Efficient Scrub Mechanisms for Error-Prone Emerging Memories *

Manu Awasthi[±] [†]
manua@cs.utah.edu

Manjunath Shevgoor[±]
shevgoor@cs.utah.edu

Kshitij Sudan [±]
kshitij@cs.utah.edu

Bipin Rajendran [‡]
brajend@us.ibm.com

Rajeev Balasubramonian [±]
rajeev@cs.utah.edu

Viji Srinivasan [‡]
viji@us.ibm.com

[±]University of Utah, [‡]IBM T.J. Watson Research Center

Abstract

Many memory cell technologies are being considered as possible replacements for DRAM and Flash technologies, both of which are nearing their scaling limits. While these new cells (PCM, STT-RAM, FeRAM, etc.) promise high density, better scaling, and non-volatility, they introduce new challenges. Solutions at the architecture level can help address some of these problems; e.g., prior research has proposed wear-leveling and hard error tolerance mechanisms to overcome the limited write endurance of PCM cells.

In this paper, we focus on the soft error problem in PCM, a topic that has received little attention in the architecture community. Soft errors in DRAM memories are typically addressed by having SECDED support and a scrub mechanism. The scrub mechanism scans the memory looking for a single-bit error and corrects it before the line experiences a second uncorrectable error. However, PCM (and other emerging memories) are prone to new sources of soft errors. In particular, multi-level cell (MLC) PCM devices will suffer from resistance drift, that increases the soft error rate and incurs high overheads for the scrub mechanism. This paper is the first to study the design of architectural scrub mechanisms, especially when tailored to the drift phenomenon in MLC PCM. Many of our solutions will also apply to other soft-error prone emerging memories. We first show that scrub overheads can be reduced with support for strong ECC codes and a lightweight error detection operation. We then design different scrub algorithms that can adaptively trade-off soft and hard errors. Using an approach that combines all proposed solutions, our scrub mechanism yields a 96.5% reduction in uncorrectable errors, a 24.4 × decrease in scrub-related writes, and a 37.8% reduction in scrub energy, relative to a basic scrub algorithm used in modern DRAM systems.

1 Introduction

Challenges in DRAM and Flash scaling [23, 19] have ignited great interest in alternative memory technologies such as PCM, STT-RAM, Memristors, and FeRAM. Most architectural studies to date have focused on the primary problems with these new technologies: long latencies, energy-intensive writes, and limited write endurance. The last problem has fueled several recent solutions that attempt wear-leveling and hard error tolerance (Pairing [14], ECP [35], SAFER [37], FREE-p [45]). However, little attention has been paid to the problem of soft error tolerance in emerging memories. This is especially true as scaling continues and as these emerging devices employ multi-level cells (MLCs). In particular, PCM and FeRAM devices are expected to suffer from the problem of resistance drift. The resistance of a cell is expected to drift upward over time, eventually causing a cell to represent a wrong state. This is a well documented problem for PCM MLCs [5] and is also listed as an important research need for PCM and FeRAM devices by the Semiconductor Research Corporation [2].

Drift-based errors require new error tolerance solutions within the memory system. Unlike DRAM errors which are largely random, drift-based errors in PCM are imminent over long time periods and multi-bit errors are expected to be very common. This is a problem that can be mitigated with device-level solutions, but not completely eliminated. Hence, ultimately, the PCM device will expose multi-bit errors and it is up to the architecture and OS to provide fault-tolerance. This technology model is no different than what is currently assumed for state-of-the-art DRAM systems; while DRAM devices provide error margins, occasional errors are possible, and ECC support and scrub mechanisms are required to tolerate these errors. Typically, SECDED (Single Error Correct, Double Error Detect) codes are used to recover from a single bit error in DRAM. If a line already has an error, it is vulnerable to a second bit error that cannot be corrected. To prevent the occurrence of this second error, the memory is constantly examined in the background. When a single bit error is detected, it is corrected and the line is written back. This is referred to as *Scrubbing* [15]. DRAM multi-bit error rates are small because for the most part, each bit error is an independent event[1]. Hence, the DRAM scrub mechanism can be very basic and it incurs a very small overhead (one DRAM read and write every 200,000 cycles). In MLC PCM, errors are not independent; if one cell has drifted to the wrong state, there is a high probability that other cells will drift in the near future [5].

*This work has been supported in parts by NSF grants CCF-0811249, CCF-0916436, and NSF CAREER award CCF-0545959

[†]The author is currently at Micron Technology Inc..

[1]In this work, we will ignore chipkill support, which is an orthogonal consideration. If an entire chip were to fail, the handling strategy is not affected by whether the chip is DRAM or PCM.

Hence, multi-bit error rates for MLC PCM will be much higher. Our analysis shows that a DRAM-like scrub mechanism will have to issue a PCM read in nearly every cycle to achieve tolerable error rates. To address this high overhead, more sophisticated scrub mechanisms are required. This is the first body of work that observes non-trivial overheads for scrubbing and proposes optimizations to scrub mechanisms for MLC PCM. We expect this to be an important area of research for future memories with higher error rates.

To reduce scrubbing overheads, we first advocate the use of multi-bit error correction support and quantify its impact on PCM device lifetime. We then propose a lightweight read mechanism that performs approximate error detection on the PCM chip itself and reduces data transfer energy to the memory controller. Next, we introduce three variants to the scrub algorithm that are synergistic with the approximate error detection mechanism. Our policies and results expose new trade-offs in PCM design and represent different points in the energy, hard error rate, and soft error rate design spectrum. We show that our architectural policies are more effective than device-level solutions at handling drift-based errors with low overheads.

2 Background

2.1 Error Tolerance in DRAM Systems

All memory devices are expected to yield soft errors. Studies have shown that DRAM systems require varying forms of error tolerance support from the architecture or OS [36]. At a bare minimum, SECDED support is required. Many platforms augment this with a scrubbing mechanism [15, 36]. A few high-end systems invest significant architecture support to provide chipkill correctness [39, 44]. Future PCM based devices will require greater architecture (and possibly, OS) support because soft error rates and multi-bit error rates in PCM devices are expected to be much higher. The topic of multi-bit error correction in DRAM has received little attention because low-frequency scrubbing can easily handle such errors. Schroeder et al. [36] report a typical scrubbing rate of 1 GB every 45 minutes, which is a small overhead (one 64 B cache line every 200,000 cycles). Every scrub operation transfers a cache line from the DIMM to the memory controller, where error detection and, if required, write of the corrected data is performed. As we show later, an unoptimized PCM system would incur a significant overhead from the basic scrubbing mechanism; a cache line would have to be transferred to the processor almost every other cycle.

2.2 PCM MLCs

While the phenomenon of drift manifests in multiple memory cell technologies and there may be many sources of soft errors in future memories, this paper will focus on drift-induced soft errors in MLC PCM devices.

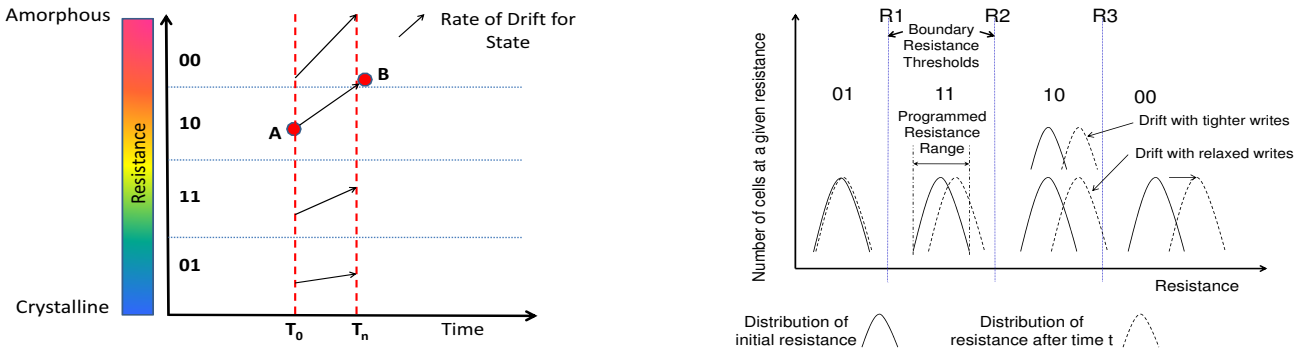
PCM cells are programmed with electrical pulses that heat the chalcogenide material. A cell can be programmed to a high-resistance amorphous state, a low-resistance crystalling state, or states that lie between these two extremes. A single-level cell (SLC) only uses the extreme states and assigns a 1 (SET) value to the crystalline state and a 0 (RESET) value to the amorphous state. By suitably controlling the programming process, a cell can be programmed into a hybrid state with a given percentage of amorphous material. As a result, a cell can have any resistance between the purely crystalline and purely amorphous extremes. The available resistance range (usually between 10^3 and 10^6 Ω) can be partitioned into multiple regions, each region representing a different state. This gives rise to *multi-level cells (MLCs)*, where a single cell can represent n bits of information by partitioning the material's resistance range into 2^n different levels. Figure 1 shows the distribution of resistance in an MLC, and the subsequent onset of drift. The resistances that represent the boundaries between neighboring levels (states) are referred to as the *boundary resistance thresholds* (R1, R2, and R3 in Figure 1b). For example, a cell may represent the '10' state if its resistance is between the boundary resistance thresholds of $10^{4.5}$ (R2) and $10^{5.5}$ (R3) Ω. It is expected that MLCs will be the primary source of density increments across successive PCM generations [1].

The programming process for MLCs is iterative [29]. After an initial Reset pulse, many short pulses are provided to gradually decrease the cell's resistance. After each pulse, the cell is read to confirm if the resistance is within the specified *programmed resistance range* for the desired state. To provide margin for error, the programmed resistance range for a state is usually tighter than the gap between the surrounding boundary resistance thresholds. At the end of a successful write, a cell's resistance is within the programmed resistance range for that state, but the exact resistance will be a function of variations in the manufacturing, programming, and crystallization processes. Prior work [17, 4] has shown that after the programming process, the resistance of the cell will follow a normal distribution: with a high probability, the cell resistance will be at the midpoint of the resistance range; with a small probability, the resistance will lie at the edges of the range (solid line distributions in Figure 1b).

2.3 Resistance Drift in PCM

Once a cell is programmed, the presence of defect structures in the chemical lattice of the chalcogenide material leads to changes in the programmed resistance, or short-term *resistance drift* [28]. The "defective crystals" stabilize over time; the material becomes more amorphous and acquires a higher resistance. Such defects are less common in the crystalline state and hence resistance drift is less significant for a cell programmed into a state with a higher percentage of crystalline material (depicted in Figure 1a). The drift is not affected by read operations; it is corrected only when the cell is written, at which point, the drift process starts all over again.

The move from SLC to MLC introduces one significant



(a)Initial programmed resistance (A) and resistance (B) after some elapsed time for a single cell.

(b)Distribution of initial programmed resistance and resistance after drift

Figure 1: Illustration of the resistance drift phenomenon.

challenge - drift-induced soft errors. Drift is not a problem for SLC PCM for two reasons. First, the rate of resistance drift is very small if the cell is programmed as mostly crystalline (Figure 1a shows that each cell drifts at a different rate). Hence, it will take a very long time for a cell programmed to be crystalline (SET) to drift to a high resistance value that represents the RESET state. Second, a cell programmed to be in the RESET state has a high rate of drift, but all resistances above the specified boundary resistance threshold represent the RESET state; so drift does not cause the cell to represent an erroneous state. However, as shown in Figure 1a, in a 4-level MLC PCM, drift can lead to frequent errors. The drift rate is higher for cells programmed with higher resistances. Consider a cell that is programmed to have an initial resistance **A** that represents the 10 state. Such a cell has a high rate of drift and after a relatively short period of time, the cell's resistance **B** becomes higher than the boundary resistance threshold. At this point, it starts to represent the neighboring 00 state, causing an error. This is an example of a "soft" or "transient" error. The error rate will increase dramatically as the resistance range is partitioned into more levels and boundary resistance thresholds are more closely spaced.

When a cell is programmed to a given state, its initial resistance R_0 will lie somewhere within the programmed resistance range for that state and follows a normal distribution (solid curves in Figure 1b). The dotted curves in Figure 1b show the distribution of resistances after some time has elapsed. In essence, each resistance has drifted to a higher value, with some cells having a resistance that lies within the threshold boundaries of the adjacent state.

Drift within the RESET state (00 in a 4-level MLC or 000 in an 8-level MLC) is not problematic because a higher resistance will continue to represent the RESET state. However, its adjacent state (10 in the 4-level MLC in Figure 1b) is considered the most drift-prone as it has the next highest drift rate and is at risk of drifting past the resistance threshold of the RESET state. Hence, to a large extent, this paper will focus on worst-case behavior in cells that are programmed to the most drift-prone state (10). We assume the use of a Gray code mapping policy that ensures that adjacent states differ in only one bit. Thus, every drift-based cell error corresponds to only a single bit error. As shown in Figure 1, we assume that the mapping of states in order is 00 (amorphous), 10, 11, 01 (crystalline).

2.4 Modeling Drift

Jeong et al. [16] describe the following equation to measure resistance drift. Given an initial resistance of R_0, the resistance $R_{drift}(t)$ can be calculated as : $R_{drift}(t) = R_0 \times t^{\alpha}$, where α represents the drift exponent and t is the time elapsed (in seconds) since programming the cell.

The value of R_0 for a given state follows a normal distribution with the mean lying at the mid-point of the programmed resistance range defined by the iterative write process [24] for that state. For most experiments in this paper, we will assume that the programmed resistance range is set to include resistances that are within ($\pm$) 2.75σ of the mean, where σ refers to the standard deviation of the normal distribution. The boundary resistance threshold is at a distance of ($\pm$) 3.0σ from the mean, allowing some margin for drift. Other programmed resistance ranges are also considered in Sections 2.5 and 5.5.

The drift exponent α depends on a number of factors. The value of α is impacted most by the state represented by the cell. The higher the initial resistance R_0, the higher the value of α. For the nominal material thickness, we adopt expected values of α for each state according to empirical data [33, 17, 4, 13]. The exact value of α for each state follows a normal distribution around the expected value for that state and is also modeled. The parameters for our drift model are based on the assumptions of Xu and Zhang [43, 42] and are summarized in Table 1.

The type and thickness of chalcogenide material impacts the drift exponent. For this study, we assume the empirical values of α recommended for standard GST material [16, 18]. Temperature also impacts the drift rate [34, 12, 46]. For this study, we assume that page coloring/wear leveling techniques will ensure load balance across banks and not

4 levels/Cell					
Storage level	Data	lg R_0		α	
		mean	deviation	mean	SDMR
0	01	3.0	0.17	0.001	40%
1	11	4.0		0.02	
2	10	5.0		0.06	
3	00	6.0		0.10	

Table 1: Configuration of 4 Level MLCs [43]. SDMR stands for Standard Deviation to Mean Ratio.

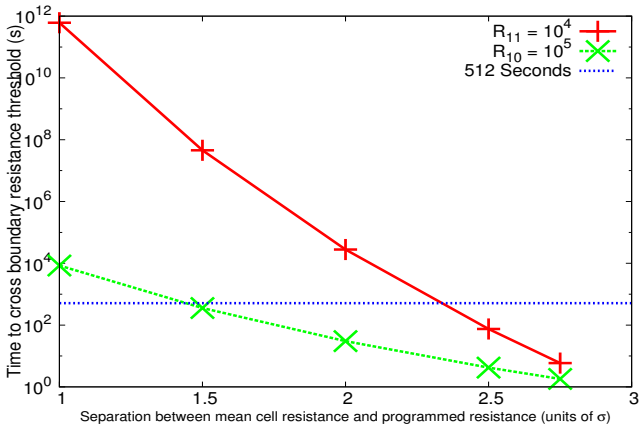


Figure 2: Representative time to drift for cells in most drift-prone states (10 and 11) in 4 level MLC as a function of separation from the mean of programmed resistance. For example, 2.5 on the X-axis refers to a cell that was programmed to a resistance of $mean + 2.5\sigma$. Boundary thresholds are at $mean + 3\sigma$. Models assumed described in Table 1, verified against published experimental data [17, 4, 13]. 512 seconds is the maximum Refresh Rate considered in this study.

create localized hotspots. If the uniform temperature of a PCM chip increases, scrub rates must be increased, and such adaptive policies are discussed in Section 3.3.

2.5 Inadequacy of Device-Level Solutions

We validated that the drift times estimated from the above drift model match the drift times derived using the distributions of R_0 and α from empirical data in prior work [43, 17, 4, 13]. Using the model discussed above, we show in Figure 2 the drift time for a 4-level cell programmed in one of the drift-prone states (10 or 11).

Figure 2 shows that for a given boundary threshold (3σ from the mean), if the cell is programmed in the 10 state at 1σ from the mean, drift time for error is under 10^4 seconds. Furthermore, the drift time drops to less than 100 seconds when the cell is programmed at 2σ from the mean resistance. Therefore, it is clear that if the problem of resistance drift is not addressed effectively, cells that have been programmed reasonably accurately cannot be treated as truly non-volatile [5].

To delay the onset of drift, it may be possible to widen the state's boundary thresholds. However, doing so compromises the density advantage of PCM, and significantly reduces the number of levels of a multi-level cell. In fact, we expect that the boundary thresholds will be squeezed closer together with each new generation.

It is not possible to introduce a correction to the read resistance during a read. For example, it has been suggested that a reference cell in a row be used to normalize the resistance of each cell in that row [26]. But this is likely to not be very effective because of the significant difference in drift times for different cells programmed to the same state, as can be seen in Figure 2. Some recent work has also proposed mitigating the effects of drift with better coding strategies [26]. While coding strategies can help and are better than reference cell based strategies, they cannot eliminate drift-based errors. Papandreou et al. [26] show that modulation coding can help an MLC PCM exhibit a raw error rate of 10^{-5} after 37 days at room temperature; this is well short of the 10^{-15} error rate target that is acceptable for such an experiment. This shows that two of the leading device-level solutions (coding and reference cells) to date are not adequate.

Based on various initial empirical data, for most of this study we assume that the boundary threshold for a state is 3σ away from the mean and the programmed resistance range is within 2.75σ from the mean. The drift time for the worst-case cell can be as low as 1.81 seconds, making it necessary to explore techniques to effectively mitigate the effects of resistance drift. If the programmed resistance range is made narrower with a more precise write process, the worst-case drift time can be increased. However, doing so causes an increase in write latency and a drop in cell endurance. The corresponding trade-offs showing that architectural techniques are more effective than precise write process to combat drift are discussed in Section 5.5.

In summary, various device-level strategies can be used to delay the onset of drift. These strategies can therefore mitigate the problem, but not eliminate it; they must be augmented with architectural solutions. These device-level strategies also incur significant cost. Our goal here is to out-do these device-level techniques with simple architectural policies. This is a philosophy similar to that used for modern-day DRAM – DRAM chips are produced at very low cost and exhibit errors; these errors are tolerated with architectural solutions that use error correction codes, RAID-like redundancy, and scrubbing.

2.6 Naive Refresh and Basic Scrub

DRAM systems employ a refresh operation to handle charge leakage in cells and a scrub mechanism to correct single-bit errors. We first show that such baseline mechanisms are highly inadequate for PCM MLC systems.

The drift problem can be addressed if a row can be read and re-written before the worst-case cell can drift to a neighboring state. For most of this study, we assume that the iterative write process can force R_0 to be in a well-defined programmed resistance range (within 2.75σ of the mean), so cells will not immediately be on the brink of drifting to a neighboring state. With such a baseline, the worst-case drift time is 1.8 seconds and every line must be refreshed every 1.8 seconds. Since a cell can typically only endure 10^8 writes, this limits the PCM device's lifetime to 1.8×10^8 sec-

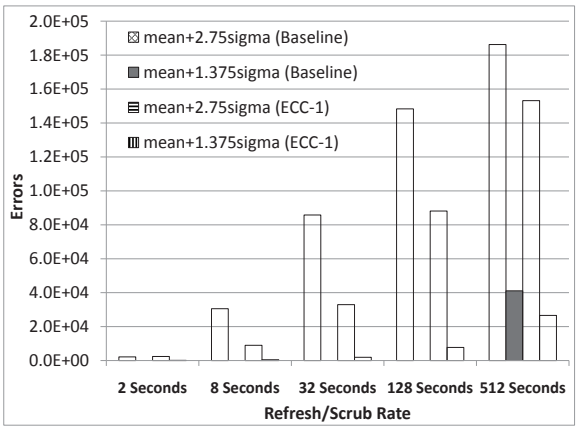


Figure 3: Uncorrectable errors in baseline systems for different programmed resistance ranges for 10^7 long-term writes. Baseline refers to a model where a re-write (refresh) is performed for each line at the specified interval. ECC-1 refers to a DRAM-like scrub mechanism where a re-write is performed only after a single error is detected. The error is detected by issuing reads at the specified interval.

onds, or only 5.7 years (ignoring short-term writes [2]). More importantly, we must refresh all of the billion 64 byte lines in a high-capacity PCM memory within 1.8 seconds; assuming 1000 ns for a PCM write [31], we must handle simultaneous refresh of about 600 lines (each 64 bytes), a process that will overwhelm the PCM system. If we assume that writes are precise enough such that the programmed resistance is within 1.375σ of the mean, the worst-case drift time is now around 500 seconds. As we show later, precise writes are expensive for many reasons: they incur a longer write time, worsen performance, and degrade lifetime. Even with such a device-level solution, simultaneous refresh of about 4 64-byte lines will have to happen simultaneously, much more onerous than the current refresh process in DRAM.

Alternatively, each line can have a SECDED code and we can keep reading PCM lines at a specified rate until a single error is detected. Only then is the corrected line re-written. Since the decoding for the SECDED code is performed at the memory controller, every line read during the scrubbing process must be sent to the processor over the memory channel, assuming an on-chip controller in modern processors. This is the basic scrub mechanism in use for DRAM-based systems today. Figure 3 shows the uncorrectable error rates for this scrubbing mechanism (ECC-1) for different programmed resistance ranges and different intervals for the read operation (detailed methodology in Section 4). Even if we assume precise writes and a 2-second read interval, as many as 97 uncorrectable errors are encountered. Further, the entire PCM main memory capacity will have to be sent to the processor within 2 seconds (roughly one cache line every 2 cycles), more than saturating even the highest-performing modern memory channel and incurring high dynamic energy cost. If the read interval is increased to a more manageable 512 seconds, the uncorrectable error rates are well over 10^4. Thus, modern solutions that work well for DRAM are highly inadequate for PCM in terms of their impact on error rates, lifetime, energy, and bandwidth needs.

[2] If the gap between writes to a block is very high, the writes are referred to as long-term. Such blocks are vulnerable to drift-based errors.

3 Architectural Support for Efficient Scrub

Drift-prone PCM systems introduce a new energy, performance, and endurance bottleneck: the scrubbing process. We carefully consider the design of efficient scrub mechanisms and propose novel solutions.We introduce a three-pronged approach to tackle the drift problem: (i) stronger ECC codes, (ii) low-cost and approximate error detection within a PCM chip, and (iii) scrub algorithm extensions that are conservative and adaptive. Most results are presented in Section 5. We mention a few in this section to preserve context. Please refer to Section 4 for detailed methodology.

3.1 Multi-Bit Error Correction Support

DRAM-based systems assume SECDED support and have a scrubbing process to trigger error correction before the incidence of a second error. If the gap between the first and second error is shorter than the scrub interval for a line, the second error may manifest before the first can be corrected, resulting in an uncorrectable multi-bit error. Thankfully, this gap is very high in DRAM systems and even a large scrub interval (3 hours for a 4 GB memory system) can yield very low uncorrectable error rates [36]. In MLC PCM devices, the gap between the first and second error in a line is much smaller than in DRAM devices, requiring a faster scrub rate. It appears intuitive that we can lower the scrub rate if we had support for multi-bit error correction. If we assume that we have a code that can correct E errors[3], then the scrub mechanism initiates recovery when it detects the E^{th} error. The required scrub rate is determined by the typical gap between the E^{th} and $E + 1^{th}$ errors in a line. For the drift model described in the previous section, we observe that as E increases, this gap does increase. Hence, if we have support to recover from many errors, we can get by with a longer scrub interval. The reason is the exponent term in the drift equation and the gaussian distributions of R_0 and α. In short, the worst-case cell has an order of magnitude less drift time than the next-worst cell, and so on.

Hence, the first change that we advocate is the introduction of multi-bit error correction codes. For a 512-bit line, Table 2 quantifies the storage overhead incurred by multi-bit error correction codes, as expressed by the Hamming bound [10]. The storage for ECC bits grows linearly with E. We observe that eight errors can be corrected with a storage overhead of 14.25%. We argue that this is an acceptable overhead as it is similar to the overhead for modern SECDED DRAM systems that have an 8-bit SECDED code for each 64-bit word.

[3] As a shorthand, we refer to such a code as *ECC-E*.

Number of Correctable Errors	Additional Bits of ECC Storage (512-bit line)	Storage Overhead (% for 512-bit line)
1	10	1.95%
2	19	3.71%
4	37	7.22%
8	73	14.25%

Table 2: ECC Overheads.

A second problem with using large ECC codes is that ECC codes can reduce PCM device lifetime [35]. When writing a PCM line, to improve endurance, we assume that a cell is written to only if the state has changed [20, 47]. While data bits in a cache line don't always undergo change, ECC bits have high entropy, *i.e.*, they are very random and very likely to flip (from 0 to 1 or vice versa) on every write. If we are writing a 512-bit line (256 4-level cells), we observe that across a suite of benchmark programs, 118 cells (46%) are expected to be re-programmed. For the corresponding 37 ECC-8 coding cells, 75% of the cells are expected to be re-programmed, i.e., a much higher level of write activity. If we assume that word and row-shifting wear-leveling optimizations [32, 6, 31, 47] are employed, we can claim that the higher activity in the ECC code will be spread across all cells. By adding an ECC-8 code to a 512-bit word, as done above, the average write activity of a cell is increased by a factor of 1.07. This 7% reduction in lifetime is worth noting; the corresponding reduction in lifetime for an ECC-4 code is only 4%. As we show later, the more efficient scrubbing process can reduce the number of scrub-related writes and possibly offset this 7% drop. This observation highlights that a scrub process must be designed carefully to balance several soft error, hard error, energy, and storage considerations. A third problem with strong ECC codes is the high cost of encoding and decoding. We will address this concern in the next sub-section.

While the use of strong ECC codes has been proposed here for soft error tolerance, it also doubles up as a hard error tolerance mechanism. Programming a PCM cell results in repeated expansion and contraction of the chalcogenide alloy. This increases the probability that the material physically detaches from the heating element, resulting in the cell being permanently stuck-at some value. Recently proposed techniques such as Pairing [14], ECP [35], SAFER [37], and FREE-p [45] address hard error tolerance in PCM, but do not address drift-induced soft errors. On the other hand, the proposed *ECC-E* code can be used to tolerate up to E errors and these errors can be either hard or soft errors. Since ECC is being maintained across a fairly large line, an ECC-8 code has a storage overhead of only 14%. In comparison, for an ECP scheme [35] to tolerate eight hard errors in a 512-bit line, a storage overhead of 16% would be incurred, and the PCM device would be intolerant of soft errors.

3.2 Light Array Read for Drift Detection (LARDD)

In conventional scrub mechanisms for DRAM, a scrub operation is issued once every 200,000 cycles [36]. This is an infrequent operation and is not worth optimizing. However, as shown in Section 2.6, scrub operations in MLC PCM devices are expected to be much more frequent. We therefore propose a different access pipeline for scrub operations.

Already, in DRAM systems, refresh is known to be a growing bottleneck because retention times are not changing, but capacity continues to increase. As a result, refresh control is moving from the memory controller to the DRAM chips [15]. Modern DRAM chips already have structures that can track the last refreshed row and timing deadlines for the next refresh operation. In a similar vein, we propose localized PCM chip control for scrub operations so that data is not constantly being shipped to the memory controller on the memory bus.

We advocate that, similar to a DRAM refresh operation, PCM rows are read from arrays, and simple logic on the PCM chip performs error detection. If panic is not triggered, the chip moves on to the next operation and no write is performed. The memory controller and processor are involved only if the error detection panics and demands that the row be corrected and re-written. The key to making this happen on a PCM chip is simple error detection. As described earlier, it is desireable to have support for codes that can correct multi-bit errors. However, the encoding and decoding cost for BCH codes is significant. The required circuitry [27] can likely not be accommodated on high-density memory chips. We therefore add a simpler error detection mechanism in addition to the BCH codes. The BCH codes are still required for eventual error correction at the memory controller, but the simpler error detection logic is invoked in the common case when the scrub operations are issued within a PCM chip. Our error detection logic is a parity-like scheme. For a 256-cell line, we partition it into eight 32-cell fields. Within each 32-cell field, we count the number of drift-prone (10) states and track if this number is odd or even ("parity"). During a scrub operation, we check to see if the number of drift-prone states within each 32-cell field has deviated from the recorded "parity". Every deviation is counted as an error and we sum the number of errors across all eight 32-cell fields. Such error estimation is clearly approximate; we must therefore be conservative in flagging a panic and re-writing a line. In Section 5, we show that such a circuit has very low cost and can possibly be included on a PCM chip. The storage overhead for parity is 8 bits per 512-bit line, a 1.625% overhead in addition to the 14.25% already being incurred for the BCH code.

We assume that parity is maintained for each 32-cell field within a PCM chip so that the chip can perform its own autonomous scrub operation in the common case without coordinating with other PCM chips. The BCH code may be associated per cache line and could be striped across multiple PCM chips. The entire cache line (plus BCH code) is read out of all PCM chips in the rank only when a panic is triggered.

The proposed read pipeline is referred to as a *Light Array Read for Drift Detection (LARDD)*. The PCM chip sequentially reads each row, performs the approximate parity-

based error detection, and moves on if no re-write is required. If the error detection raises a panic signal, the row is shipped to the memory controller where error correction and re-write are handled. This is unlike the DRAM scrub process, where every cache line is shipped to the memory controller. The scrub process is therefore made up of several periodic LARDDs and occasional re-writes to a line. The scrub interval is the time between successive LARDDs to a given PCM row.

3.3 Scrub Algorithms

Our basic scrub algorithm issues LARDDs at a specified frequency, and triggers a line re-write when a minimum number of parity-based errors are encountered in that line. We now introduce a few extensions to this basic algorithm that make the scrubbing process more efficient.

Headroom. The first extension is the use of headroom. If we assume that we have an ECC-8 code that can recover from up to eight errors, a line re-write must be triggered before the line encounters its ninth error. If the LARDD triggers a re-write after observing eight parity-based errors, many uncorrectable errors may slip through. This can happen for two reasons. One, the parity-based error detection mechanism is approximate; eight parity-based errors may represent more than eight actual errors and the line would be uncorrectable. Second, if the eighth and ninth errors happen in quick succession without an intervening LARDD, the line becomes uncorrectable. The probability of such uncorrectable errors can be reduced by triggering a line re-write well before the maximum error budget E is reached. This is referred to as the *Headroom* scheme, where a line re-write is triggered as soon as the parity-based error detection circuit identifies at least $E - h$ errors. If a high headroom h is provided, the uncorrectable error rate drops significantly. But because we are being conservative in our error correction, line re-writes are triggered more often, thus accelerating wearout and increasing the hard error rate. Thus, the choice of headroom introduces a trade-off between hard error rates and uncorrectable soft error rates.

Gradual. The *Gradual* scheme uses non-uniform LARDD rates for each line. If a line has very few drift-based errors, it is unlikely that the next LARDD will trigger a re-write. Hence, a line can employ a small LARDD frequency and increase the frequency as more drift-based errors are detected. To keep the design simple, we assume only two LARDD frequencies. The LARDD frequency is doubled after $E-h-g$ parity-based errors are encountered. Each line must maintain a Gradual bit to track its LARDD frequency; those lines with a Gradual bit set to zero will skip every alternate LARDD. These Gradual bits can be stored in the PCM memory itself. An entire row of Gradual bits (representing hundreds of PCM rows) can be read at a time by the LARDD controller on the PCM chip. After all those rows have been scrubbed, the updated row of bits can be written back into PCM cells. The *Gradual* scheme can cause a slight increase in the uncorrectable error rate (in the rare event that a flurry of soft errors happen between consecutive LARDDs), but it can significantly reduce the energy overhead associated with LARDDs.

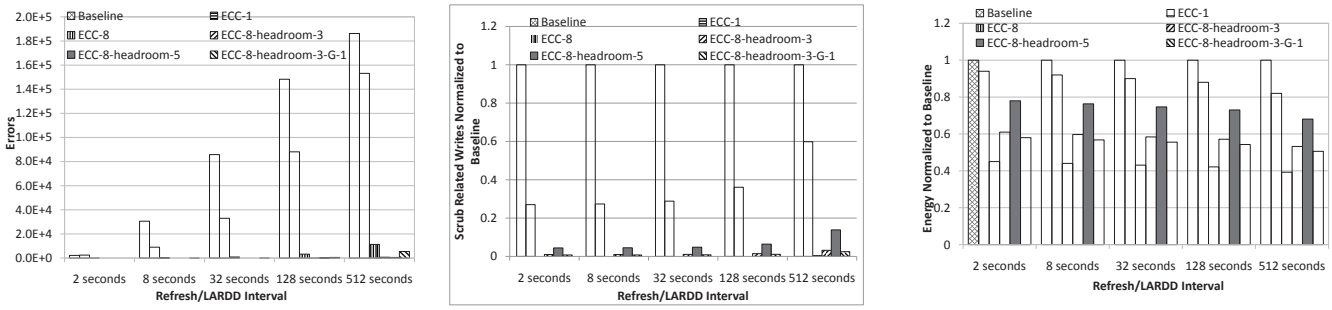
Adaptive. The uncorrectable error rate can be a function of dynamic events. For example, prior work [46] has shown that drift is accelerated at higher temperatures. If a PCM device heats up, the pre-selected LARDD rate may lead to an unexpected number of uncorrectable errors. Similarly, if we execute a workload that has many more drift-prone states, the error rate would again go up. In yet another example, that will be used as an adaptive LARDD case study in this paper, as a PCM device ages, the number of worn out cells (hard errors) in a line increases. This reduces the soft error budget for that line as only a maximum of E errors (hard or soft) can be corrected by the *ECC-E* code. Again, a pre-selected LARDD rate will cause more uncorrectable errors as the number of hard errors in the device increases. As a result, the LARDD rate will necessarily have to be adaptive. Every epoch (say W writes), we examine the uncorrectable errors in the last epoch, and double the LARDD rate if the errors exceed a threshold. Assuming that we start with a LARDD policy with headroom h, we get rid of the headroom policy after a line is known to have at least $E-h-1$ hard errors (to prevent the full re-write from happening almost immediately after the write). We also consider an adaptive policy where the faster LARDD rate is only applied to lines with greater than HE hard errors. Similar to the Gradual optimization described earlier, a bit is tracked per line to figure out when LARDDs must be skipped.

4 Simulation Methodology

Since we must model millions of writes to a PCM device over its lifetime, detailed cycle-by-cycle simulations with real workloads is not an option. However, our experiments are frequently guided by observations and data from detailed Simics simulations which model PCM as main memory storage behind a 256 MB DRAM cache, and simulate multi-threaded and multi-programmed workloads from PARSEC [3] and SpecJBB2005.

In each experiment, we model the behavior of 10^7 long-term writes to 64 byte cache lines. For each write, we probabilistically estimate the cells that will encode states 00, 01, 10, and 11 (for a 4-level MLC). The probabilities are derived from Simics simulations which capture the percentage occurrence of each state in actual cache line data. For each cell, we estimate the value of R_0 and α based on the normal distributions specified in Table 1. Based on this, we compute the time for each cell to drift to a neighboring state and identify the cells that will be among the first to fail.

We assume robust wear-leveling techniques [32, 6, 31, 47] will be used, and a 64 GB PCM main memory will likely handle well over 10^{17} writes in its lifetime. Many lines may have successive writes within a short interval and will not be prone to drift-based errors, *i.e.*, the line is re-written before its worst-case cells drift to neighboring states. But we expect many of the lines in the PCM device to have a long interval between successive writes, especially in the follow-



(a) Uncorrectable Errors (b) Scrub-Related Writes (c) Scrub Energy

Figure 4: Error rates, scrub-related writes, and scrub energy for various schemes as a function of Refresh/LARDD interval.

$Energy_{Read}$	$Write\ Energy_{01}$	$Write\ Energy_{11}$	$Write\ Energy_{10}$	$Write\ Energy_{00}$
10 pJ	50 pJ	100 pJ	400 pJ	1600 pJ

Table 3: Per (Cell) Access Read and Write Energy for different programmed values derived from [41].

ing situations: (i) read-only data structures in long-running applications, (ii) data of long-running applications that are resident in the DRAM cache for a very long time, and (iii) filesystem data when PCM is used as persistent storage. Our simulations are focused on estimating error rates for the first 10^7 such long-term writes, *i.e.*, we always assume that drift-based errors show up before the next write to that line. For each such write, we estimate if a given error tolerance mechanism would have allowed an error to escape correction.

To get an estimate of the impact of our proposed policies on cell endurance, we report the total number of scrub or refresh related writes to a line. The eventual impact of this on lifetime will be a function of the percentage of long-term writes in a workload.

Energy per read and write is estimated based on the data of Xu et al. [41] for each cell state and is summarized in Table 3. We also quantify the energy cost of the parity-based error detection circuit (described shortly). Our energy estimates only include the energy consumed within PCM chips for scrub operations; we do not include the energy cost of data transfers on the memory channel or the cost of ECC encode/decode at the memory controller.

We compare our proposed mechanisms against two baseline techniques described in Section 2.6. The first technique is similar to DRAM refresh that blindly reads and re-writes lines at a specified rate. The second is a basic scrub mechanism (ECC-1) similar to that used in DRAM, where reads are issued at the specified rate and a SECDED code is used to correct any observed single-bit errors.

5 Results

5.1 Impact of Strong ECC Codes

We first show the impact of strong ECC codes on drift-induced error rates. We assume a scrub mechanism that maintains an *ECC-E* code per line, issues LARDDs at a given frequency, and re-writes the line if E errors are detected. Even with strong ECC support, there are occasions when errors go undetected or cannot be corrected; this happens when multiple errors happen between successive LARDDs. For a given LARDD rate, the error rate as a function of E depends on how errors are clustered over time. Since the values of R_0 and α follow a normal distribution, there are few outlier worst-case cells that are separated significantly in their drift times (roughly by a factor of two). Hence, we observe that as E increases, the error rate drops sharply (as the likely time gap between the E^{th} and $(E+1)^{th}$ error goes up) as shown in Figure 5. Also, for a given E, the uncorrectable error rate increases sharply as the LARDD interval is increased. With an 8-second interval between LARDDs, the ECC-8 scheme yields significantly fewer uncorrectable errors (228) than the baseline ECC-1 scrub mechanism (9,016). However, the error rate with ECC-8 is still very high. Figure 5 shows that while multi-bit error tolerant codes can help the basic scrub mechanism, there is still much room for improvement.

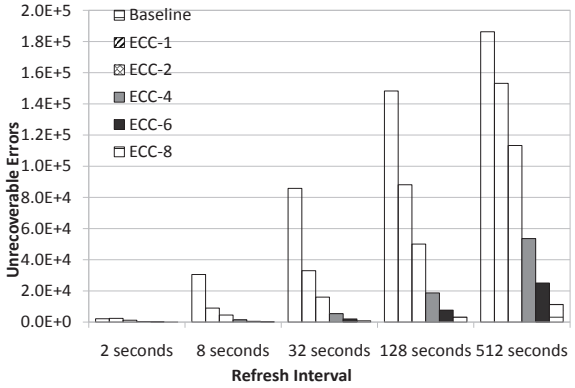


Figure 5: Uncorrectable errors for 10^7 long-term writes for the proposed scrub mechanism as a function of E and LARDD interval. The ECC-1 model represents a DRAM-like scrub mechanism.

5.2 Impact of Headroom and Gradual

Headroom. To reduce the likelihood of errors slipping through, the Headroom scheme triggers a re-write operation even before the E^{th} error happens. In our nomenclature,

Policy	Reduction in uncorrectable error rate	Reduction in scrub-related writes	Reduction in scrub energy
ECC-8	92%	173.7×	52.4%
ECC-8-headroom-3	99.6%	18.8×	35.4%
ECC-8-headroom-3-gradual-1	96.5%	24.4×	37.8%

Table 4: Summary of improvements with proposed scrub mechanisms for a 512 second LARDD interval, relative to the ECC-1 baseline.

a *headroom-h* scheme triggers a re-write after the LARDD detects at least $E - h$ errors. An uncorrectable error with a *headroom-3* scheme now happens if errors $E - 3$, $E - 2$, $E-1$, E, and $E+1$, all happen within one LARDD interval, an event with significantly lower probability.

Figure 4a shows that *headroom-h* helps reduce error rate, and also helps reduce LARDD frequency for a given error rate. The *ECC-8-headroom-3* scheme is able to yield only 647 errors at a LARDD rate of 512 seconds (roughly one cache line read every 500 cycles). In comparison, the *ECC-1* baseline yields 10^5 errors at that LARDD rate.

However, since the policy panics and re-writes sooner than the no-headroom policy, the number of scrub-related writes increases as shown in Figure 4b. Thus, there is a clear and significant trade-off between endurance (hard error rates) and uncorrectable soft error rates when designing scrub policies for MLC PCM. The *ECC-8* scheme issues 173× fewer scrub-related writes than the *ECC-1* scheme; the *ECC-8-headroom-3* scheme gives up some of this advantage and is only 19× better than the *ECC-1* baseline for scrub related writes. While some recent papers have shown that wear-leveling and other optimizations can increase PCM lifetimes to over 10 years, it is important to realize that some of that lifetime may be lost to drift-induced soft-error tolerance techniques. Hence, another conclusion of our study is that hard error avoidance and tolerance in PCM will be even more important in the future.

Using the energy estimates in Table 3, we compare energy consumption of various headroom schemes in Figure 4c. While the *ECC-E* scheme is more energy-efficient than the *ECC-1* baseline, an *ECC-E-headroom-h* scheme consumes more energy compared to its *ECC-E* counterpart because of the higher re-write frequency. So the use of headroom introduces a trade-off involving uncorrectable soft error rates, hard error rates, and energy.

Headroom-Gradual. As described in Section 3.3, to reduce the energy overhead of LARDD operations, we employ adaptive LARDD rates. If a line has fewer than $E - h - g$ errors, we halve the LARDD frequency. Our analysis shows that such a scheme reduces the number of LARDDs, but leads to a higher error rate. We also observe that the error rates are manageable only when it is combined with a headroom scheme. For all of our evaluations, we only show results for $g = 1$.

Figure 4 shows that at a 512-second LARDD interval, *headroom-gradual* leads to an 8× higher error rate than the *headroom* scheme, but reduces the number of LARDDs by 25.5%, thus consuming 5% less scrub-related energy.

For a given ECC support, we can make two significant conclusions from Figure 4 - (i) increased headroom leads to fewer errors, but decreases lifetime and increases energy consumption, and (ii) a headroom-gradual scheme leads to a higher uncorrectable error rate as compared to a pure headroom scheme, but decreases energy consumption, and has a very small (negative) effect on lifetimes. The quantitative improvements for the proposed schemes, relative to the *ECC-1* baseline are summarized in Table 4.

Policy	2 s	8 s	32 s	128 s	512 s
LARDD-ECC-8	45	84	632	2,043	4,153
LARDD-ECC-8-headroom-3	0	0	15	1,491	2,441
LARDD-ECC-8-headroom-5	0	0	0	6	486
LARDD-parity-4/8	0	19	29	871	2,647

Table 5: Uncorrectable errors with parity and ECC schemes. A parity-based scheme that panics at 4 parity errors has similar uncorrectable error rates as a scheme that panics at 5 actual errors (ECC-8-headroom-3).

5.3 Impact of Parity-Based Approximate Error Detection

Strong ECC codes incur a non-trivial overhead in terms of circuit complexity, latency, and energy. Since any off-chip storage array is likely to have some form of error protection, incurring this overhead on every request out of the LLC is to be expected. However, incurring this cost on every LARDD will likely be prohibitively expensive.

We first describe a basic baseline ECC circuit. We use a BCH scheme similar to the one used in [27] to correct up to 8 errors. The ECC circuitry is split into two parts, the first used to detect and the second used to correct errors. Since error detection is used on every LARDD, the error detect circuitry is optimized to complete in one cycle. To minimize area and leakage power overheads, the encode and decode circuitry are shared. Also, since error correction is only performed when an error is detected, the correction circuitry is power gated when not in use to reduce leakage power. It uses Berlekamp-Massey algorithm to calculate the error location polynomial and Chien search to evaluate error position. From [27] and [7], we estimate the energy used by an ECC circuit capable of correcting 8 errors for a 512 bit line to be 192 pJ. This is in addition to the 2560 pJ incurred on average for every PCM line read [41].

To alleviate this cost, and make it possible to perform error detection on the PCM chip itself, we introduce the parity-based approximate error detection circuit described in Section 3.2. Our error detection circuit can be approximate because we provide headroom. Assuming that an ECC-8 code exists for a 256-cell line, a full re-write is triggered as soon as 4 of the 8 32-cell words flag a parity error.

Our parity scheme consists of simple combinational logic per bit to detect the drift prone 10 states and signal a 1. Each PCM line consists of 256 PCM cells and this is divided into 8 sets of 32 cells each. Parity is calculated for each of these sets by XORing all the bits of the detector circuit. The parity calculated for every 32 cell set is compared with a parity bit stored during the write. If more than 4 sets

have mismatches, a Panic signal is raised, which triggers a full re-write. The energy (1.47 pJ) used by this parity circuit is estimated by synthesizing this circuit using 65 nm libraries in Synopsys Design Compiler assuming worst case switching. The parity optimization makes a LARDD even lighter, reducing the cost of a line read from 2752 pJ to 2561 pJ (a 7% reduction). Even more importantly, the area of our parity-based circuit ($1,656\mu^2$ at 65 nm) is 11 times less than that of the baseline ECC circuit [27].

We note that the overhead of proposed LARDD schemes can be further reduced by using 3D-stacked memory chips. The logic interface dies on such 3D stacks can accommodate circuits for error detection and adaptive LARDD rates, without impacting PCM cell density. This is consistent with a recent trend [38] to move more functionality to the logic interface dies on 3D stacks to save bandwidth.

We carry out simulations on real cache line data so that the distribution of drifted cells across the 8 32-cell words is cognizant of how states are distributed across real cache lines. In Table 5, we compare the error rates for our initial LARDD-ECC-8 policies and the LARDD-parity-4/8 policy and show that the error rates are not much more than the headroom-3 scheme.

5.4 Adaptive Scrub Rate Case Study

Drift times and error rates are a function of many factors. We expect that the OS will track error rates over time and occasionally increase or decrease LARDD rates to best balance the error rate and LARDD overhead. As a case study, we evaluate such an approach for an important scenario: the emergence of hard errors over the lifetime of a PCM chip.

We simulate a model where a line may have a number of hard errors (but less than E hard errors) with a probability that is linearly proportional to the number of writes already seen by the simulation. In other words, there are few hard errors early in the simulated lifetime of the line, but the number of hard errors gets closer to E by the end of the simulated lifetime. A line is considered defunct if it has E hard errors. As the hard errors increase, we effectively can handle fewer soft errors and the error rates start to increase.

In our experiments, we assume that E is 8, h is 3, and HE is 2. Recall from the discussion in Section 3.3 that a faster LARDD rate is only used if a line has more than HE hard errors. We start with a LARDD interval of 16 seconds and the interval is halved if more than 5 uncorrectable errors are detected in the previous epoch (10^5 long-term writes). In a baseline with zero hard errors, a total of 4 uncorrectable errors were detected in the entire simulation. Once hard errors were introduced as described above, the number of uncorrectable errors grows to 149,000. With our dynamic policy, the LARDD interval is progressively reduced in the latter half of the simulation to 64 ms, while keeping the number of uncorrectable errors below 1,000. If a longer LARDD interval is used for lines with fewer than HE errors, LARDD energy is reduced by 22%, but the number of errors increases by 18%. We also observed similar trends when we assumed other hard error proliferation rates. The OS would have to consider the above trade-offs when setting the parameters of such adaptive policies.

5.5 Comparison Against Device-Level Solutions

In this section, we consider two of the most effective device-level solutions to delay the onset of drift-based errors. We show that architectural solutions are superior to these device-level alternatives in almost every regard.

The first device-level solution is non-uniform banding. Instead of splitting the available resistance range equally among all resistance states, the range is split in a non-uniform manner so that drift-prone states have more widely spaced boundary resistance thresholds. This device-level solution does not incur higher implementation complexity, but is less effective as the number of levels in a cell is increased. In our experiments, the boundary resistance threshold for the two drift-prone states is widened by 10%, while that for the non-drift-prone states is reduced by 10%.

The second device-level solution attempts more precise writes. In most experiments in this paper, we assume that the programmed resistance range allows cells with resistance at most 2.75σ from the mean. Instead, we can implement more precise writes by halving the programmed resistance range to allow cells with resistance at most 1.375σ from the mean. This means that a cell will have to drift further to cross the boundary threshold (which is situated at a resistance 3σ from the mean), allowing the baseline device to get away with a much longer refresh/LARDD interval. We'll shortly compare the error rates with these competing approaches. The cost of this device-level solution is summarized next.

Currently, not much work exists on the viability of precise writes in PCM. Based on work in [21, 22, 25, 11], we project the following impacts on latency, energy, and endurance. The iterative write process subjects a cell to multiple current pulses and after every pulse, we check to see if the resistance is within the programmed resistance range. If we provide a pulse that causes a big jump in resistance, it may be possible to miss the acceptable range altogether. So the typical jump needs to be a little bit smaller than the width of the acceptable resistance range. Thus, with a high likelihood, the final pulse will place the cell within the acceptable range. However, as already discussed, because of variations, the resistance follows a normal distribution within that range.

If a write must be made more precise (narrower acceptable range), each pulse will have to cause a smaller jump in resistance. So the write will involve many short resistance jumps. Assuming a fixed pulse width (say, 5 ns), the cell can be either programmed with many low-amplitude pulses, or a few high-amplitude pulses [21, 22]. The former provides the higher precision at a latency penalty. Typically, if the programmed resistance range is being halved, we will need twice as many pulses, each causing half the resistance jump. Fortunately, the higher precision write also consumes

Policy/Improvement	Baseline ECC-1 (2.75σ)	ECC-1 + Precise Write (1.375σ)	ECC-1 + Non-uniform banding	ECC-8 (2.75σ)	ECC-8-headroom-3 (2.75σ)
Uncorrectable errors	153,164	26,610	70,935	11,293	674

Table 6: Uncorrectable errors for a LARDD interval time of 512 seconds. We show a baseline scrub mechanism (ECC-1) combined with device-level solutions (precise writes and non-uniform banding). These are compared against architectural solutions that do not change programmed resistance ranges or boundary resistance thresholds.

less overall energy (since energy is proportional to I^2). The work of Lin et al. [21] shows an example where a write with twice the precision consumes four times less energy and twice the latency. To measure the impact of higher write latency, we ran detailed Simics simulations with a state-of-the-art memory scheduling model that buffers writes and drains them once they reach a high water mark. We observed that a 200-cycle increase in write latency can reduce IPC by 5.2%. The impact of write precision on endurance is not yet exactly known (to the best of our knowledge). The work of Goux et al. [9, 8] shows that endurance is very strongly influenced by the RESET pulse at the start of the iterative write process. Subsequent pulses have a smaller impact, as they typically do not result in melting the active volume. Endurance is also a linear function of the number of pulses. Thus, we can conclude that a write that is twice as precise and needs twice as many low-amplitude pulses will have an endurance that may be at most $2\times$ worse. Thus, the precise write incurs a penalty in terms of latency and endurance.

We see how the competing approaches compare in terms of error rates. Table 6 considers a baseline PCM device (with the 2.75σ programmed resistance range), a PCM device with a more precise 1.375σ programmed resistance range, and a PCM device with non-uniform bands (10% wider boundary thresholds for drift-prone states). All three models have a basic DRAM-like scrubbing mechanism (ECC-1) with a LARDD interval of 512 seconds. We see that precise writes and non-uniform banding are able to bring the error rate down from 153.2K in the baseline to 26.6K and 70.9K, respectively. Instead, if the baseline was augmented with architectural solutions (scrub with ECC-8 and scrub with ECC-8-headroom-3), while keeping the programmed resistance range fixed at 2.75σ, we are able to bring the errors down to 11.3K and 674, respectively. The architectural solutions are also superior because they delay the issue of writes, thus improving energy and endurance. The precise write approach also degrades performance because it increases write latency. Thus, if we had to pick a single approach to mitigate drift-based errors, an architectural solution with LARDDs, multi-bit error correction, and headroom appears most desireable. The architectural solution can be combined with device-level solutions to achieve even higher gains.

6 Related Work

Recently, many architecture-level solutions have been proposed to address PCM challenges [32, 31, 20, 47, 30, 46]. Only one targets resistance drift in MLC PCM [46]. In the eDRAM space, strong BCH based ECC methods have been used for a variety of purposes, most recently in *Hi-ECC* [40]. Hi-ECC uses 5EC6ED codes to reduce refresh power in eDRAM, along with other optimizations.

Mitigating Resistance Drift A few device-level solutions for drift mitigation have been proposed recently [18, 34, 16, 43, 42, 26]. These techniques are in early stages of design and range from corrective voltage pulses to dopant materials to reference cells and modulation coding. Zhang and Li [46] propose the Helmet architecture that uses a number of methods to counter drift: encoding mechanisms to reduce the occurrence of drift-prone states, switching from MLC to SLC mode, and modifying the OS for temperature-aware page allocation. Our work looks at a completely distinct set of optimizations that are focused on scrub-based tolerance of errors exposed by drift. We anticipate that some of the Helmet optimizations can be combined with our scrub policies to further reduce error rates and overheads.

Hard Error Correction Ipek at al. [14] propose a dynamically replicated memory architecture that allows for continued operation through graceful degradation of PCM cells. Schechter et al. [35] attempt to deal with hard errors in resistive memories by tracking worn out cells and storing a pointer to these cells along with the cache line. Seong et al. [37] propose SAFER, a technique to recover from the "stuck-at" hard errors in PCM based on the observation that even if a bit is *stuck* at a particular value, that value is still readable with the right encoding (inversion or not). Yoon et al. [45] propose FREE-p, where a failed data block stores a pointer to another data block that is used as a substitute. These solutions can only handle cells that are known to have failed and are not effective for soft error tolerance.

In Flash, strong error correction codes and adaptive programming voltages are employed, with no background scrub mechanism. Apart from the Flash coding strategies, to the best of out knowledge, we are not aware of other work that combines lightweight detection codes with stronger correction codes. The lightweight LPDC error detection strategy for Flash is more sophisticated and longer latency than parity-based schemes proposed in this work.

7 Conclusions

In this work, we show that drift-based multi-bit errors in PCM MLC devices are a significant problem. Traditional scrub mechanisms that have worked for DRAM are highly inadequate for MLC PCM. We show how scrub mechanisms can be extended with stronger ECC codes, headroom schemes, and adaptive scrub rates. These extensions have an impact on uncorrectable soft error rates, lifetime, and energy. Our combined policy (ECC-8-headroom-3-gradual)

yields a 96.5% error rate reduction, a 24.4 $\times$ reduction in scrub-related writes, and a 37.8% reduction in scrub energy, relative to a DRAM-like (ECC-1) scrub mechanism. The impact of this improvement on overall device lifetime and FIT rates would depend on a number of factors, including the frequency of long-term writes in the workload. We show that these architectural techniques are much more effective than device-level approaches.

References

[1] Multilevel Phase Change Memories. `http://www.thinfilmproducts.umicore.com/feature.asp?page=art6`.

[2] Research Needs for Memory Technologies. Technical report, Semiconductor Research Corporation (SRC), 2011. www.src.org/program/grc/ds/research-needs/2011/memory.pdf.

[3] C. Benia et al. The PARSEC Benchmark Suite: Characterization and Architectural Implications. Technical report, Princeton University, 2008.

[4] M. Boniardi, D. Ielmini, S. Lavizzari, A. Lacaita, A. Redaelli, and A. Pirovano. Statistical and scaling behavior of structural relaxation effects in phase-change memory (PCM) devices. 2009.

[5] G. W. Burr, M. J. Breitwisch, M. Franceschini, D. Garetto, K. Gopalakrishnan, B. Jackson, B. Kurdi, C. Lam, L. A. Lastras, A. Padilla, B. Rajendran, S. Raoux, and R. S. Shenoy. Phase Change Memory Technology, 2010. http://arxiv.org/abs/1001.1164v1.

[6] S. Cho and H. Lee. Flip-N-Write: A Simple Deterministic Technique to Improve PRAM Write Performance, Energy, and Endurance. In *Proceedings of MICRO*, 2009.

[7] W. Gao and S. Simmons. A study on the VLSI implementation of ECC for embedded DRAM. In *Proceedings of IEEE CCECE*, 2003.

[8] L. Goux, T. Gille, D. Castro, G. Hurkx, J. Lisoni, R. Delhougne, D. Gravesteijn, K. De Meyer, K. Attenborough, and D. Wouters. Transient Characteristics of the Reset Programming of a Phase-Change Line Cell and the Effect of the Reset Parameters on the Obtained State. *Electron Devices, IEEE Transactions on*, 56(7), 2009.

[9] L. Goux, D. Tio Castro, G. Hurkx, J. Lisoni, R. Delhougne, D. Gravesteijn, K. Attenborough, and D. Wouters. Degradation of the Reset Switching During Endurance Testing of a Phase-Change Line Cell. *Electron Devices, IEEE Transactions on*, 56(2), 2009.

[10] R. Hamming. Error detecting and error correcting codes. *Bell System Technical Journal*, 1950.

[11] Y. Hwang, C. Um, J. Lee, C. Wei, H. Oh, G. Jeong, H. Jeong, C. Kim, and C. Chung. MLC PRAM with SLC write-speed and robust read scheme. In *Proceedings of Symposium on VLSI Technology*, 2010.

[12] D. Ielmini, M. Boniardi, A. Lacaita, A. Redaelli, and A. Pirovano. Unified mechanisms for structural relaxation and crystallization in phase-change memory devices. *Microelectronic Engineering*, 86(7-9):1942 – 1945, 2009. INFOS 2009.

[13] D. Ielmini, D. Sharma, S. Lavizzari, and A. Lacaita. Reliability Impact of Chalcogenide-Structure Relaxation in Phase-Change Memory (PCM) Cells ;Part I: Experimental Study. *Electron Devices, IEEE Transactions on*, 56(5), may 2009.

[14] E. Ipek, J. Condit, E. Nightingale, D. Burger, and T. Moscibroda. Dynamically Replicated Memory : Building Reliable Systems from nanoscale Resistive Memories. In *Proceedings of ASPLOS*, 2010.

[15] B. Jacob, S. W. Ng, and D. T. Wang. *Memory Systems - Cache, DRAM, Disk*. Elsevier, 2008.

[16] C.-W. Jeong, D.-H. kang, H.-J. Kim, S.-P. Ko, and D.-W. Lim. Multiple Level Cell Phase-Change Memory Devices Having Controlled Resistance Drift Parameter, Memory Systems Employing Such Devices and Methods of Reading Memory Devices, 2008. United States Patent Application, Number US 2008/0316804 A1 .

[17] S. Kang, W. Y. Cho, B.-H. Cho, K.-J. Lee, C.-S. Lee, H.-R. Oh, B.-G. Choi, Q. Wang, H.-J. Kim, M.-H. Park, Y. H. Ro, S. Kim, C.-D. Ha, K.-S. Kim, Y.-R. Kim, D.-E. Kim, C.-K. Kwak, H.-G. Byun, G. Jeong, H. Jeong, K. Kim, and Y. Shin. A 0.1- μm 1.8-V 256-Mb Phase-Change Random Access Memory (PRAM) With 66-MHz Synchronous Burst-Read Operation. *Solid-State Circuits, IEEE Journal of*, 42(1), jan. 2007.

[18] A. Kostylev and W. Czubatyj. Method of Eliminating Drift in Phase-Change Memory, 2004. United States Patent Application, Number US 2004/0228159 A1 .

[19] S. K. Lai. Flash memories: Successes and challenges. *IBM Journal of Research and Development*, 52(4.5), 2008.

[20] B. Lee, E. Ipek, O. Mutlu, and D. Burger. Architecting Phase Change Memory as a Scalable DRAM Alternative. In *Proceedings of ISCA*, 2009.

[21] J.-T. Lin, Y.-B. Liao, M.-H. Chiang, I.-H. Chiu, C.-L. Lin, W.-C. Hsu, P.-C. Chiang, S.-S. Sheu, Y.-Y. Hsu, W.-H. Liu, K.-L. Su, M.-J. Kao, and M.-J. Tsai. Design optimization in write speed of multi-level cell application for phase change memory. In *Intl. Conf. of Electron Devices and Solid-State Circuits*, 2009.

[22] J.-T. Lin, Y.-B. Liao, M.-H. Chiang, and W.-C. Hsu. Operation of Multi-Level Phase Change Memory Using Various Programming Techniques. In *Proceedings of ICICDT*, 2009.

[23] W. Mueller, G. Aichmayr, W. Bergner, E. Erben, T. Hecht, C. Kapteyn, A. Kersch, S. Kudelka, F. Lau, J. Luetzen, A. Orth, J. Nuetzel, T. Schloesser, A. Scholz, U. Schroeder, A. Sieck, A. Spitzer, M. Strasser, P.-F. Wang, S. Wege, and R. Weis. Challenges for the DRAM cell scaling to 40nm. 2005.

[24] T. Nirschl, J. Phipp, T. Happ, G. Burr, B. Rajendran, M.-H. Lee, A. Schrott, M. Yang, M. Breitwisch, C.-F. Chen, E. Joseph, M. Lamorey, R. Cheek, S.-H. Chen, S. Zaidi, S. Raoux, Y. Chen, Y. Zhu, R. Bergmann, H.-L. Lung, and C. Lam. Write Strategies for 2 and 4-bit Multi-Level Phase-Change Memory. 2007.

[25] A. Pantazi, A. Sebastian, N. Papandreou, M. J. Breitwisch, C. Lam, H. Pozidis, and E. Eleftheriou. Multilevel Phase-Change Memory Modeling and Experimental Characterization. In *Proceedings of EPCOS*, 2009.

[26] N. Papandreou, H. Pozidis, T. Mittelholzer, G. Close, M. Breitwisch, C. Lam, and E. Eleftheriou. Drift-tolerant multilevel phase-change memory. In *Proceedings of International Memory Workshop (IMW)*, 2011.

[27] S. Paul, F. Cai, X. Zhang, and S. Bhunia. Reliability-Driven ECC Allocation for Multiple Bit Error Resilience in Processor Cache. *IEEE Transactions on Computers*, 99, 2010.

[28] A. Pirovano, A. Lacaita, F. Pellizzer, S. Kostylev, A. Benvenuti, and R. Bez. Low-field amorphous state resistance and threshold voltage drift in chalcogenide materials. *IEEE Transactions on Electron Devices*, 51(5), may 2004.

[29] M. Qureshi, M. Franceschini, and L. Lastras. Improving Read Performance of Phase Change Memories via Write Cancellation and Write Pausing. In *Proceedings of HPCA*, 2010.

[30] M. Qureshi, M. Franceschini, L. Lastras-Montano, and J. Karidis. Morphable Memory System: A Robust Architecture for Exploiting Multi-Level Phase Change Memory. In *Proceedings of ISCA*, 2010.

[31] M. Qureshi, J. Karidis, M. Franceschini, V. Srinivasan, L. Lastras, and B. Abali. Enhancing Lifetime and Security of PCM-Based Main Memory with Start-Gap Wear Leveling. In *Proceedings of MICRO*, 2009.

[32] M. Qureshi, V. Srinivasan, and J. Rivers. Scalable High Performance Main Memory System Using Phase-Change Memory Technology. In *Proceedings of ISCA*, 2009.

[33] A. Redaelli, A. Pirovano, A. Locatelli, and F. Pellizzer. Numerical Implementation of Low Field Resistance Drift for Phase Change Memory Simulations. In *Non-Volatile Semiconductor Memory Workshop*, 2008.

[34] S. Kostylev and T. Lowrey. Drift of Programmed Resistance In Electrical Phase Change Memory Devices, 2008.

[35] S. Schechter, G. Loh, K. Strauss, and D. Burger. Use ECP, not ECC, for hard Failures in Resistive Memories. In *Proceedings of ISCA*, 2010.

[36] B. Schroeder, E. Pinheiro, and W. Weber. DRAM Errors in the Wild: A Large-Scale Field Study. In *Proceedings of SIGMETRICS*, 2009.

[37] N. Seong, D. Woo, V. Srinivasan, J. Rivers, and H. Lee. SAFER: Stuck-At-Fault Error Recovery for Memories. In *Proceedings of MICRO*, 2010.

[38] A. Udipi, N. Muralimanohar, R. Balasubramonian, A. Davis, and N. Jouppi. Combining Memory and a Controller with Photonics through 3D-Stacking to Enable Scalable and Energy-Efficient Systems. In *Proceedings of ISCA*, 2011.

[39] A. N. Udipi et al. Rethinking DRAM Design and Organization for Energy-Constrained Multi-Cores. In *Proceedings of ISCA*, 2010.

[40] C. Wilkerson, A. Alameldeen, Z. Chishti, W. Wu, D. Somasekhar, and S.-L. Lu. Reducing Cache Power with Low-Cost, Multi-Bit Error Correcting Codes. In *Proceedings of ISCA*, 2010.

[41] W. Xu, J. Liu, and T. Zhang. Data Manipulation Techniques to Reduce Phase Change Memory Write Energy. In *Proceedings of ISLPED*, 2009.

[42] W. Xu and T. Zhang. A time-aware fault tolerance scheme to improve reliability of multilevel phase-change memory in presence of significant resistance drift. *IEEE Transactions on VLSI Systems*, PP(99), 2010.

[43] W. Xu and T. Zhang. Using Time-Aware Memory Sensing to Address Resistance Drift Issue in Multi-Level Phase Change Memory. In *Proceedings of ISQED*, 2010.

[44] D. Yoon and M. Erez. Virtualized and Flexible ECC for Main Memory. In *Proceedings of ASPLOS*, 2010.

[45] D.-H. Yoon, N. Muralimanohar, J. Chang, P. Ranganathan, N. Jouppi, and M. Erez. FREE-p: Protecting Non-Volatile Memory against both Hard and Soft Errors. In *Proceedings of HPCA*, 2011.

[46] W. Zhang and T. Li. Helmet: A Resistance Drift Resilient Architecture for Multi-level Cell Phase Change Memory System. In *Proceedings of DSN*, 2011.

[47] P. Zhou, B. Zhao, J. Yang, and Y. Zhang. A Durable and Energy Efficient Main Memory Using Phase Change Memory Technology. In *Proceedings of ISCA*, 2009.