Tri-Level-Cell Phase Change Memory: Toward an Efficient and Reliable Memory System

Nak Hee Seong Sungkap Yeo Hsien-Hsin S. Lee

School of Electrical and Computer Engineering
Georgia Institute of Technology
Atlanta, GA 30332
nakhee.seong@gmail.com {sungkap, leehs}@gatech.edu

ABSTRACT

There are several emerging memory technologies looming on the horizon to compensate the physical scaling challenges of DRAM. Phase change memory (PCM) is one such candidate proposed for being part of the main memory in computing systems. One salient feature of PCM is its multi-level-cell (MLC) property, which can be used to multiply the memory capacity at the cell level. However, due to the nature of PCM that the value written to the cell can drift over time, PCM is prone to a unique type of soft errors, posing a great challenge for their practical deployment. This paper first quantitatively studied the current art for MLC PCM in dealing with the resistance drift problem and showed that the previously proposed techniques such as scrubbing or error correction mechanisms have significant reliability challenges to overcome. We then propose tri-level-cell PCM and demonstrate its ability to achieving $10^5\times$ lower soft error rate than four-level-cell PCM and $1.33\times$ higher information density than single-level-cell PCM. According to our findings, the tri-level-cell PCM shows 36.4% performance improvement over the four-level-cell PCM while achieving the soft error rate of DRAM.

1. INTRODUCTION

Phase change memory (PCM) is a promising alternative memory technology for future computing systems. Based on chalcogenide compound made of Ge, Sb, and Te (GST), the value of stored data is represented with its material state indicated by its current resistance level. When a PCM cell is heated up to a temperature over the melting point and cooled down within several tens of nano-seconds, the cell enters a high resistive amorphous state. In contrast, the PCM cell enters a low resistive crystalline state when it is exposed to a temperature lower than the melting point and cooled down slowly. The resistance of a PCM cell is known to be around 10^3 ohms in the crystalline state and around 10^6 ohms in the amorphous state. Moreover, when we alter the temperature and the duration induced to the PCM cell, it is found that the resistance can be anywhere in-between these two states. Multi-level-cell (MLC) PCM exploits these intermediate states in-between the crystalline and amorphous states to store more data per cell.

Although the MLC PCM increases information density, this technique requires a finer-grain control over the resistance of a cell. To locate the resistance of a cell within a predefined range, the MLC PCM requires an iterative-writing mechanism, which reads the resistance immediately after a write to check whether the cell needs to be rewritten or not. This iterative-writing degrades the write latency. Recent studies showed that the write latency of a four-level-cell is about 4x $\sim$ 8x slower than that of a single-level-cell (SLC) PCM [13].

Besides the performance issue, a far more critical problem of making MLC PCM practical is its reliability concern caused by the resistance drift. The resistance drift is the phenomenon that the resistance of a PCM cell increases over time. Such drifting causes a unique type of soft errors that is different from soft errors of DRAM. In DRAM, soft errors are produced by particle strikes, and the error rate is independent of the stored value. In contrast, MLC PCM cell array is impervious to particle strikes but experiences soft errors caused by resistance drift. Although reasons for these two types of soft errors are different, we generally call both soft errors because (1) they are not hard errors, which indicate permanent hardware failure, and (2) they share solutions such as scrubbing or error correction mechanisms.

Resistance drift was not a problem in SLC PCM because the rate of such drift is proportional to the initial resistance of the cell and is nearly zero for the crystalline state. However, the resistance of MLC PCM cells at the intermediate states could cross their state boundary and lead to undesirable errors due to the state changes. This new type of soft errors caused by resistance drift, if left unaddressed, will make MLC PCM unreliable. In this paper, we mathematically formulate the drift-induced soft error rates of MLC PCM. With this analytical model, we evaluate the previously proposed ideas for reducing soft errors and show that four-level-cell PCM still cannot deliver reliability that matches DRAM reliability — it requires other architectural mechanisms. We then propose tri-level-cell (3LC) PCM and show that the 3LC PCM can achieve the soft error rate of DRAM and the performance of SLC PCM.

2. BACKGROUND AND MOTIVATION

A multi-level PCM cell can store more than one bit by defining the intermediate states between set and reset states [11]. The resistance of a PCM cell is as low as 10^3 ohms in the set state and 10^6 ohms in the reset state. By further exploiting the resistance difference, a PCM cell can have two or even more intermediate states in addition to set and reset to increase data density per cell. For example, four-level-cell (4LC) PCM stores two bits per cell by exploiting two additional intermediate states, while eight-level-cell

(8LC) PCM stores three bits per cell with six more intermediate states. Such multi-level-cell (MLC) PCM requires the following operations to function correctly. Firstly, an MLC PCM cell needs iterative write-and-verify steps to verify its written value. When the resistance fails to fall into a predefined range, a PCM chip needs to repeat the write-and-verify step. This iterative writing process takes up to eight times longer than a typical write in single-level-cell (SLC) PCM [13]. Secondly, when the resistance of an MLC PCM cell is drifted and crosses the storage level boundary, a soft error (*i.e.,* bit flipping) occurs and needs to be recovered by error correction mechanism, which can be costly.

Unfortunately, the soft error rate (SER) due to resistance drift in MLC PCM is fairly high. With a detailed model of resistance drift [20], we calculate the probability of the SER of an MLC PCM cell. As we will show in Section 4, the SER increases over time because the resistance of the MLC PCM cell increases over time. In other words, the chance of crossing the storage level boundary increases over time along with the resistance. More importantly, Appendix A shows that the SER of MLC PCM is significantly higher than that of DRAM. To address such shortcomings, Xu *et al.* [20] proposed a time-aware error correction scheme, which employs extra cells for storing predefined reference resistance values. The reference cells are adjusted to the predefined values whenever the other cells in its corresponding data block are written. When reading the data block, the resistance of the reference cells are used to compensate the drifted resistance in other cells. By using such a technique, the SER (called raw bit error rate in [20]) could be reduced from $10^{-3} \sim 10^{-1}$ to $7 \times 10^{-4} \sim 10^{-2}$. On the other hand, Awasthi *et al.* proposed an efficient scrub mechanism for MLC PCM [1]. The mechanism effectively reduced 99.6% of uncorrectable errors; however, the lowest possible SER for long-term writes[1] of 4LC PCM was 6.74×10^{-5}.

DRAM also experiences soft errors caused by particle strikes. Its SER is known to be an average of $25,000 \sim 75,000$ FIT (failures in time per billion hours of operation) per Mbit, *i.e.,* $25 \times 10^{-12} \sim 75 \times 10^{-12}$ per bit-hour [17]. For example, 16GB of DRAM is expected to have 3.43 to 10.31 soft errors every hour. In contrast, 4LC PCM with SER of 6.74×10^{-5} (the lowest SER for long-term writes in [1]) is expected to incur 9.26×10^6 errors, near 10^6 times more errors than DRAM. Moreover, in this comparison, an eight-bit correction BCH ECC is assumed [1] whereas no ECC was assumed in DRAM. Nonetheless, 4LC PCM shows several orders of magnitude higher SER than DRAM even with sophisticated ECC support.

Many architectural and device-level techniques were proposed to alleviate the downsides of this 4LC PCM reliability [9, 20, 1]; however, we argue that 4LC PCM still requires additional architectural solutions to be a practical device for main memory. For one, reference-cell based or time-varying threshold methods introduce the following problems. If we adopt redundant PCM cells for storing reference values to compensate the increase in resistance, a block of PCM cells must share the redundant cells to mitigate the capacity overhead [20]. As such, any writing operation should read and rewrite the entire block of cells including reference cells. This strategy triggers more writes to the cells, reduces their lifespan, consumes more power, and degrades performance. Secondly, if the scrub mechanism is used for reducing soft errors [1], the memory controller will spend more time in scrubbing than DRAM, which degrades the overall performance of the memory subsystem. Lastly, DRAM-style self-refresh cannot directly be applied to MLC PCM. Refreshing a PCM cell does not consume off-chip bandwidth nor utilize memory controller; however, such refresh cannot benefit from error correcting codes. In this case, cells must be refreshed while they still hold the correct information. Therefore, refresh interval should be as short as hundreds of milliseconds. Such frequent refresh introduces new problems: (1) higher chip-level power, (2) slower responsiveness of PCM because writing a 4LC PCM cell takes $1.15\mu s$, a few orders slower than DRAM, and (3) decreased lifespan of PCM cells due to frequent writes. In summary, 4LC PCM not only has a higher SER than DRAM even with sophisticated techniques but also requires extra overheads that have yet to be quantitatively evaluated.

[1]The original paper [1] defined a long-term write as follows. Some PCM cells experience sufficiently high timing gap between writes. These types of writes are called long-term writes.

The motivation of this research stems from these observations. As we will show in later sections, if we reduce the number of storage levels from four to three, a PCM cell shows fewer errors than DRAM, and thus eliminates the need of ECC, reference cells, and scrubbing. We compare our proposed tri-level-cell (3LC) PCM over 4LC PCM throughout this paper to demonstrate that 3LC PCM is a cost-effective solution for putting multi-level cells into practical use.

3. TRI-LEVEL-CELL (3LC) PCM

A straightforward approach toward 3LC PCM is removing the most error-prone state from 4LC PCM. To do so, we first discuss the physical parameters of 4LC PCM. By measuring the resistance drift of reset and set states from iterative experiments, Ielmini *et al.* [7, 8] showed that the drift can be represented by a power-law model as

$$R_{drift}(t) = R \times \{\frac{t}{t_0}\}^{\alpha}. \tag{1}$$

In Equation (1), R and t_0 are normalization constants and α is a drift exponent. Because the main cause of the drift is the structural relaxation of the amorphous state, the drift exponent of the reset state is much larger than that of the set state in the experiments. In other words, the drift exponent will increase as the portion of the amorphous state of a cell increases.

As mentioned earlier, the resistance drift causes soft errors in MLC PCM. To estimate the impact of resistance drift on reliability in MLC PCM, we make the following assumptions for the normalization constants and the drift exponent for storage levels. According to the experiments performed by Nirschl *et al.* [11], the iterative write-and-verify method adjusts the programmed resistance, R, to be located within a desired resistance range. In addition, $\log_{10} R$ follows a normal (Gaussian) distribution. In this paper, we assume that the logarithm of a normalization resistance, $\log_{10} R$ will follow a normal distribution of $N(\mu_R, \sigma_R^2)$. In addition, a desired programmed resistance range for a given state is set to the range within $10^{\mu_R \pm 2.75 \times \sigma_R}$ Ω and the upper and lower sensing boundaries for the state are set to $10^{\mu_R \pm 3 \times \sigma_R}$ Ω. The value of a drift exponent is also assumed to follow a normal distribution of $N(\mu_\alpha, \sigma_\alpha^2)$. The parameters in our drift analysis are based on prior work [1, 20] and summarized in Table 1. Note that Table 1 takes process variation into account by using normal distribution for modeling both the resistance and the rate of the resistance drift. By using such distribution, we assume that PCM cells are not manufactured equally.

A soft error occurs when the resistance of a MLC PCM cell is drifted above the upper boundary of its programmed state. From the state-boundary settings described above, the condition of a soft error can be represented as

$$R_{drift}(t) > 10^{\mu_R + 3 \times \sigma_R}. \tag{2}$$

Table 1: Configuration Variables of Four-Level-Cell (4LC) PCM When $t_0 = 1$ s.

Storage Level	Data	$\log_{10} R$		α	
		μ_R	σ_R	μ_α	σ_α
0	01	3.0	$\frac{1}{6}$	0.001	$0.4 \times \mu_\alpha$
1	11	4.0		0.02	
2	10	5.0		0.06	
3	00	6.0		0.10	

Table 2: Configuration Variables of Tri-Level-Cell (3LC) PCM When $t_0 = 1$ s.

Storage Level	$\log_{10}(R)$		α	
	μ_R	σ_R	μ_α	σ_α
0	3.0	$\frac{1}{6}$	0.001	$0.4 \times \mu_\alpha$
1	4.0		0.02	
2	6.0		0.10	

In other words, Table 1 shows that the target resistance values for the four storage levels are 10^3, 10^4, 10^5, and $10^6\Omega$, respectively, and the sensing boundaries are $10^{3.5}$, $10^{4.5}$, and $10^{5.5}\Omega$. For instance, when the resistance of a cell programmed for storage level 2 drifts and becomes larger than $10^{5.5}$ Ω, the cell is read as the next storage level and generates a soft error.

With assumptions that $\log_{10} R$ and α follow normal distribution in Table 1, we can calculate the probability of soft errors as a function of time. The detailed formula is derived in Appendix A. As we will show in later sections, the most error-prone state in 4LC PCM is the third storage level for the following reasons. For one, the fourth storage level (amorphous state) in the highest resistance range does not generate soft errors. In addition, because α is proportional to R, the third storage level experiences the rapidest resistance drift among all levels. On the other hand, however, if we remove the third storage level, this will not only remove the errors generated by itself but also reduce most of the errors created by the second storage level. For instance, the majority of errors generated by the second storage level occurs on the boundary between the second and the third storage levels, which can be avoided by not using the third storage level. Table 2 shows our design points for 3LC PCM. Note that 3LC PCM is different from 3LC NAND, which commonly refers to three layer cell NAND that stores 3 bits per cell. More specifically, 3LC NAND implements eight different storage levels per cell and do not require binary to ternary conversion, which will be discussed in Section 5. Therefore, our 3LC PCM design is unique in that it introduces a new way of storing binary information on a ternary device.

Given the physical parameters in Table 2, we calculate the SER of 4LC PCM and 3LC PCM. Note that we use the analytical model in Appendix A and present the results in Tables 3 and 4. Table 3 shows the SER of two intermediate storage levels of 4LC PCM as a function of time since they were written while Table 4 shows the SER of the first two storage levels of 3LC PCM. For example, if a 3LC PCM cell is written to the second storage level at $t = 0$, the SER of the cell is 5.93×10^{-14} at $t = 2^{45}$. Note that we mark "too small" in the tables when Mathematica 8.0 outputs zero due to lack of precision. As Table 4 shows, there is no error in 3LC PCM up to 2^{34} seconds or more than 500 years. Because of such low SER, scrubbing will be unnecessary for 3LC PCM in the time range of interest. For the same reason, ECC or other similar techniques can be waived. In summary, the SER of 3LC PCM is even lower than that of DRAM. It does not require scrubbing nor ECC to achieve

Table 3: Probability of Soft Error of Four-Level-Cell (4LC) PCM by Equation (7) in Appendix A

Time (s)	Storage Level 1	Storage Level 2
2	(too small)	5.85E-06%
2^2	1.59E-12%	0.02%
2^3	5.85E-06%	0.12%
2^4	7.45E-04%	0.28%

Table 4: Probability of Soft Error of Tri-Level-Cell (3LC) PCM by Equation (7) in Appendix A

Time (s)	Crystalline State	Intermediate State
$2 \sim 2^{34}$	(too small)	(too small)
2^{35}	2.28E-16%	(too small)
2^{40}	1.59E-14%	(too small)
2^{45}	5.71E-10%	5.93E-14%

the satisfactory level of reliability. To further justify the use of 3LC PCM over 4LC PCM, we quantitatively compare and evaluate these two design options in the subsequent sections.

4. REVISITING FOUR-LEVEL-CELL (4LC) PCM

Given the soft error rates in Table 3, it is clear that 4LC PCM is unusable as main memory without additional architectural mechanisms for reducing SER. Researchers have proposed several drift-tolerant approaches such as error correction schemes [1, 22, 12, 20], data encoding schemes using relative resistance difference [12, 22], a reference cell scheme [6], a time-aware drift estimation mechanism [20], and most recently an efficient scrubbing scheme [1]. Among them, we focus on the most recent work by Awasthi *et al.* [1] that studied an architectural mechanism combining a memory scrubbing scheme with a strong error-correction method for lowering soft error rates of 4LC PCM. However, as we will show, even with the most efficient scrubbing mechanism, the SER of 4LC PCM is still much higher than that of DRAM.

4.1 Estimating Scrubbing Overhead

In this section, we compare the SER of 4LC PCM to that of contemporary DRAM. First, we assume a 16GB PCM main memory with eight banks (*i.e.*, 2GB per bank) using a 256B data block[2] as a basic access unit as assumed in prior literature [19, 18]. According to recent work by Choi *et al.* [3], the read and write latencies in SLC PCM are $120ns$ and $150ns$, respectively. Considering that iterative write-and-verify steps are required for MLC PCM, we assume that scrubbing a cache line takes $1.15\mu s$.

The rationale behind such high scrubbing time, $1.15\mu s$, is the following. PCM scrubbing must rewrite all cells even without errors. For DRAM, scrubbing writes only when error happens; however, PCM scrubbing is different because of the time-dependent error characteristics as discussed in Equation (1). For example, we assume that a PCM cell is written at $t = 0s$ and has 100 seconds of scrubbing period. We also assume that a memory controller cannot find an error from this cell in the first scrubbing attempt at $t = 100s$. In other words, even though the resistance became $R_{drift}(t) = R \times \{\frac{100s}{t_0}\}^\alpha$, $R_{drift}(t)$ did not cross the decision boundary. If the memory controller does not rewrite the cell at $t =$

[2]A large last-level DRAM cache is typically used to compensate for the relatively slower PCM access latencies. Its cache-line size is assumed to be 256B

Table 5: Maximum Capacity Per Bank of Four-Level-Cell (4LC) PCM by Soft Error Rates and Scrubbing Overhead

Scrubbing Period (s)	$SER_{combined}$	Scrubbing Overhead 100%	12.5%	1%
2	1.46E-06%	488MB	61.0MB	4.88MB
2^2	0.005%	977MB	122MB	9.77MB
2^3	0.030%	1.95GB	244MB	19.5MB
2^4	0.071%	3.91GB	488MB	39.1MB
2^5	0.132%	7.81GB	977MB	78.1MB

$100s$, the cell will spend another 100 seconds until the next scrubbing round, or a total of 200 seconds since the initial writing operation. On the second scrubbing attempt at $t = 200s$, the amount of resistance drift for this cell becomes $R_{drift}(t) = R \times \{\frac{200s}{t_0}\}^{\alpha}$, significantly larger than when it was at $t = 100s$. All in all, scrubbing in PCM must rewrite all cells on every visit to avoid exponentially increasing chances of errors. Given that scrubbing a cache line must rewrite cells with or without errors, scrubbing a cache line takes a read operation ($0.15\mu s$) and a consecutive write operation ($1.00\mu s$), a total of $1.15\mu s$. In addition to the scrubbing time, we also assume that each storage level has the same probability of occurrences.

The first column in Table 6 shows that the scrubbing overhead decreases as the scrubbing period increases. Here, the scrubbing overhead is defined as $\frac{\text{Time used for scrubbing}}{\text{Scrubbing period}}$. A 2GB PCM bank has 8M cache lines. Thus, 9.65 seconds ($\simeq 8 \times 2^{20} \times 1.15\mu s$) are required for scrubbing the entire physical PCM even if all eight PCM banks are scrubbed in parallel. As shown in Table 3, even when the memory controller performs nothing but scrubbing (100% overhead, *i.e.*, the memory controller will not have time to respond to any memory request), the SER of storage level 2 in 4LC PCM is 0.12%, which is significantly higher than that of DRAM. Moreover, if we use the scrubbing period of 45 minutes as in the DRAM for real-world servers [17], the SER of a PCM cell programmed to storage level 2 will escalate to 5%, which is intolerable. Clearly, 4LC PCM with scrubbing mechanisms cannot guarantee the most basic reliability by any standard. To reach a low SER and reduce the scrubbing overhead simultaneously, the maximum PCM capacity per bank must be limited. Our next section will show the largest capacity of 4LC PCM that the scrubbing mechanism can support.

4.2 Reducing Capacity to Achieve Low Soft Error Rates

Another way of lowering the SER of 4LC PCM is to limit the maximum capacity. We assumed that the capacity of 4LC PCM is 2GB per bank in Section 4.1 when estimating the scrubbing overhead. Since the scrubbing overhead increases proportionally with the capacity, 4LC PCM can achieve lower SER if we continue to cut down the capacity. Table 5 shows the results.

Table 5 calculates the maximum capacity of 4LC PCM for different combinations of SER and scrubbing overhead. The leftmost column represents the scrubbing period for each 256B memory block. The next column represents the combined SER which is an average of SER of all four states in 4LC PCM. However, because the third storage level shows significantly larger SER than the others, the combined SER is close to one fourth of that of the third storage level. Then, we show the maximum capacity by scrubbing overhead. When the overhead is 100%, the memory controller cannot service any request from the upper memory hierarchy. Since 100% scrubbing overhead is infeasible, the third column of Table 5 is the upper bound.

Table 5 also shows the maximum capacity when the scrubbing overheads are set to 12.5% and 1%, respectively. For example, if we design 4LC PCM with the scrubbing overhead of 1.0%, leaving 99% of the time for servicing memory requests, the maximum PCM capacity will be merely 4.88MB for achieving an average SER of 1.46×10^{-6}%. Note that when 4LC PCM comprises multiple ranks or banks, scrubbing can be performed in parallel. Thus, when one bank is being scrubbed, the other banks can respond to requests from the CPU. However, even with eight banks, the maximum capacity amounts to 39.1MB, which is still substantially below the needed main memory capacity. In sum, although a lower SER can be achieved by reducing the total capacity of 4LC PCM, the memory capacity becomes too small to be useful.

4.3 Using Error-Correcting Codes

Error-correcting codes (ECC) can be applied to compensate the SER of 4LC PCM. For example, the industry standard (72,64) Hamming code [4] can correct single bit errors by adding 8 redundant bits on top of 64-bit data.[3] This scheme is commonly found in main memory of server systems because of the simplicity in encoding and decoding [21]. Moreover, stronger ECC can also be used to protect data from multiple bit errors. For example, BCH codes [2, 5] correct 8-, 16-, 24-, or 40-bit errors from 256, 512, 1024 bytes of data depending on the size of redundant bits. Because decoding BCH codes requires more computing power and time than (72,64) Hamming code, these codes are not frequently used for latency-sensitive devices such as main memory but commonly found in slower devices such as NAND-based storage. With the combined SER for each cell of 4LC PCM developed in previous sections, we calculate the error rates after applying (72,64) Hamming code and various BCH codes. Note that for every ECC evaluated in this section, we fix the data size at 512 bits, the same size as in the previous study[1].

(72,64) Hamming code corrects one bit error, and thus, having more than two bit errors among 72 bits is uncorrectable. In addition, since storing 72 bits requires 36 4LC PCM cells, the probability of having more than two bit errors out of 36 cells can be calculated as follows. Note that by using Gray codes as described in Table 1, one step change in storage levels is limited to affect only one bit in two-bit data. Thus, two bit errors can happen only when two 4LC PCM cells are changed due to resistance drift.

Probability of having at least two bit errors in 72 bits is

$$\begin{aligned}
&= 1 - P(\text{no errors}) - P(\text{one bit error}) \\
&= 1 - (1 - SER_{combined})^{36} \\
&\quad - \binom{36}{1}(1 - SER_{combined})^{35}(SER_{combined}) \\
&= P_{error}(72b).
\end{aligned} \tag{3}$$

Now we calculate the probability of uncorrectable errors in 512 bits. 512 bits comprises eight of 64 bits data, therefore, to reconstruct the entire 512 bits, all eight blocks should not generate any uncorrectable error. If we define the result of Equation (3) as $P_{error}(72b)$, then the probability of uncorrectable error for 512 bits is defined as

$$P_{error}(512b) = 1 - (1 - P_{error}(72b))^8.$$

The fourth column in Table 6 shows the results. In Table 6, we cal-

[3]The capacity overhead is 12.5%.

Table 6: Probability of Uncorrectable Errors by ECC and $SER_{combined}$ for 2GB per Bank 4LC PCM

		Probability of Uncorrectable Errors for 512 bits $= P_{error}(512b)$					
Scrubbing Period (Overheads)	$SER_{combined}$	No ECC	$(72, 64)$	BCH-8 (512b+80b)	BCH-16 (512b+160b)	BCH-24 (512b+240b)	BCH-32 (512b+320b)
2^3 seconds (100+%)	0.030%	7.4%	0.05%	(too small)	(too small)	(too small)	(too small)
2^4 seconds (60.29%)	0.070%	16.4%	0.24%	1.44E-10%	(too small)	(too small)	(too small)
2^5 seconds (30.15%)	0.133%	28.9%	0.86%	3.80E-8%	(too small)	(too small)	(too small)
2^6 seconds (15.07%)	0.218%	42.8%	2.26%	2.64E-6%	(too small)	(too small)	(too small)
2^7 seconds (7.54%)	0.325%	56.5%	4.84%	7.45E-5%	(too small)	(too small)	(too small)
2^8 seconds (3.77%)	0.475%	70.4%	9.76%	1.54E-3%	1.27E-10%	(too small)	(too small)
2^9 seconds (1.88%)	0.668%	82.0%	17.8%	0.02%	2.32E-8%	4.11E-13%	(too small)
2^{10} seconds (0.94%)	0.91%	90.4%	29.4%	0.18%	2.15E-6%	2.81E-12%	(too small)
2^{11} seconds (0.47%)	1.21%	95.6%	44.2%	1.08%	1.10E-4%	1.34E-9%	(too small)
2^{12} seconds (0.24%)	1.57%	98.3%	60.6%	4.61%	3.14E-3%	2.66E-7%	8.69E-12%

culate the probability of uncorrectable errors by scrubbing period, scrubbing overheads, and $SER_{combined}$. If we compare the error rates of 4LC PCM to that without ECC, (72,64) Hamming code reduces the error rates, but those rates are still too high for practice. The results indicate that 4LC PCM must use stronger ECC that requires more redundant bits and higher computational overheads.

Now we calculate the probability of having uncorrectable errors with stronger ECC. On top of the 512-bit data, BCH-8 corrects up to 8 bits errors by adding 80 redundant bits, and BCH-16 corrects up to 16 bits errors by adding 160 redundant bits.[4] We generalize Equation (3) for calculating the probability of having at least n bit errors out of m bits as follows.

Probability of having at least n bit errors out of m bits is

$$= 1 - \sum_{k=0}^{n-1} \binom{m}{k} (1 - SER_{combined})^{m-k} (SER_{combined})^k. \tag{4}$$

Table 6 also shows the results from Equation (4). When the scrubbing period is 2^8 seconds, the scrubbing overhead is 3.77%, and $P_{error}(512b)$ is 1.54×10^{-3}% for BCH-8. The error rate is significantly smaller than that of 4LC PCM with (72,64) Hamming code; however, still $10^6 \sim 10^7$ times higher than the SER of DRAM without ECC support.

Table 6 shows that if we limit the maximum scrubbing overhead to 1%, 4LC PCM is only usable with BCH-32. However, such requirement prevents 4LC PCM from being used as main memory for the following reasons. For one, a memory controller with a complex error-correcting mechanism requires extra chip area and design effort, raising the chip cost. Since memory controllers are typically integrated in the same die with processor cores these days, designers need to design and fabricate a customized CPU for supporting 4LC PCM. In addition, the higher computational overhead in decoding increases the memory latency and degrades the performance. For these reasons, the majority of commodity systems typically implement no ECC schemes or at most the (72,64) Hamming code.

Moreover, the most critical downside of BCH-32 is the capacity overhead. To correct up to 32 errors from 512 bits of data, we must add 320 parity bits to make a total of 832 bits of data. In storing 832 bits, 416 4LC PCM cells are needed. On the other hand, 416 3LC PCM cells can store 659 bits ($= \lfloor log_2(3^{416}) \rfloor$). Note that because 3LC PCM has no soft error in the time range of interest, all 659 bits can be used to store useful information without parity bits. In summary, 3LC PCM theoretically achieves higher information density than 4LC PCM.

[4]The capacity overheads are 15.6% and 31.3%, respectively.

4.4 Increasing Programming Precision

We discussed earlier that the parameters in our analysis are based on previous studies [1, 20], which assumed that desired programmed resistance ranges are $10^{\mu_R \pm 2.75\sigma_R}$ for μ_R and σ_R in Table 1. If we can increase the programming precision or fine-tune the resistance levels from $10^{\mu_R \pm 2.75\sigma_R}$ to $10^{\mu_R \pm 1.375\sigma_R}$, for example, then SER of 4LC PCM will decrease; however, at a cost of more write iterations.

Although the relationship between resistance ranges and the number of write iterations has yet to be discovered, we estimate it by extending the previous study, which proposed a mathematical model for calculating write iterations [13]. Their method plugged the ratio between the number of desired storage levels and the number of levels that can be reached in a single write attempt into the Bernoulli distribution. We extend this model to calculate how many iterations will be required in increasing programming precision. More specifically, 4LC PCM with half of the resistance range ($10^{\mu_R \pm 1.375\sigma_R}$) can be considered as 8LC PCM with unused four storage levels in-between four storage levels.[5] Such narrow range triggers almost 2x more write iterations, resulting into the following problems. Extra iterations slow down the writing process, degrade the performance, wear out PCM cells, and consume extra power. Nevertheless, SER of such 4LC PCM in 32 seconds is 1.9×10^{-7}%, which is more than 10^{10} more errors than 3LC PCM. In summary, one can decrease SER of 4LC PCM by increasing programming precision; however, 4LC PCM with such high precision compromises write latency and lifespan of PCM cells while still suffering from significantly higher SER than 3LC PCM or DRAM. The next section explains practical implementation for 3LC PCM.

5. TRI-LEVEL-CELL (3LC) PCM IN PRACTICE

5.1 Binary-to-Ternary Conversion

Since tri-level cells do not match any conventional binary digital system, we need an efficient way of converting binary information to the ternary number system and vice versa. The efficiency of

[5]According to the original notation, F changes from 0.5 to 0.25.

conversion methods can be evaluated by cell utilization and implementation feasibility. In other words, it is always desirable for a conversion method to fully utilize the cell capacity with minimal hardware overheads.

First, we need a way to evaluate the cell utilization of a conversion method. For example, if a conversion method uses $\frac{n}{2}$ four-level cells to store n-bit data, then the four-level cells can be regarded as fully utilized. On the contrary, if a conversion method needs n four-level cells to store n-bit data, then its cell utilization is halved. This concept can be generalized as follows. When a conversion method uses m k-level cells to store n-bit data, its cell utilization is

$$\text{Cell Utilization} = \frac{\log_k 2^n}{m} = \frac{n}{m}\log_k 2. \tag{5}$$

In this equation, 2^n is the number of different states represented by n-bit data and $\log_k 2^n$ is the theoretically minimum number of k-level cells to store n-bit data. For instance, if a 4LC PCM requires a BCH-32 error correction scheme to prevent drift-induced soft errors, the cell utilization of the binary to quaternary conversion is calculated as $\frac{512}{416}\log_4 2 \simeq 0.615$ where BCH-8 requires 320 more parity bits to recover 512-bit data from eight errors.

As mentioned in Section 3, the 3LC PCM does not require any complicated error correction scheme. However, some capacity loss of 3LC PCM is unavoidable due to binary-to-ternary conversion. In other words, when using n bits as a basic store unit, the minimum number of 3LC PCM cells to store the n bits is $\lceil n \log_3 2 \rceil (= m)$. For example, storing three-bit data requires at least two 3LC PCM cells. The two 3LC PCM cells are used for differentiating 2^3 states even though they can represent maximum 3^2 states. Thus, one of the states represented by two 3LC PCM cells is remained unused. Figure 1 shows the achievable cell utilization for $\langle n, m\rangle$ binary-to-ternary conversion methods by the size of a basic store unit, n, from one to 32. Among those conversion methods, the $\langle 19, 12\rangle$ conversion storing 19 bits to 12 3LC PCM cells can achieve the highest cell utilization of 0.999, while the cell utilization of a $\langle 8, 6\rangle$ conversion method is at most 0.841. For reference, in the $\langle 1, 1\rangle$ and $\langle 2, 2\rangle$ conversion methods, a 3LC PCM cell acts like a SLC PCM cell and the cell utilization is 0.631.

In evaluating conversion methods, another important factor that should be considered is implementation complexity. Those $\langle n, m\rangle$ conversion methods for 3LC PCM can be implemented using either look-up tables (LUT), arithmetic units, or basic logic gates. Each implementation method has its own advantages and disadvantages. Using a look-up table can reduce the conversion latency, while the number of table entries is exponentially increased when we increase the size of a basic store unit, n. On the other hand, calculating ternary numbers using arithmetic units occupies less hardware overhead than LUT; however, the latency is compromised due to slow arithmetic operations. In general, increasing a basic store unit, n, to achieve higher cell utilization results in higher hardware costs or longer access latencies. Moreover, since a conversion method should be embedded inside a memory chip, its hardware cost and access latency are critical for its implementation feasibility.

Therefore, we propose to use a simple number mapping method such as $\langle 1, 1\rangle$, $\langle 2, 2\rangle$, and $\langle 3, 2\rangle$ conversion methods that are implementable with simple logic gates. With a combination of the three conversion methods, we can build any conversion method whose cell utilization is less than or equal to the cell utilization of $\langle 3, 2\rangle$, 0.946. For example, a $\langle 8, 6\rangle$ conversion can be translated to two $\langle 3, 2\rangle$ conversions and one $\langle 2, 2\rangle$ conversion, *i.e.,* $2\langle 3, 2\rangle + \langle 2, 2\rangle = \langle 8, 6\rangle$, and a $\langle 16, 11\rangle$ conversion can be translated to five $\langle 3, 2\rangle$ conversions and one $\langle 1, 1\rangle$ conversion, *i.e.,* $5\langle 3, 2\rangle + \langle 1, 1\rangle = \langle 16, 11\rangle$.

Table 7: An example of the $\langle 3, 2\rangle$ number mapping method.

3-digit binary	2-digit ternary	control signals			
		cell$_1$ for t_1		cell$_0$ for t_0	
($b_2b_1b_0$)	(t_1t_0)	s_{11}	s_{10}	s_{01}	s_{00}
000	00	0	0	0	0
001	01	0	0	0	1
010	12	0	1	1	x
011	02	0	0	1	x
100	10	0	1	0	0
101	20	1	x	0	0
110	22	1	x	1	x
111	21	1	x	0	1

Relationship between ternary levels and control signals:

"Programming"

t_c	p_{c1}	p_{c0}
2	1	x
1	0	1
0	0	0

"Reading"

t_c	r_{c1}	r_{c0}
2	1	x
1	0	1
0	0	0

where "x" means redundant condition

Table 7 shows an example of the $\langle 3, 2\rangle$ number mapping method. In this example, eight ternary states except for the 11 state are used to represent three-bit binary data. This simple number mapping method can be implemented using several logic gates. We assume that three-bit data, $b_2b_1b_0$, is stored to two tri-level cells, t_1t_0, and each cell uses two control signals, p_{c1} and p_{c0}, to select a programming current corresponding to its state, where c indicates a corresponding cell number. When the relationship between the cell states and their control signals is defined as in Table 7, the control signals can be represented by the three binary bits as

$$\begin{aligned} p_{11} &= b_2 \cdot b_1 + b_2 \cdot b_0, & p_{10} &= b_2 + b_1 \cdot \overline{b_0} \\ p_{01} &= \overline{b_2} \cdot b_1 + b_1 \cdot \overline{b_0}, & p_{00} &= b_1 + \overline{b_2} \cdot b_0. \end{aligned}$$

Similarly, when reading a 3LC PCM cell, its programmed resistance is represented by the outputs of two sense-amplifiers, r_{c1} and r_{c0} as described in Table 7. From the four outputs of two cells, the three-bit data can be decoded by simple logic gates as

$$\begin{aligned} b_2 &= r_{11} + r_{10} \cdot \overline{r_{01}} \cdot \overline{r_{00}} \\ b_1 &= r_{01} + r_{11} \cdot r_{00} \\ b_0 &= r_{11} \cdot \overline{r_{01}} + \overline{r_{11}} \cdot \overline{r_{10}} \cdot r_{01} + \overline{r_{11}} \cdot \overline{r_{10}} \cdot r_{00}. \end{aligned}$$

Our ternary-to-binary conversion employs AND, OR, and NOT gates, and the critical path of the conversion is maximally three-gate delay. Considering that the typical operating frequency of main memory is significantly lower than that of CPU, three more gate delays will not result in a major impact in timing.

In this section, we show that by using a $\langle 3, 2\rangle$ number mapping method we can achieve the cell utilization of up to 0.946 with low-cost hardware. Thus, when using an $\langle 8, 6\rangle$ conversion composed of two $\langle 3, 2\rangle$ and one $\langle 2, 2\rangle$ conversion methods, 512-bit data can be stored in 384 tri-level cells. Here, it is noteworthy that 4LC PCM requires 416 cells to store 512-bit data when using a BCH-32 error correcting scheme to achieve a confident level of reliability.

5.2 Bandwidth Enhancement

So far, we achieved the desired reliability with 3LC PCM by eliminating the most error-prone state from the four-level cell PCM as shown in Figure 2(a). To program a cell to the intermediate state,

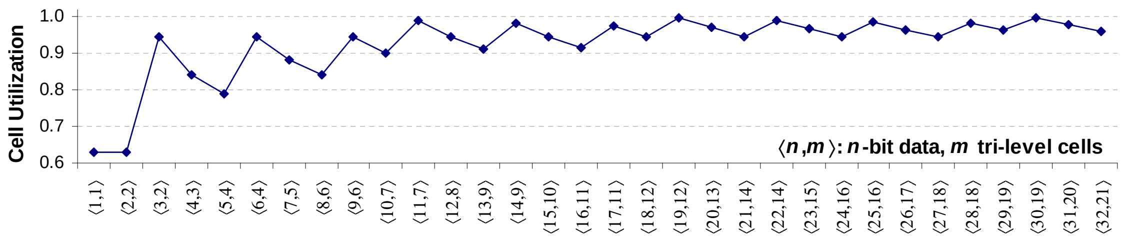
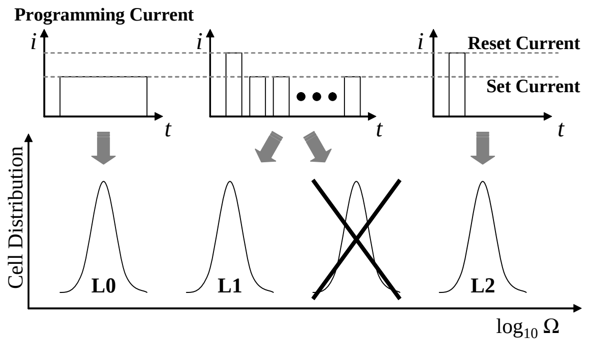
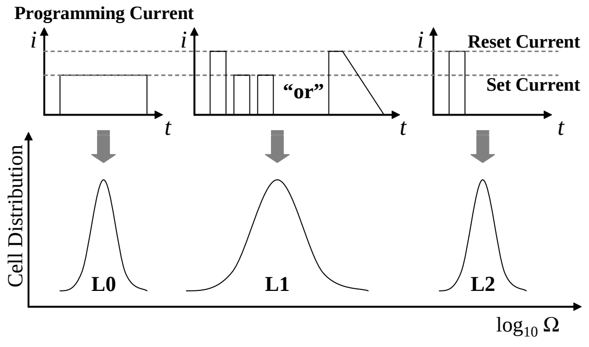


Figure 1: Tri-Level-Cell Utilization



(a) 3LC PCM Programming

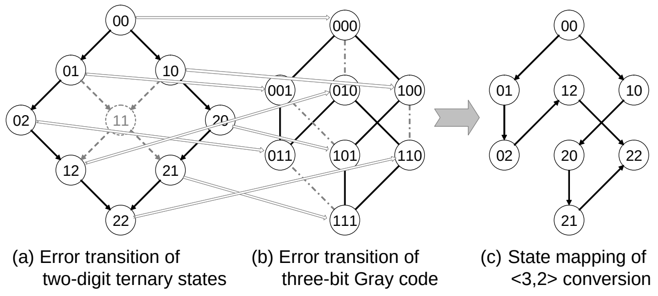


(b) Bandwidth-Enhanced (BE) 3LC PCM Programming

Figure 2: Cell Distribution vs. Programming Sequence

"L1", 3LC PCM uses the same write-and-verify iterations as 4LC PCM. Since a prime concern in 4LC PCM is to maximize its reliability, it is desirable to precisely tune the resistance of intermediate levels to secure drift margins between storage levels as large as possible. However, such precise programming leads to a long write latency, which is the root cause of low write bandwidth in MLC PCM. The question is whether the tight resistance ranges for intermediate levels achieved by write-and-verify iterations are still necessary for 3LC PCM.

According to our analytical model, the 3LC PCM using the same resistance ranges with 4LC PCM is virtually free from drift-induced soft errors. As described in Table 4, its SER is extremely small even comparing with that of DRAM, indicating that the resistance range for the intermediate level is unnecessarily tight. Since the tight resistance range is obtained by sacrificing the write latency, we can reduce the write latency by relaxing the resistance range and eventually improve the overall write bandwidth. Therefore, we propose bandwidth-enhanced 3LC (BE-3LC) PCM using a relaxed resistance range for the intermediate level.



00
01
10
02
11
20
12
21
22
000
001
010
100
011
101
110
111
00
01
12
10
02
20
22
21
(a) Error transition of two-digit ternary states
(b) Error transition of three-bit Gray code
(c) State mapping of <3,2> conversion

Figure 3: State mapping of $\langle 3, 2\rangle$ conversion for efficient error correction in 3LC PCM

Figure 2(b) shows two examples to program a relaxed intermediate level in 3LC PCM. The relaxed intermediate level can be programmed with less number of write iterations because of its widened resistance range. Another choice in programming the relaxed intermediate level is to use the moderate-quenched (MQ) programming which controls the falling slope of a reset current pulse [10]. By using the MQ programming method, the write latency of an intermediate level in 3LC PCM cell can be reduced below the set latency, *i.e.*, the write latency of SLC PCM.

As mentioned earlier, relaxing the acceptable resistance range for the intermediate level helps to reduce the write latency and enhance write bandwidth. However, it reduces the drift margin between resistance levels, and the new narrower margin increases drift-induced SER. In Section 5.3, we will introduce how to use conventional ECC schemes for the slightly increased SER of BE-3LC PCM. Also, we will evaluate the SER of both 3LC PCM and BE-3LC PCM in Section 6.1.

5.3 Efficient $\langle 3, 2\rangle$ Conversion for Error Correction

Using error correcting codes can improve the reliability of 3LC PCM as in other memory systems. The problem is that how 3LC PCM can efficiently use the conventional ECC schemes. In the case of 4LC PCM, four states of a cell is encoded with two-bit Gray code. By doing so, one state transition in a four-level cell affects only one bit in binary data, which enables to use a binary error-correcting code for correcting the state transition of four-level cells. Similarly, if one drift-induced error in a tri-level cell affects only one bit in the corresponding binary code, a binary error correcting code can be used for recovering the data from the drift-induced error.

In this section, we propose a state mapping method of $\langle 3, 2\rangle$ conversion for using binary error correcting codes. Figure 3(a) shows possible state transitions caused by drift-induced errors in two 3LC PCM cells. Because the resistance drift increases the resistance level of a PCM cell, *i.e.,* from level 0 to level 1 or from level 1 to level 2, the state transitions are uni-directional. The main idea is to map the state transition graph of two 3LC PCM cells into the transition graph of the three-bit Gray code depicted in Figure 3(b).

First, we exclude "11" state from the state mapping of our $\langle 3,2\rangle$ conversion. Note that the state "11" was excluded because it has four transition edges, which cannot be mapped into the Gray code, and also because two tri-level cells have one more state than a three-bit binary code. Then, the rest of states and edges are mapped into the Gray code graph as shown in Figure 3(c)[6], which indicates that all one-hop error transitions of the two-ternary-cell states except ones from/to the "11" state are translated to one-hop error transitions of the three-bit binary code. Note that we need a special process for the "11" state because removing the "11" state from the state mapping cannot prevent error transitions to the "11" state. When the "11" state is read from two 3LC-PCM cells, it indicates that the state results from one or more drift-induced errors. Also, considering the monotonically increasing nature of resistance drift, only "00", "01", and "10" states can be shifted to the "11" state. Thus, when the "11" state is read from the two tri-level cells, we can limit the maximum number of transition-error hops to one by substituting it with a "00" state. By doing so, one state transition error caused by resistance drift affects only one data bit. For example, we assume that (72,64) Hamming code is used for single error correction and double error detection (SECDED). The 72-bit code can be stored in 48 3LC PCM cells when using $\langle 3,2\rangle$ conversion. With the state mapping of $\langle 3,2\rangle$ conversion, the (72,64) Hamming code detects two drift-induced errors in the 48 tri-level cells and corrects one drift-induced error.

Furthermore, it is noteworthy that a 72-bit PCM DIMM comprises 8 PCM chips, and each PCM chip has a 9-bit datapath, which are matched to three $\langle 3,2\rangle$ conversion units. Otherwise, if eight 8-bit PCM chips are used to build a 64-bit PCM DIMM for symmetry, each chip is forced to use $\langle 8,6\rangle$ conversion. As a result, the 64-bit data is stored in 48 3LC PCM cells, which is the same amount of 3LC PCM cells that are used to store 72 bits. Therefore, when plugged into a realistic PCM DIMM organization, our $\langle 3,2\rangle$ state mapping method allows to use the (72,64) Hamming code without additional storage overhead.

5.4 Temperature dependency

Throughout this paper, we compare 3LC and 4LC with the assumption that they are operated at the same ambient temperature. When they are operating at a higher temperature, 4LC suffers more from the high error rates because 4LC makes use of more storage levels and secures smaller safety margin among storage levels. Therefore, 3LC holds clear benefit over 4LC even at a higher temperature. Moreover, previous study showed that PCM cells are not sensitive to thermal disturbance [19]. Such observation is notable when considering that the temperature of GST material in PCM should be raised up to $300 \sim 600^{\circ}C$ when programming. Since typical operating temperature of PCM device is under $100^{\circ}C$, we argue that temperature dependency will show little impact on the findings presented on this paper.

6. EVALUATION

6.1 Soft Error Rate of BE-3LC PCM

As discussed in Section 5.2, 3LC PCM can reduce writing latency by using fewer writing iterations. As such, the distribution of the resistance is compromised, and which will increase the SER

[6]This state mapping is the same as one in Table 7.

Table 8: Physical Parameters for the Second Storage Level of 3LC PCM When $t_0 = 1$ s.

Writing Strategy	$\log_{10}(R)$		α	
	μ_R	σ_R	μ_α	σ_α
Iterative	4.0	0.167	0.02	$0.4 \times \mu_\alpha$
Non-iterative	4.255	0.188	0.02157	

of the PCM cell. In this section, we formulate the relationship between writing latency and the SER of 3LC PCM and argue that 3LC PCM can achieve the writing latency close to SLC-PCM without compromising the SER.

Kang *et al.* [10] shows the distribution of the resistance of a PCM cell by two different writing strategies; (i) iterative writing (write and verify) and (ii) writing without iterations. As 3LC PCM does not use the third storage level, we focus on the distribution of the second storage level. More specifically, we read the distribution of the resistance of the second storage level from Figure 1 based on [10] and calculate the mean and the variance of the resistance when a cell is written without iterations. As we show in Table 8, σ_R and μ_α are worsened from 4.0 to 4.255 and from 0.167 to 0.188, respectively.

In Table 8, physical parameters for two different writing methods, iterative and non-iterative methods, are taken from two different technology nodes. However, we compensate such differences by not taking the exact numbers but taking relative ratios between two different writing methods from [10]. In addition, we assume linear increment in μ_α and σ_α by σ_R for estimating the distribution of α. For example, we use the physical parameters in Table 1 and apply 0.04 increment in μ_α for every 10x in R.

After obtaining the physical parameters of the second storage level of 3LC PCM, we calculate the SER of 3LC PCM by using analytical models discussed in Appendix A. The summary of results is as follows. Firstly, the majority of the errors happen in between the set state and the second storage level. Such errors are not due to the resistance drift, but because of the initial writing failure. For example, the memory controller writes 01 to a 3LC PCM cell, and the cell reads 00 immediately after the writing. The error rate for this case is $3.04 \times 10^{-3}\%$. Secondly, if we exclude such initial writing failures, the SER of 3LC PCM caused by resistance drift is negligible until $t = 2^{20}$ seconds. Table 9 shows the error rate for this case.

When a PCM chip reads PCM cells immediately after writing them, the chip can detect and rewrite the cells to fix the initial writing failures. More specifically, we assume that the PCM chips rewrite the cells by sensing the written values immediately after writing. Even though such strategy is similar to the iterative writing commonly used in 4LC-PCM, this strategy is different in terms of the expected numbers of iterations. For the $(100 - 3.04 \times 10^{-3})\%$ of the time, writing to our proposed BE-3LC-PCM finishes at the first attempt. The second attempt is required for only $3.04 \times 10^{-3}\%$ of the time, and the expected numbers of writing iterations in this case is close to one. Moreover, we also show the SER of 3LC-PCM with industry standard (72,64) ECC support in the rightmost column of Table 9. This column is calculated based on the fact that 48 3LC-PCM cells store 72 bits, and (72,64) ECC corrects one bit error. Again, the proposed PCM with (72,64) ECC shows a negligible SER until $t = 2^{25}$ seconds. In summary, writing to 3LC PCM cells must be followed by reading and verifying. Such small overhead will remove majority of the errors, and 3LC PCM experiences no errors in the time range of our interest.

Table 9: Soft Error Rates of Intermediate Storage Level of BE-Tri-Level-Cell (BE-3LC) PCM

Scrubbing Period (s)	Iterative Writing	BE Writing	BE Writing $+(72,64)$ ECC
2^5	(too small)	(too small)	(too small)
2^{10}		(too small)	(too small)
2^{15}		(too small)	(too small)
2^{20}		3.60E-16%	(too small)
2^{25}		1.28E-10%	2.66E-15%

6.2 Performance

4LC PCM requires a scrubbing mechanism and a multiple-error correction scheme for a confident level of reliability. However, to use the 4LC PCM, we have to consider other aspects such as performance and hardware overhead. If the gain from its high density needs other considerable cost, the 4LC PCM will be regarded as infeasible. First, to evaluate the performance impact of using MLC PCM, we simulated 26 applications from SPEC2006 benchmark using SESC [16]. The read and write latencies of SLC PCM are assumed to be 150ns including a row activation latency (tRC) of 120ns, and 200ns considering an internal write verification delay, respectively [3]. For 3LC and 4LC PCMs, its read latency is assumed to be the same as that of SLC's because 3LC and 4LC can be read as fast as SLC if they have multiple sense amplifiers in parallel. On the contrary, 3LC and 4LC's write latency is assumed to be 1000ns because of its iterative write-and-verify iterations [1]. Similar to other studies [14, 15], an 8MB L3 DRAM cache composed of 256B cache-lines is employed to hide the PCM access latency. Also, we assumed a PCM main memory composed of eight 2GB banks and we modeled a memory controller that can efficiently schedule memory requests by exploiting bank-level parallelism and PCM row buffer hits. Note that in the request scheduling, read requests have higher priority than write requests because write accesses are typically not on the critical path in terms of performance.

Figure 4 shows the relative instruction-per-cycle (IPC) values normalized to the IPC of SLC PCM. As in the recent paper [1], the two 4LC PCM configurations are assumed to use a BCH code capable of eight error corrections for a 512-bit data block.[7] According to our analysis, the 4LC PCM with the BCH code has to scrub the entire memory space every eight seconds to achieve a DRAM-level soft error rate. However, the eight-second scrubbing is impossible because the minimum latency to scrub a 2GB PCM bank is about 9.6 seconds. Thus, we chose a 16 second scrubbing for 4LC PCM.

Awasthi *et al.* proposed a scrubbing overhead reduction scheme called *Light Array Read for Drift Detection* (LARDD) [1]. For LARDD scheme, we assume an eight-second LARDD period, again, to achieve a DRAM-level soft error rate. More specifically, BCH-8 column in Table 6 shows that LARDD with eight-second period has sufficiently low SER. However, we argue that the actual SER for this scheme is higher than what we show in Table 6 for the following reasons. The benefit of LARDD comes from the fact that LARDD skips scrubbing rounds for data blocks. In other words, if LARDD cannot skip scrubbing rounds, then its behavior is exactly the same as typical scrubbing. As a result, some data blocks are revisited in at least $2\times$ of the period, which will increase the SER.

[7]We assumed the encoding and decoding latencies of the eight-error-correction BCH code take one memory clock cycle because the encoding and decoding logic can be fully parallelized by accepting its exponentially increased area overhead.

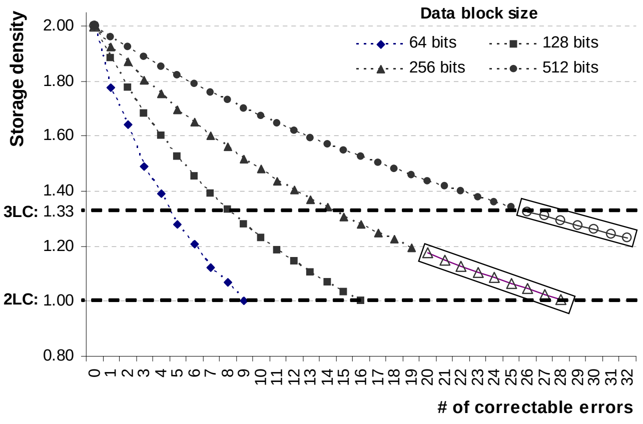


Figure 6: Information density of 4LC PCM

As shown in Figure 4, the 4LC PCM scrubbed every 16 seconds experienced 72.2% performance degradation on average. Especially, 429.mcf that shows the highest write frequency (2.81 per 1000 instructions) incurred 95.2% performance degradataion. This is because of the five times longer write latency of 4LC PCM and its scrubbing overhead occupying 60.0% of total execution time. This tremendous performance degradation can be reduced by employing the LARDD scheme. However, the LARDD still experienced 26.7% performance degradation. This means that although LARDD reduces the write frequency to PCM, there are still too many read-and-check operations performed inside a chip, leading to substantial performance degradation. On the other hand, the 3LC PCM experienced only 10.4% performance degradation on average, although its write latency is also 1000 ns as in 4LC PCM.

Furthermore, the performance of 3LC PCM can be improved by using the bandwidth-enhanced (BE) 3LC. Figure 5 shows the relative IPC of BE-3LC PCM which is normalized to the IPC value of SLC PCM. The write latency of BE-3LC PCM is obviously decreased close to SLC PCM's latency, however, estimating the accurate write latency is beyond this research scope. Thus, we performed a sensitivity study varying its write latency from 350ns down to 200ns monotonically decremented by a 50ns interval. In addition, the SER of BE-3LC is confined to less than 3.6×10^{-18} when the BE-3LC is scrubbed every 2^{20} seconds, as mentioned in Section 6.1. Thus, all the BE-3LC PCM configurations are assumed to use a 2^{20} second scrubbing scheme. The relative IPC of the four configurations are 0.982, 0.988, 0.994, and 1.000, respectively. As a result, BE-3LC PCM makes it feasible to achieve the increment of memory capacity with negligible performance degradation, compared with SLC PCM.

6.3 Information Density

Another way to reinforce 4LC PCM reliability is to increase the number of correctable errors in a data block. However, if the number of additional cells required for a multiple error correction code is equal to or larger than the number of data cells, then it is meaningless to use 4LC PCM. For example, since the BCH code correcting nine errors in a 64-bit data block requires additional 64 bits, a total of 64 4LC PCM cells should be used to store the 128 bits. Then, the 4LC PCM using a nine-bit correction (128,64) BCH code is no better than using SLC PCM.

Here, we define *information density* as the number of data bits stored in one cell to measure the cell efficiency. For instance, information density of SLC PCM is 1.00 because every SLC PCM

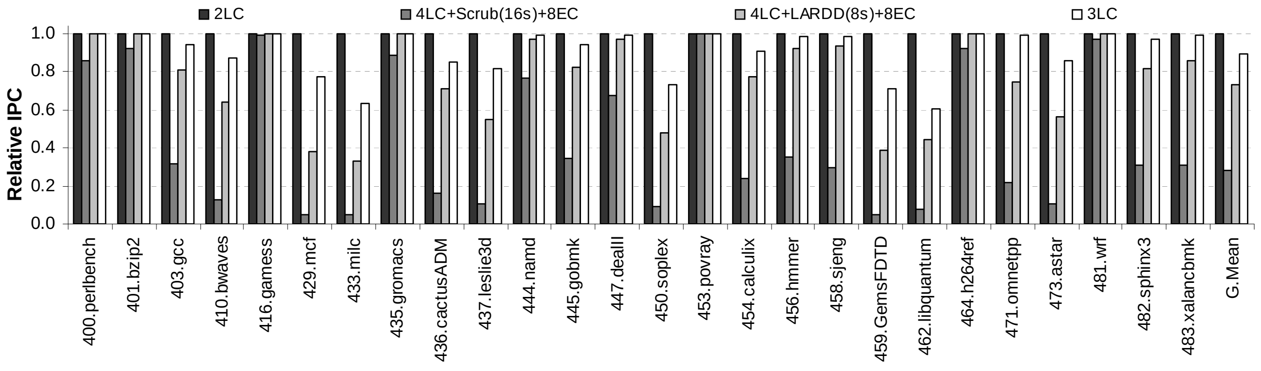


Figure 4: Performance comparison with 4LC and 3LC

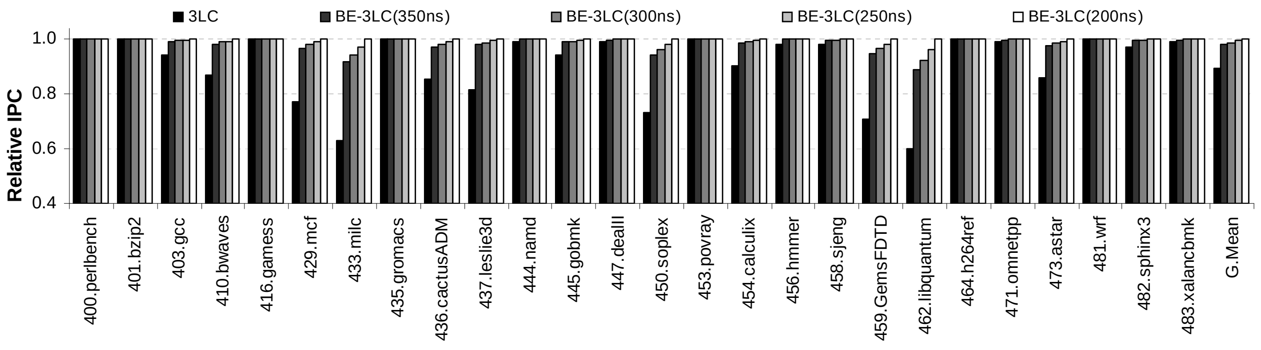


Figure 5: Sensitivity study of bandwidth-enhanced 3LC

cell stores one data bit, and SLC PCM does not require capacity overheads from ECC. In the case of 3LC PCM, it uses an $\langle 8, 6\rangle$ conversion scheme and thus its information density is $\frac{8}{6} \simeq 1.33$. In the proposed BE-3LC PCM, a (72,64) hamming code is stored to 48 cells. Thus, its information density is still $\frac{64}{48} \simeq 1.33$.

In Figure 6, we compare the information density of 4LC PCM with SLC PCM and our proposed 3LC PCM. For example, the eight-bit correction (592,512) BCH code uses $\frac{592}{2}$ cells to store a 512-bit data block and its information density is 1.73. However, the 4LC PCM using a (592,512) BCH code requires an eight-second scrubbing scheme to achieve confident reliability, which seriously degrades performance as discussed in Section 6.2. If we use a strong error correction code recovering a data block from more errors, we can reduce the scrubbing frequency and diminish its performance degradation caused by scrubbing operations. Thus, we evaluate the SER of each configuration when a scrubbing period is 2^{10} seconds. Because it spends 9.65 seconds to scrub all 256B memory lines in a 2GB PCM bank, the maximum performance degradation caused by scrubbing operations can be limited to less than 1.00% ($> \frac{9.65}{2^{10}}$). According to our analytical model, when the size of a data block is 512 bits, a 26 or more error correction scheme is required to achieve the same level of SER with the proposed BE-3LC in Table 9. When using a 256-bit data block, an error correcting scheme has to be able to correct 20 or more errors. As shown in Figure 6, those configurations marked in rectangles have lower information density than that of 3LC PCM, 1.33. In other words, 3LC PCM is more efficient than 4LC PCM to store data bits at the same level of reliability.

7. CONCLUSION

In this paper, we asked the question of how reliable the widely studied four-level-cell (4LC) PCM can be by exploiting the schemes aimed at overcoming the resistance drift problems. We modeled the resistance drift in MLC PCM and showed that conventional ECC schemes and scrubbing mechanisms are not usable in 4LC PCM for minimizing their drift-induced soft errors to a satisfactory reliability level due to their unduly overheads and certain physical limitation. We evaluated architectural approaches to addressing drift issues in 4LC PCM including efficient scrubbing mechanisms and multiple error correction schemes. We found that the latest scrubbing mechanism still suffers significant performance degradation (26.7%) compared to the use of 2LC PCM. On the other hand, the performance of scrubbing mechanism can be alleviated by using a stronger error correction code for correcting multiple errors, however, the increase of codeword length compromises *information density*, *i.e.,* the number of data bits stored in each cell, to lower than 1.33.

In this paper, we propose tri-level-cell (3LC) PCM to remove the most drift-error-prone level from 4LC PCM. This new technology eliminates the reliability concerns due to drift-induced errors. Furthermore, by relaxing an acceptable resistance range of the intermediate level, the programming latency of 3LC PCM can be reduced close to that of 2LC PCM, making the performance impact negligible. Also, we propose a state-mapping $\langle 3, 2\rangle$ conversion to efficiently store binary data to tri-level (ternary) cells. The state-mapping $\langle 3, 2\rangle$ conversion scheme can be implemented with simple logic gates. Another merit of the state-mapping scheme is that it enables a conventional binary ECC scheme such as a $(72, 64)$ Hamming code for correcting a ternary cell error while maintaining its information density to at least 1.33. By using our 3LC PCM, we can obtain benefits from the inherently increased memory capacity without concerns of memory reliability and performance degradation.

8. ACKNOWLEDGMENT

This research is supported by an IBM faculty award, an NSF grant CNS-0644096, and gift from Intel Corp. The authors would also like to thank Karthikeyan Sankaralingam of the University of Wisconsin for technical discussion and valuable feedback.

9. REFERENCES

[1] M. Awasthi, M. Shevgoor, K. Sudan, B. Rajendran, R. Balasubramonian, and V. Srinivasan, “Efficient Scrub Mechanisms for Error-Prone Emerging Memories,” in *Proceedings of the International Symposium on High Performance Computer Architecture*, 2012.

[2] R. Bose and D. Ray-Chaudhuri, “On a class of error correcting binary group codes,” *Information and control*, vol. 3, no. 1, pp. 68–79, 1960.

[3] Y. Choi, I. Song, M.-H. Park, H. Chung, S. Chang, B. Cho, J. Kim, Y. Oh, D. Kwon, J. Sunwoo, J. Shin, Y. Rho, C. Lee, M. G. Kang, J. Lee, Y. Kwon, S. Kim, J. Kim, Y.-J. Lee, Q. Wang, S. Cha, S. Ahn, H. Horii, J. Lee, K. Kim, H. Joo, K. Lee, Y.-T. Lee, J. Yoo, and G. Jeong, “A 20nm 1.8V 8Gb PRAM with 40MB/s Program Bandwidth,” in *Technical Digest of the 2012 IEEE International Solid-State Circuits Conference*, 2012.

[4] R. Hamming, “Error detecting and error correcting codes,” *Bell System Technical Journal*, vol. 29, no. 2, pp. 147–160, 1950.

[5] A. Hocquenghem, “Codes correcteurs d’erreurs,” *Chiffres*, vol. 2, no. 2, pp. 147–156, 1959.

[6] 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 the 2010 Symposium on VLSI Technology (VLSIT)*, 2010, pp. 201–202.

[7] D. Ielmini, A. Lacaita, and D. Mantegazza, “Recovery and drift dynamics of resistance and threshold voltages in phase-change memories,” *IEEE Transactions on Electron Devices*, vol. 54, no. 2, pp. 308–315, 2007.

[8] D. Ielmini, S. Lavizzari, D. Sharma, and A. Lacaita, “Physical interpretation, modeling and impact on phase change memory (PCM) reliability of resistance drift due to chalcogenide structural relaxation,” in *Proceedings of the IEEE International on Electron Devices Meeting (IEDM)*, 2007, pp. 939–942.

[9] L. Jiang, Y. Zhang, and J. Yang, “ER: elastic RESET for low power and long endurance MLC based phase change memory,” in *Proceedings of the 2012 ACM/IEEE international symposium on Low power electronics and design*, 2012, pp. 39–44.

[10] D. Kang, J. Lee, J. Kong, D. Ha, J. Yu, C. Um, J. Park, F. Yeung, J. Kim, W. Park *et al.*, “Two-bit cell operation in diode-switch phase change memory cells with 90nm technology,” in *Proceedings of 2008 Symposium on VLSI Technology*, 2008, pp. 98–99.

[11] T. Nirschl, J. Phipp, T. Happ, G. Burr, B. Rajendran, M. Lee, A. Schrott, M. Yang, M. Breitwisch, C. Chen *et al.*, “Write strategies for 2 and 4-bit multi-level phase-change memory,” in *IEEE International Electron Devices Meeting (IEDM)*, 2007, pp. 461–464.

[12] N. Papandreou, H. Pozidis, T. Mittelholzer, G. Close, M. Breitwisch, C. Lam, and E. Eleftheriou, “Drift-tolerant multilevel phase-change memory,” in *2011 3rd IEEE International Memory Workshop (IMW)*. IEEE, pp. 1–4.

[13] M. Qureshi, M. Franceschini, and L. Lastras-Montano, “Improving read performance of phase change memories via write cancellation and write pausing,” in *Proceedings of the International Symposium on High Performance Computer Architecture*, 2010.

[14] M. Qureshi, V. Srinivasan, and J. Rivers, “Scalable high performance main memory system using phase-change memory technology,” in *Proceedings of the 36th International Symposium on Computer Architecture*, 2009.

[15] M. K. Qureshi, J. Karidis, M. Fraceschini, V. Srinivasan, L. Lastras, and B. Abali, “Enhancing Lifetime and Security of Phase Change Memories via Start-Gap Wear Leveling,” in *Proceedings of the International Symposium on Microarchitecture*, 2009.

[16] J. Renau, B. Fraguela, J. Tuck, W. Liu, M. Prvulovic, L. Ceze, S. Sarangi, P. Sack, K. Strauss, and P. Montesinos, “SESC simulator,” January 2005, http://sesc.sourceforge.net.

[17] B. Schroeder, E. Pinheiro, and W. Weber, “Dram errors in the wild: a large-scale field study,” in *Proceedings of the eleventh international joint conference on Measurement and modeling of computer systems*. ACM, 2009, pp. 193–204.

[18] N. H. Seong, D. H. Woo, and H.-H. S. Lee, “Security Refresh: Protecting Phase-Change Memory against Malicious Wear Out,” *IEEE Micro*, vol. 31, no. 1, pp. 119–127, 2011.

[19] N. H. Seong, D. H. Woo, V. Srinivasan, J. A. Rivers, and H.-H. S. Lee, “SAFER: Stuck-at-fault error recovery for memories,” in *Proceedings of the 43rd IEEE/ACM International Symposium on Microarchitecture*, 2010.

[20] W. Xu and T. Zhang, “A time-aware fault tolerance scheme to improve reliability of multilevel phase-change memory in the presence of significant resistance drift,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 19, no. 8, pp. 1357–1367, 2011.

[21] D. H. Yoon and M. Erez, “Virtualized and flexible ecc for main memory,” in *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems*, 2010.

[22] W. Zhang and T. Li, “Helmet: A resistance drift resilient architecture for multi-level cell phase change memory system,” in *Proceedings of 2011 IEEE/IFIP 41st International Conference on Dependable Systems & Networks (DSN)*, 2011, pp. 197–208.

APPENDIX

A. ANALYTICAL ERROR MODEL AND VALIDATION

In building an analytical error model for both 4LC PCM and 3LC PCM, we continue discussion on top of Table 1 and Table 2. First, we define two more variables, $m = \log_{10} R$ and $n = \log_{10} t$. By substituting Equation (1) with m and n, we obtain

$$\log_{10}(R_{drift}(t)) = \log_{10} R + \alpha \log_{10} t = m + n\alpha.$$

Thus, the condition of a soft error can be rewritten as

$$m+n\alpha > \mu_R + E$$
$$n\alpha > \mu_R + E - m,$$
$$\text{where } E = \begin{cases} 0.5 \text{ for storage level 0, 1, and 2 of 4LC PCM} \\ 0.5 \text{ for storage level 0 of 3LC PCM} \\ 1.5 \text{ for storage level 1 of 3LC PCM.} \end{cases}$$

As α follows $N(\mu_\alpha, \sigma_\alpha^2)$, $n\alpha$ follows $N(n\mu_\alpha, (n\sigma_\alpha)^2)$. The probability for $n\alpha$ to be more than $\mu_R + E - m$ can be calculated as follows.

Probability of soft error for a given m is

$$= 1 - \Phi(\frac{\mu_R + E - m - n\mu_\alpha}{n\sigma_\alpha}), \quad (6)$$
$$\text{where } \Phi(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{-x^2/2} dx.$$

Here, we also take the effect of the iterative writing into account. As mentioned earlier, cell programming iterates a write-and-verify sequence until $\log_{10} R$ is less than $\mu_R + 2.75\sigma_R$ or larger than $\mu_R - 2.75\sigma_R$. Therefore, the probability density function of a random variable m, $f(m)$ can be expressed as

$$f(m) = \begin{cases} \frac{1}{K}\phi(\frac{m-\mu_R}{\sigma_R}) & \mu_R - 2.75\sigma_R < m < \mu_R + 2.75\sigma_R \\ 0 & \text{otherwise,} \end{cases}$$
$$\text{where } K = \int_{\mu_R + 2.75\sigma_R}^{\mu_R - 2.75\sigma_R} \phi(\frac{m - \mu_R}{\sigma_R}) dm,$$
$$\text{and } \phi(x) = \frac{1}{\sqrt{2\pi}} e^{-x^2/2}.$$

Therefore, we can obtain the probability of soft error as a function of time ($t = 10^n$) by integrating Equation (6) with a random variable m for $\mu_R - 2.75\sigma_R < m < \mu_R + 2.75\sigma_R$.

Probability of soft error is

$$= \int_{\mu_R + 2.75\sigma_R}^{\mu_R - 2.75\sigma_R} (1 - \Phi(\frac{\mu_R + E - m - n\mu_\alpha}{n\sigma_\alpha})) f(m) dm. \quad (7)$$

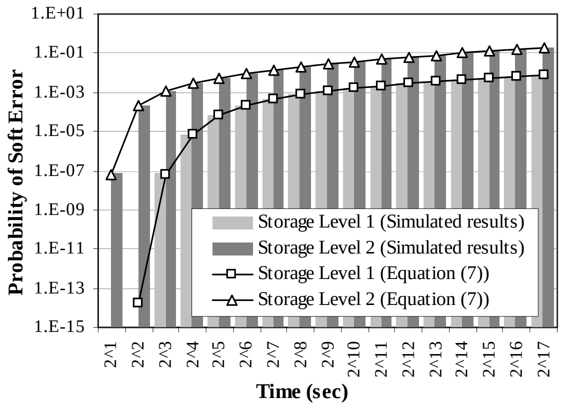


Figure 7: Probability of Soft Error of Four-Level-Cell (4LC) PCM Over Time

We evaluate Equation (7) for 4LC PCM and also run Monte Carlo simulations to verify these equations. In the simulation, we randomly picked R and α from their corresponding normal distributions in Table 1 and calculate the drift resistance, $R_{drift}(t)$, to determine if it generates any soft error. For each storage level, the simulator executes one billion trials. Figure 7 shows the results side by side. We omit the soft error rates for set and reset states, *i.e.,* , the storage level 0 and 3, because (i) resistance drift in level-3 states does not lead to a soft error, and (ii) the error rates of level-0 states are too small to be evaluated and can be ignored. For example, Mathematica 8.0 shows the first non-zero error rate for level-0 states when $t = 2^{35}$ or 1090 years, and the error rate is 2.3×10^{-18}. Similarly, note that three data points for storage level 1 and 2 are missing because either (i) Mathematica 8.0 returns zero for Equation (7) or (ii) Monte Carlo simulation found no error in one billion trials. By comparing results from two independent sources, we validate the accuracy of our theoretically derived Equation (7) by simulation.