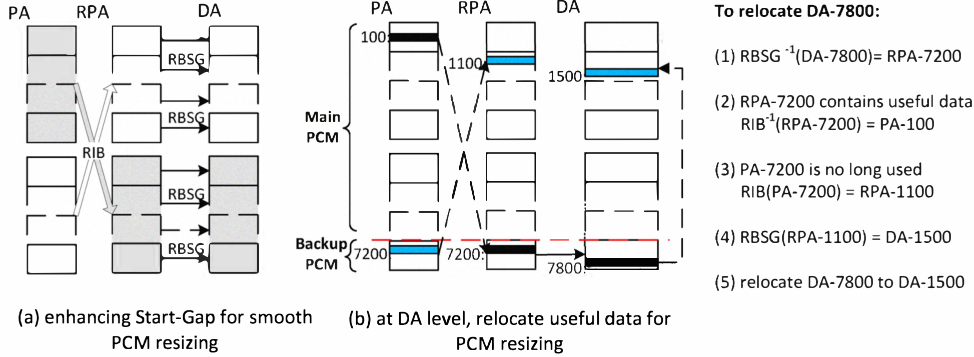

Figure 8. Integrating Start-Gap and LLS with the support for PCM resizing.

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 [18], [16], 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) [2]. We evaluate performance after different portions of memory cells fail. At each point, we simulate 1 billion instructions after 1 billion warmup instructions. Checkpoints are set after skipping the warmup phase in each program. We use recent latency numbers from Numonyx — PCM read and write latencies are 50ns and 1000ns respectively [20].

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

Figure 9. Baseline configurations.

Since it is impractical to simulate the whole lifetime of PCM chips of this size, we follow the same simplified approach from [24]. We assume uniform wear leveling that evenly distributes write operations to all lines in the usable memory space. Each write alters 50% cells within 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 to each line.

B. 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 [24] with a mapping table at the page level (discussed in Section 3). `Page-Ideal` is an ideal version of [10] to support contiguous PCM space. In this implementation, when there is a need to match 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 up these two pages.

Figure 10 summarizes the lifetime comparison of different salvaging schemes. We chose the same cell variances as [24] (discussed in Section 2.1). 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, i.e., 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 from perfect PV-aware wear leveling, perfect line salvaging, and perfect cooperation among both. The gap between `oracle` and `ECP` motivates our design of a 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 non-usable. LLS has smaller usable

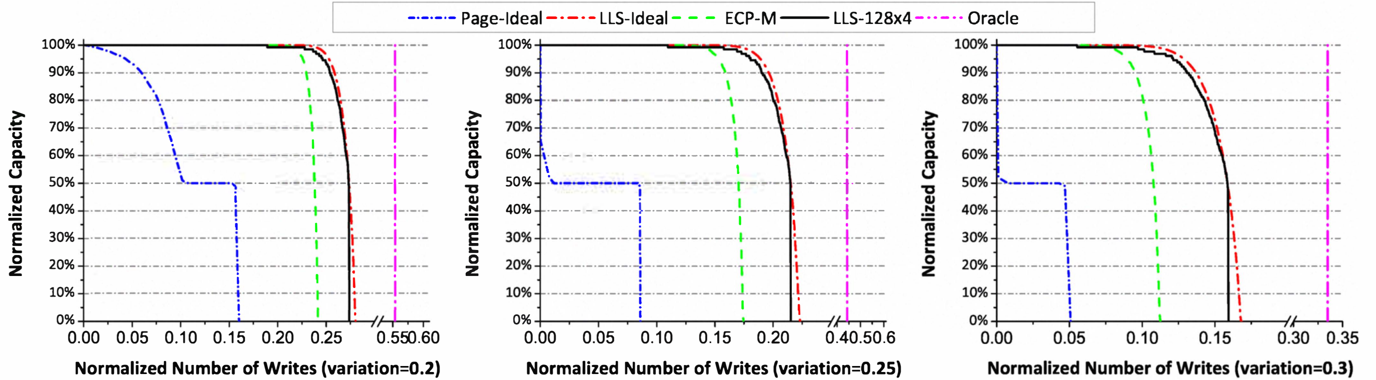

Figure 10. Lifetime comparison of different salvaging schemes with different variances.

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 24% extra lifetime when compared to the baseline ECP — 14%, 24%, and 41% for three variances respectively.

In Figure 10, `LLS-Ideal` shows the total usable space at the line level, which gives an upper bound of all line level salvaging techniques. `oracle` assumes PV-aware wear leveling and `LLS-Ideal` assumes wear leveling scheme that evenly distributes writes. The difference between `LLS` and `LLS-Ideal` are the idle lines in the backup, i.e., no broken main PCM lines are mapped to them. From the figure, idle lines account for a small percentage. The loss of exploiting their available endurance is small.

Projected lifetime in months. The above results are presented based on 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 9), 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 — $256\text{MB} \div 64\text{B} \times 10^8 \times 0.18 \times 1000\text{ns} = 7.2\text{e}8$ ns = 28 months. 0.18 represents that the exploited PCM endurances when PV is 0.18 of the 10^8 no-PV chip (from Figure 10(b)). In other scenarios, attack traffic to a subset of addresses may shorten lifetime [23] while normal traffic having less writes will prolong the lifetime.

C. 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.

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 16MB 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 bit map as shown in Figure 7. Due to its low modification frequency, the bitmap can be stored in PCM and protected using ECP.

In comparison, to support contiguous PCM space, current salvaging schemes [24] [10] also need a mapping table. A simple page-level mapping needs 8GB/4KB = 2M 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 percentage of failed cells, we measured the hit rate under different percentages of survived lines. The results are summarized in Figure 11. The y-axis is the percentage of accesses to failed lines hit in bitmap cache. We observed 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.

2) Translation Logic: 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. As described above, 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 Figures 12 and 13. 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 128×4=512 bits, and at most, the last 256 bits belong to the backup space. To simplify

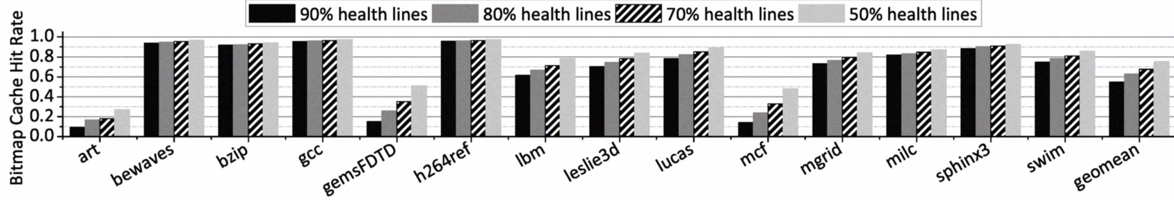

Figure 11. The effectiveness evaluation of an on-chip bitmap cache.

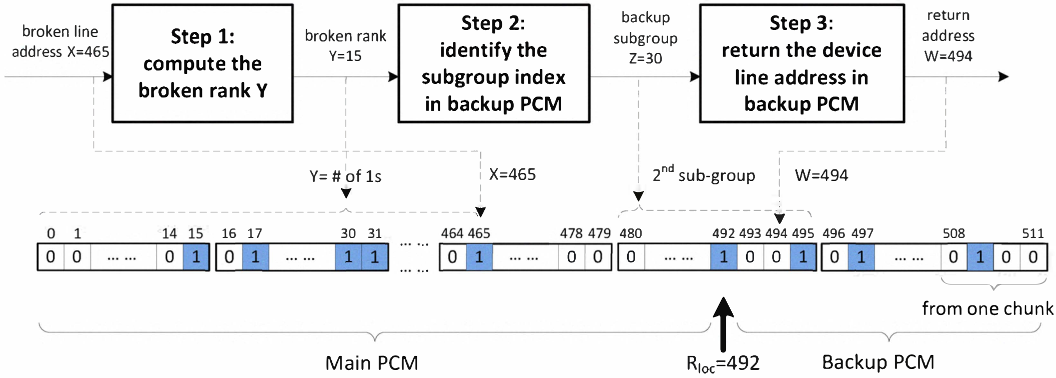

Figure 12. Address mapping in three steps.

the discussion, we assume five chunks are currently in the backup space.

Figure 12 shows an overview of our three step implementation. Given a broken line X=465 (group offset),

- 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.

Figure 13 presents three bit operations to assist address translation. The mapping from a PCM device address to its salvaging group (group id, offset) is shown in Figure 13(a). We generate two 512-bit bit-masks — A-mask and B-mask as shown in Figure 13(b). 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 5×4=20 bits of the B-mask.

Given an address X, 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

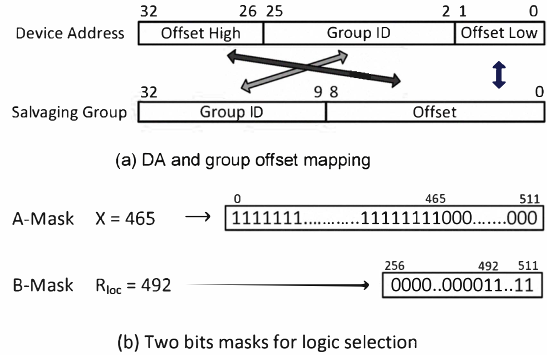

Figure 13. Bit operations to assist address translation.

logic proposed in [22]. 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 2.1GHz low power CPU using 65nm technology [22]. The 32 5-bit results are again summarized using 3:2 compressor and Wallace-tree structure to get the broken line rank Y.

Due to space constraints, we omit circuit details of the second and the third steps. We reuse the population counting logic in these steps to reduce hardware cost. Step 2 can be skipped if backup PCM contains less than 5 chunks (only one group exists).

We did a custom design of the proposed logic using PTM 45nm technology [21]. Our design needs 55K transistors and 13Kμm^2 die area. The total latency is 2.68ns (=0.98ns+

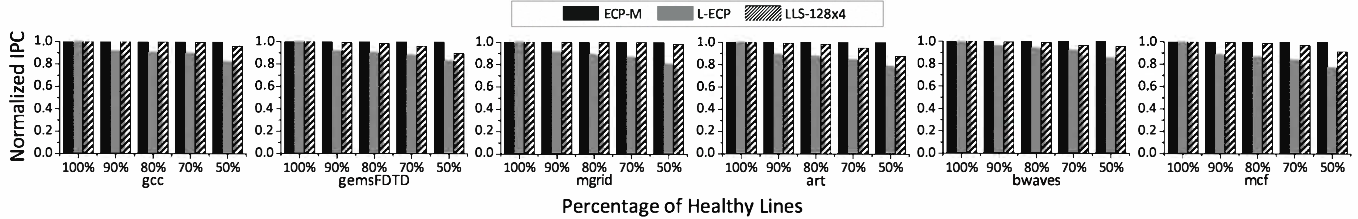

Figure 14. Performance comparison of ECP, L-ECP, and LLS.

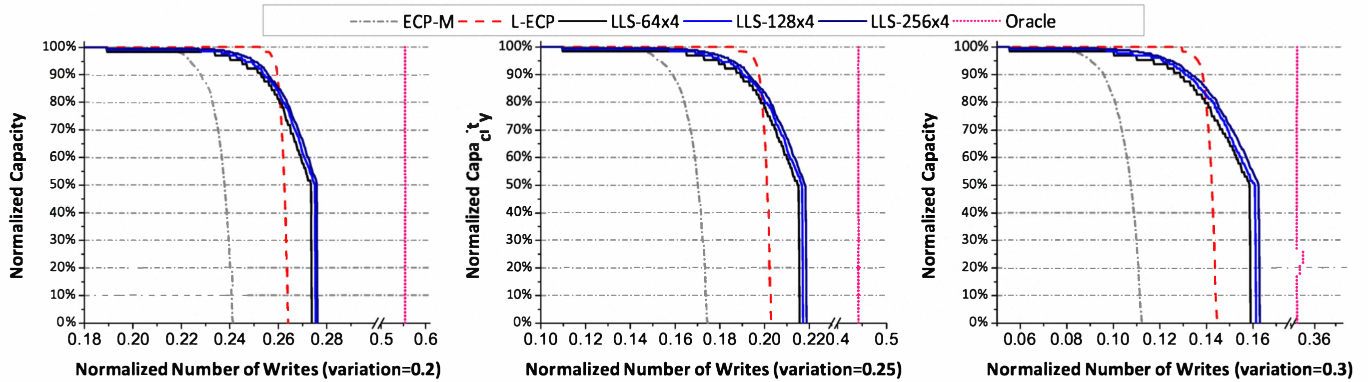

Figure 15. Comparing the lifetime of different grouping choices.

1.24ns+0.46ns) 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 <0.1nJ per access. However, note that PCM energy consumption is dominated by write access, which is 1nJ per bit accessed. The PCM read energy consumption is <<0.1nJ per bit accessed [20], and thus, this cost can be omitted. Given that a cache line contains 512 bits, and the read/write ratios of normal benchmarks is less than 10, energy overhead from the bitmap cache access and address translation is a modest 2% of total energy consumption: (0.4nJ+0.1nJ) out of $(1\text{nJ}\times 512\times 0.5+\delta)/11$). δ indicates the omitted read energy consumption.

To study the sensitivity in forming salvaging groups, we chose a different number of chunks. The results are summarized in Figure 15. We found that the setting using 128 chunks with 4 lines per chunk gives a better trade-off between lifetime and overhead.

D. LLS and Layered-ECP

We next discuss layered-ECP (L-ECP), a page-level salvaging scheme proposed in [24]. L-ECP reserves one 64B line for each 4KB page, i.e., 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, i.e. no integration of L-ECP and LLS. Figure 15 shows that L-ECP extends 10-20% extra lifetime from ECP while LLS achieves about 8% more lifetime beyond L-ECP. L-ECP's performance overhead comes from the extra access to the reserved line. Figure 14 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.

VIII. Conclusion

In this paper, 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 24% on average with modest hardware cost and performance overhead.

IX. Acknowledgmemnts

We thank the anonymous referees for their valuable comments and suggestions. The authors acknowledge the support from PCM@pitt research group. This research is also supported by National Science Foundation grants CNS-CAREER-0747242, CNS-1012070, CCF-0811295, CCF-0811352, and CNS-0702236.

References

[1] G. W. Burr, et al., "Phase Change Memory Technology," J. of Vacuum Science & Technology B, 28(2), 2010.

[2] S. Bird, A. Phansalkar, L. K. John, A. Mericas, and R. Indukuru, "Performance Characterization of SPEC CPU Benchmarks on Intel's Core Microarchitecture based Processor," *SPEC Benchmark Workshop*, 2007.

[3] A. Ansari, S. Gupta, S. Feng, and S. Mahlke, "ZerehCache: Armoring Cache Architectures in High Defect Density Technologies," *IEEE/ACM International Symposium on Microarchitecture*, 2009.

[4] A. Beaumont-Smith, and C.C. Lim, "Parallel Prefix Adder Design," *IEEE Symposium on Computer Arithmetic*, 2001.

[5] S. Cho, and H. Lee, "Flip-N-Write: A Simple Deterministic Technique to Improve PRAM Write Performance, Energy and Endurance," *IEEE/ACM International Symposium on Microarchitecture*, 2009.

[6] A. P. Ferreira, M. Zhou, S. Bock, B. R. Childers, R. Melhem, and D. Mosse, "Increasing PCM Main Memory Lifetime," *Design, Automation and Test in Europe (DATE)*, 2010.

[7] B. Gleixner, F. Pellizzer, and R. Bez, "Reliability Characterization of Phase Change Memory," *EPCOS* 2009.

[8] The International Technology Roadmap for Semiconductors report, 2007, 2009. http://www.itrs.net/.

[9] Intel Tera-scale research chip overview, http://www.intel.com/.

[10] E. Ipek, J. Condit, E. B. Nightingale, D. Burger, and T. Moscibroda, "Dynamically Replicated Memory: Building Reliable System from Nanoscle Resistive Memories," *International Conference on Architectural Support for Programming Languages and Operating Systems*, 2010.

[11] K. Kim and S. J. Ahn, "Reliability Investigation for Manufacturable High Density PRAM," *43rd Annual International Reliability Physics Symposium*, 2005.

[12] S. Lai, and T. Lowrey, "OUM – A 180nm NVM cell element technology for stand alone and embedded applications," *IEDM Technical Digest*, 2001.

[13] S. Lai, "Current Status of the Phase Change Memory and its Future," *IEDM Technical Digest*, 2003.

[14] B. Lee, E. Ipek, O. Mutlu, and D. Burger, "Architecting Phase-Change Memory as a Scalable DRAM Alternative," *International Symposium on Computer Architecture*, 2009.

[15] M. Lee, M. J. Breitwisch, and C. H. Lam, "Phase Change Memory Program Method without Over-Reset," *US Patent Application*, US2010/0110778 A1.

[16] M. K. Qureshi, J. Karidis, M. Franceschini, V. Srinivasan, L. Lastras, B. Abali, "Enhancing lifetime and security of PCM-based main memory with start-gap wear leveling," *IEEE/ACM International Symposium on Microarchitecture*, pages 14–23, 2009.

[17] M. K. Qureshi, L. A. Lastras-Montano, M. M. Franceschini, and J. P. Karidis, "Practical and Secure PCM-Based Main-Memory System via Online Attack Detection," *Workshop on the Use of Emerging Storage and Memory Technologies*, co-located with HPCA 2010, 2010.

[18] M. K. Qureshi, V. Srinivasan, and J. A. Rivers, "Scalable High Performance Main Memory System using Phase-Change Memory Technology," *International Symposium on Computer Architecture*, 2009.

[19] Nvidia GeForce GTX-480 GPU Specification, http://www.nvidia.com/.

[20] Numonyx white paper, "Phase Change Memory (PCM): A new memory technology to enable new memory usage models, "http://numonyx.com/Documents/WhitePapers/Numonyx_Phase ChangeMemory_WhitePaper.pdf.

[21] Y. Cao, "Predictive Technology Model," http://www.eas.asu.edu/∼ptm.

[22] R. Ramanarayanan, S. Mathew, V. Erraguntla, R. Krishnamurthy, and S. Gueron, "A 2.1GHz 6.5mW 64-bit Unified PopCount/BitScan Datapath Unit for 65nm High-Performance Microprocessor Execution Cores," *International Conference on VLSI Design*, 2008.

[23] N. H. Seong, D. H. Woo, and H. S. Lee, "Security Refresh: Prevent Malicious Wear-out and Increase Durability for Phase-Change Memory with Dynamically Randomized Address Mapping," *International Symposium on Computer Architecture*, 2010.

[24] S. Schechter, G. H. Loh, K. Strauss, and D. Burger, "Use ECP, not ECC, for Hard Failures in Resistive Memories," *International Symposium on Computer Architecture*, 2010.

[25] A. Seznec, "A Phase Change Memory as a Secure Main Memory," *Computer Architecture Letters*, Jan. 2010.

[26] http://www.simics.com/.

[27] F. Yeung, et al., "$Ge_2Sb_2Te_5$ Confined Structures and Integration of 64Mb Phase-Change Random Access Memory," Japanese Journal of Applied Physics, 2005.

[28] W. Zhang, and T. Li, "Characterizing and Mitigating the Impact of Process Variations on Phase Change based Memory Systems, " *IEEE/ACM International Symposium on Microarchitecture*, 2009.

[29] P. Zhou, B. Zhao, J. Yang, and Y. Zhang, "A Durable and Energy Efficient Main Memory Using Phase Change Memory Technology," *International Symposium on Computer Architecture*, 2009.