Practical Nonvolatile Multilevel-Cell Phase Change Memory

Doe Hyun Yoon*
IBM Thomas J. Watson Research Center
dyoon@us.ibm.com

Jichuan Chang
Hewlett-Packard Labs
jichuan.chang@hp.com

Robert S. Schreiber
Hewlett-Packard Labs
rob.schreiber@hp.com

Norman P. Jouppi
Hewlett-Packard Labs
norm.jouppi@hp.com

ABSTRACT

Multilevel-cell (MLC) phase change memory (PCM) may provide both high capacity main memory and faster-than-Flash persistent storage. But slow growth in cell resistance with time, *resistance drift*, can cause transient errors in MLC-PCM. Drift errors increase with time, and prior work suggests refresh before the cell loses data. The need for refresh makes MLC-PCM volatile, taking away a key advantage. Based on the observation that most drift errors occur in a particular state in four-level-cell PCM, we propose to change from four levels to three levels, eliminating the most vulnerable state. This simple change lowers cell drift error rates by many orders of magnitude: three-level-cell PCM can retain data without power for more than ten years. With optimized encoding/decoding and a wearout tolerance mechanism, we can narrow the capacity gap between three-level and four-level cells. These techniques together enable low-cost, high-performance, genuinely nonvolatile MLC-PCM.

Categories and Subject Descriptors

B.3.4 [**MEMORY STRUCTURES**]: Reliability, Testing, and Fault-Tolerance—*Error-checking*

General Terms

Design, Reliability

Keywords

Memory, Phase Change, Multilevel Cell, Nonvolatility

1 Phase-Change Memory for Exascale Systems

The cost, power, density, and reliability of memory will be critically important for exascale systems. While DRAM is not out of the question, sustaining exponential growth (a.k.a. Moore's law) in DRAM capacity is becoming more and more difficult, and researchers have proposed alternatives, including *phase change memory* (PCM), as a scalable substitute. PCM provides these advantages: low (almost no) idle power, robustness against particle-induced (soft) errors, nonvolatility, and *multilevel-cell* (MLC) capability for high density. MLC-PCM has higher storage density than either DRAM or *single-level-cell* (SLC)[1] PCM, allowing MLC-PCM to compete with Flash. Its nonvolatility makes it suitable for important HPC (high-performance computing) uses such as checkpoint files, in situ postprocessing, and other uses.

MLC-PCM, however, has some challenges. Prior work has focused on slow writes [25], wear leveling [26], and hard error correction [27, 28, 39, 24]. Here we address another important complicating characteristic. A PCM cell is a controllable resistor whose resistance can be set at one of several levels, but the resistance slowly changes over time. This is called *resistance drift*.

Drift errors and refresh: Resistance drift may cause errors. Once a cell is programmed to a certain state, the cell resistance increases over time. The resistance may drift into the next state region, and then the sensing circuit will read a different value than the one that was written. This phenomenon is analogous to charge leakage in a DRAM cell. A cell retains its data only for a finite amount of time. An intuitive solution is to read and rewrite each cell before it loses its data—periodic refresh for MLC-PCM [2].

A cell with faster than average drift, or which is written at the high end of its designated resistance state, and which is not refreshed often enough, will produce transient drift-induced errors. (We use the term *transient error* in a memory cell to mean an error that, while it may not recur repeatedly and reliably, is more likely to be seen in the future having been seen in the cell once.)

The purpose of refresh is to restore cells to nominal resistance values in order to prevent drift-induced errors. We use an error correcting code (ECC) as part of the refresh mechanism [2]. The ECC mechanism first corrects any drift-induced (and other) errors. Thus, for every cell, at least once per refresh period, we read, correct if needed, and re-write (even if no ECC-corrected error occurred). The refresh resembles a DRAM scrubbing mechanism, but its main purpose and effect is to restore nominal analog resistance values. Adding the ECC helps to prolong the refresh period.

Nonvolatility: We define a memory or storage device to be nonvolatile if it retains its contents for a long time, measured in years, without consuming power. Any memory that needs frequent refresh is therefore volatile.

Arguably, no storage medium is categorically nonvolatile—all need to be re-read and re-written at some frequency to ensure *permanent* retention of the data. They differ in the frequencies with

*This work was done while the author was at Hewlett-Packard Labs.

[1] A single-level cell has multiple, in fact two, levels, so the term single-level cell should instead be two-level, or single-bit cell. But this misuse is now well established.

SC '13 November 17 - 21, 2013, USA

http://dx.doi.org/10.1145/2503210.2503221

which refresh needs to be done. DRAM, for example, requires very frequent refresh to avoid errors. Disk can retain data for years, but not forever. Our practical definition of nonvolatility implies years of data retention without power supply.

Is MLC-PCM nonvolatile?: According to published drift models, a typical four-level-cell (4LC) PCM needs refresh every tens of minutes, even with some optimizations we discuss later. Unless we reduce drift error rates by several orders of magnitude, 4LC-PCM is therefore volatile. Applications that require nonvolatility (high-bandwidth file systems [9], persistent data structures [8, 34, 33], in-memory checkpointing [11], *etc*.) become problematic if not impossible in this scenario. Our key contribution is to restore nonvolatility at modest cost in capacity. We do this by backing off to a three-level cell.

Making 4LC-PCM usable as volatile memory: A 4LC-PCM with periodic refresh may potentially be used as volatile memory; the 4LC cell has a twofold density advantage *vis a vis* DRAM and SLC-PCM. Unfortunately, a recent study [38] showed that a naïve 4LC cell has resistance drift vulnerabilities; with a practical ECC scheme used to refresh, the refresh interval is too short, leaving too little bandwidth for application uses. This study therefore claimed that the required frequent refresh renders 4LC-PCM impractical, even as volatile memory.

Based on our study of error rates, we concur that published drift models imply that naïve 4LC-PCM is impractical. But we show how 4LC-PCM can be made useful. We improve it by a more careful data encoding, a strong but realizable ECC, and a better choice of the resistance levels to be used. This lengthens its required refresh interval to a practical value (17 minutes).

Restoring nonvolatility: Further increase in retention time (to more than a year) can qualify MLC-PCM as nonvolatile storage.

We observe that in a four-level cell, most drift errors occur in a particular cell state (the second highest resistance level); hence, we propose to change from four-level (two bits per cell) to three-level (ternary cell), avoiding the vulnerable state (and most, if not all, drift errors).

We propose three-level-cell (3LC) designs, which achieve several orders of magnitude lower drift error rates than the best four-level-cell designs. Using a simple ECC, refresh is needed only after ten years, which makes 3LC-PCM nonvolatile by our definition.

This approach works because drift is, in a sense, self-limiting. The rate of drift decreases monotonically with time. Thus, it is practical to control drift errors simply by widening the safety margins between cell states.

The use of ternary cells naturally raises two question: how to store binary information in the non-power-of-two-level cells and how to compensate for the reduced information capacity.

We propose to use an encoding mechanism (*3-ON-2*) that stores three bits on two ternary cells, similar to the ones in [18, 29]. This only achieves 1.5 bits per cell, which is 25% lower than in the four-level designs. We propose a novel, low-storage-overhead wearout tolerance mechanism for the 3-ON-2 design (*mark-and-spare*). Including the storage overhead of error correcting information (for both wearout failures and drift errors), the proposed 3-ON-2 mechanism is only 7.4% less dense than a four-level-cell design. The full 3-ON-2 scheme has other practical advantages:

- low read latency with simple ECC ($8\times$ faster ECC decoding than the four-level-cell design),
- low-storage-overhead wearout tolerance with mark-and-spare (only two spare cells per wear-out failure, compared to the five cells per wearout failure of prior work),

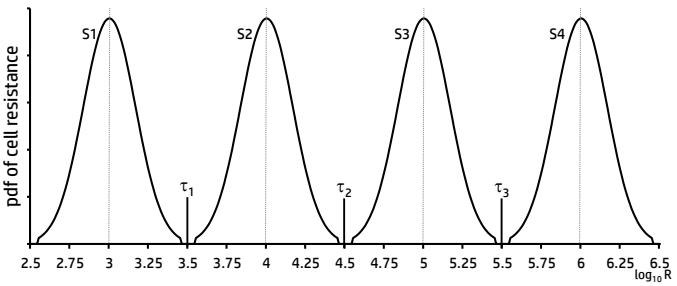


Figure 1: State mapping in a 4-level cell.

- high performance and low energy (33% higher performance and 24% lower energy than the 4LC design), and
- long retention time (tens of years), to simplify data management and enable new applications (e.g., in-memory checkpointing, database, file system).

2 PCM and Resistance Drift Models

2.1 Phase change memory

PCM cells are chalcogenide materials (e.g., alloys of germanium, antimony, or tellurium: $\mathrm{Ge_2Sb_2Te_5}$, GeSb, or $\mathrm{Sb_2Te_3}$). A cell has a continuum of states of which the extremes are amorphous and fully crystalline. Resistance in the fully crystalline and amorphous states differs by 3-4 orders of magnitude [6]. A two-level, one bit cell would use only these extreme states.

A *SET* operation applies a low-amplitude, long current pulse, which forces the cell into the crystalline state. A *RESET* operation uses a high-amplitude, short current pulse, which forces the cell into the amorphous state.

2.2 Multilevel-cell phase change memory

Like Flash, PCM derives some of its cost advantages from memory cells that store more than one bit of information. A sequence of precisely controlled SET and RESET operations adjusts the fraction of crystalline material in a cell and can set the resistance to an intermediate value. Though some recent material papers present the evidence of 8- or 16-level cells [23], here we assume a four-level cell as the baseline MLC (with expected usage in the near future).

In MLC-PCM, the actual resistance of a written cell is a random variable whose distribution is lognormal [19, 4]: the logarithm of the resistance of the written cell is normally distributed, with mean at or close to a nominal value (μ_R), and with some standard deviation (σ_R). Figure 1 shows the resistance distribution in four-level cell PCM. We denote the four cell states *S1*, *S2*, *S3*, and *S4*, from lowest to highest resistance. In a naïve encoding, the nominal values of these states are evenly distributed in the log-resistance domain, and so are the threshold values (τ_1 to τ_3) at the inter-state boundaries. A write is accepted if it falls within $2.75 \times \sigma_R$ of the nominal value; otherwise, the cell is re-written (*iterative write-and-verify* [23]). Thus the density of the written cell's log-resistance lies under truncated Gaussians.

2.3 Resistance drift

Unfortunately, cell resistance is not constant over time; it drifts. Let the cell be programmed at time $t = 0$, and let the cell resistance be sensed as R_0 after a very small amount of time, t_0. Then $R(t)$, the cell resistance at time t ($t > t_0$), is modeled as Equation 1 [17].

Table 1: MLC-PCM resistance and drift parameters [37].

	state	$\log_{10} R$		α	
		μ_R	σ_R	μ_α	σ_α
Lowest resistance	S1	3	1/6	0.001	$0.4 \times \mu_\alpha$
	S2	4		0.02	
	S3	5		0.06	
Highest resistance	S4	6		0.1	

$$R(t) = R_0 \times \left(\frac{t}{t_0}\right)^{\alpha} \quad (1)$$

The exponent α determines the drift rate. Due to process variation, every cell experiences different drift rates. The drift rate α, hence, is modeled as a random variable; the mean (μ_α) and standard deviation (σ_α) are very small at S1 (lowest resistance) and increase with the cell resistance. Table 1 describes μ_R and σ_R of the four cell states as well as μ_α and σ_α [37]. Since $\alpha < 1$, the rate of resistance drift, dR/dt, is monotonically decreasing; moreover, the quantity of interest, which is $\log R$ grows as $\log t$, so the growth rate of log resistance drops dramatically.

2.4 Transient errors caused by resistance drift

We illustrate how resistance drift causes transient errors in Figure 2. When a cell is programmed to the S2 state, for example, its resistance lies initially within $\pm 2.75\sigma_R$ of the nominal value ($10^4\Omega$). It then begins to drift upwards.

Resistance drift does not cause a logical error until the increased resistance crosses a threshold into the next state. The time to drift errors (the *retention time*), is determined by two factors: the initial resistance R_0 and the drift rate α. Due to variability of the initial resistance, cells programmed to relatively low resistance (e.g., 1 in Figure 2) retain the stored value longer, cells programmed to relatively high resistance (e.g., 2 and 3 in Figure 2) cause transient errors more quickly. And cells with higher than average drift rate (α) will also suffer errors more quickly.

As discussed, the drift rate α is very small in S1; the infinitesimal drift essentially never changes an S1 state into an S2 state. The drift rate is higher in S2 and even higher in S3. However, the highest-resistance state (S4) does not suffer from drift errors. Increasing the resistance in S4 cannot change the cell state: any resistance higher than τ_3 is treated as S4. Hence, drift errors occur only in intermediate states (S2 and S3), and when states are uniformly spaced in log-resistance, S3 is most prone to drift errors.

We first estimate the naïve four-level-cell PCM (4LC^n) cell error rate per refresh period as a function of the refresh interval. Because of the variability of initial resistance and drift rate, we take $N = 10^9$ samples from these distributions, and find the fraction of cells in which resistance drifts across a state threshold in the given refresh interval. (Our methodology is similar to that of [38].) Figure 3 shows cell error rates of states S2 and S3. As discussed, cell error rates of S1 and S4 are practically zero. Note that S3 has approximately an order of magnitude higher cell error rates than S2.

3 Related Work

There is little prior work on drift-induced transient errors. We first briefly review the work on mitigating resistance drift, then discuss wearout tolerance techniques and other topics.

Prior work on resistance drift: Reference cell [16] and time-aware sensing [37] are circuit-level techniques for mitigating drift

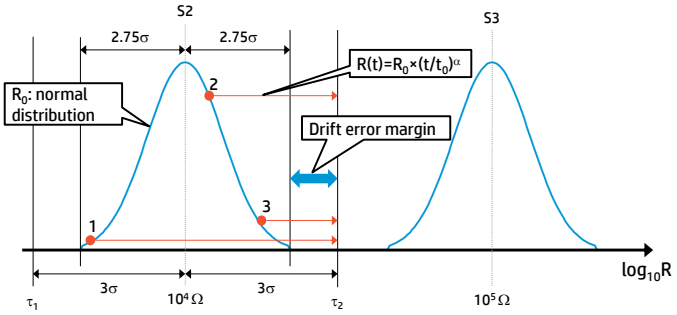


Figure 2: Transient errors due to resistance drift.

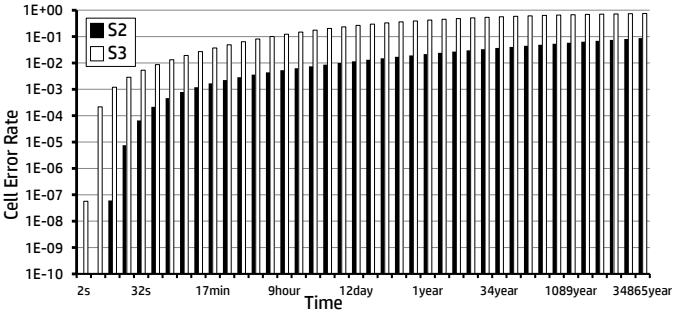


Figure 3: Drift error rates in a conventional four-level cell.

errors. These complementary drift error reduction techniques show limited improvement in error rate.

Linear, systematic, block ECC are widely used for correcting errors in current memory devices. A BCH (Bose Chaudhuri Hocquenghem) [5, 14] code is useful for correcting drift errors, with its capability for detecting and correcting random bit errors. We use the notation BCH-n for an n-bit-correcting BCH code.

Permutation coding [22] encodes data through permuting the cell resistance levels in a sequence of cells that, before permutation, have monotonically increasing resistance. Data are preserved if the permutation does not change due to drift. While this coding technology is more drift-resilient than level-based encoding, the decoding procedure is complex, involving analog sensing of resistance values, sorting, finding the most likely basic pattern, permutation, and a table lookup. The specific coding scheme stores 11 bits in 7 memory cells, achieving 1.57 bits per cell, and the cell error rate is kept as low as $1E-5$ for more than 37 days.

Other coding techniques reduce the frequency of cells that are in vulnerable states (S2 and S3). Helmet uses selective inversion and rotation to reduce the number of S3 cells [40]. Symbol-based value encoding also makes S3 less common [35]. These mechanisms depend on data value locality; random signals and compressed or encrypted data may defeat them.

Refresh for MLC-PCM was suggested in [2], where the term used was scrubbing, rather than refresh.

Our work is based on the analytical resistance drift model of [38]. The authors of that study recently proposed *tri-level cell* PCM [29] to mitigate drift errors. This work is very close to ours in that it uses three-level cells and stores three bits in two ternary cells. Our work, however, discusses more than just using three-level cells for drift error tolerance. We address wearout failures and propose a low-cost wearout tolerance mechanism (mark-and-spare), leveraging the unused state in 3-ON-2. We qualitatively compare our work with tri-level cell PCM in Section 6.7.

Wearout failure tolerance: PCM errors are caused not only by resistance drift but also by cell wearout. In fact, MLC-PCM has lower endurance than SLC-PCM, and cell wearout is a serious problem. Most prior proposals on wearout have focused on SLC-PCM: Error Correcting Pointers (ECP) [27], FREE-p [39], PAYG [24], SAFER [28], *etc*. In this work, we extend ECP to support MLC-PCM. FREE-p and PAYG are also applicable to MLC-PCM, but applying SAFER to MLC-PCM is not straightforward.

ZombieMLC [3] proposes a wearout tolerance mechanism for permutation-coded MLC PCM. It prepends anchor cells, whose values are known, to a permutation-coded message and permutes it again to map '0' and '1' on the stuck-at-0 and -1 cells, respectively; hence, it tolerates wearout failures. While ZombieMLC is an interesting technique, it incurs relatively complex decoding procedures (solving a linear equation using Galois field arithmetic, in addition to already not-so-simple permutation decoding), and the presented examples with four-level cells have very low information density: 1.33 and 1 bits per cell (using four-level cells).

Non-power-of-two-level cells: Like our 3LC proposal, elastic RESET [18] uses three-level cells, but for a different purpose (longer lifetime). Elastic RESET uses a uniform state mapping and does not achieve the same low error rate that we do.

The information encoding in our paper as well as elastic RESET can be viewed as a special case of *enumerative source encoding* [10]. Such an advanced coding theory can help design future MLC memory with any non-power-of-two levels.

4 Target Refresh Interval and Cell Error Rate

Before studying drift error rates and proposing a new architecture, we should first answer the following questions: What retention time is acceptable in volatile memory? What cell error rate is tolerable?

4.1 What PCM memory refresh interval is acceptable?

To guarantee long-term data integrity, every block in an MLC-PCM device must be refreshed before any block fails.

PCM write is slow (around 100 ns), and MLC-PCM write is even slower (around 1 us). Assuming each refresh of a 64B block takes 1 us, refreshing a 16GB device takes around 268 s; at a refresh interval of 17 minutes, the PCM device is available only 74% of the time. Figure 4 shows PCM availability in a function of refresh interval. This large penalty can be alleviated by refreshing each memory bank independently. While a bank is being refreshed, other banks are available to users. For instance, bank availability can be as high as 97% in an 8-bank PCM device at the 17-minute refresh interval.

The refresh rate, however, cannot be arbitrarily high. PCM write operations draw considerable power, limiting PCM write throughput and hence refresh rate. A recent 8Gb SLC-PCM prototype has 3.2 GB/s read bandwidth, but write bandwidth is only 40MB/s, after a dramatic $5\times$ improvement [7]. MLC-PCM write may draw more current than SLC-PCM write, and write throughput is likely to be lower than SLC-PCM write throughput. Aggressively assuming 40MB/s write throughput, refreshing a 16GB MLC-PCM takes around 410 s; the refresh interval should be much longer than this time.

Considering both bank availability and write throughput, we use 17 minutes (2^{10} s) as an acceptable refresh interval: This interval is over twice the time to complete one pass of refresh (410 s) so that the processor can still write back dirty cache lines, and bank availability is reasonably high (97%) so that the processor can easily read data.

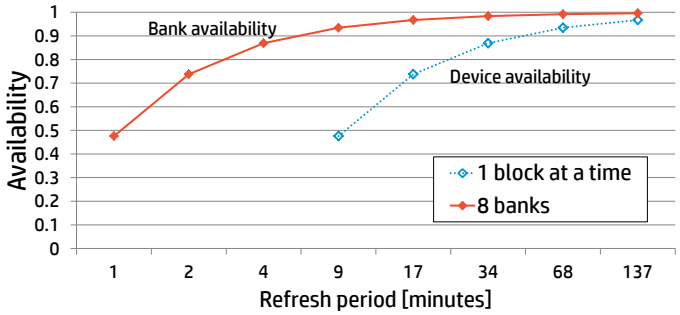


Figure 4: PCM availability as a function of refresh interval.

4.2 What cell error rate is tolerable?

While the actual target cell error rates are available only to the process engineers of memory vendors, we set a conservative goal in this study: fewer than one erroneous block per MLC-PCM device for more than ten years; i.e., device mean time between failures (MTBF) is more than 10 years.

We refresh blocks periodically, correcting some cell errors with an ECC in each such period. A block becomes erroneous at the time of refresh if the block has a larger number of cell errors than the associated ECC can correct. Hence, the block error rate (BLER) is a function of cell error rate (CER) and ECC, as shown in Figure 5. These rates both represent the fraction of all blocks or cells erroneous at the end of the refresh period, and they are functions of the refresh interval. (The dependence of CER on refresh interval is given in Section 5.3.) Figure 5 shows the per-period BLER as a function of the CER and ECC.

Once a block is written, the CER and BLER monotonically increase until the block is re-written or refreshed. Over the ten year period, the block's BLER periodically reaches its maximum. The cumulative BLER, giving the per block error probability after ten years, is the cumulative probability of error over all the refresh periods in the ten year window. To meet the reliability goal, the per-period BLER should be lower than a threshold, the target BLER, that is inversely proportional to the number of refresh events, or proportional to the refresh interval, as shown by the three dotted lines in Figure 5.

An *N*-byte PCM device has *N/M M*-byte blocks, and BLER of *M/N* is equal to one erroneous block per device. Since PCM is a potential substitute for DRAM, we assume 64B blocks and 16GB devices in this study, giving a target cumulative BLER of $3.73E-9$, which corresponds to the uppermost dotted line in the figure. The two lower dotted lines give the target per block, per period BLER, assuming refresh intervals of one year and of 17 minutes.

What does this figure imply? If we want true nonvolatility, we need to remain below the upper dotted line in BLER per period. To see what ECC is needed, we first find the CER with ten-year refresh interval via our Monte Carlo technique based on the resistance drift model and the probability density for resistance and drift parameters. This gives the CER with ten-year refresh. We then find the point at which that particular CER curve (in Figure 5) drops below the dotted line. The BCH code required lies below that point. For memory use, with 17 minute refresh, the target BLER per period is the lowest dotted line. We would re-determine the CER per period with 17 minute refresh, and again find the BCH code at which the new CER curve drops below the 17-minute BLER line.

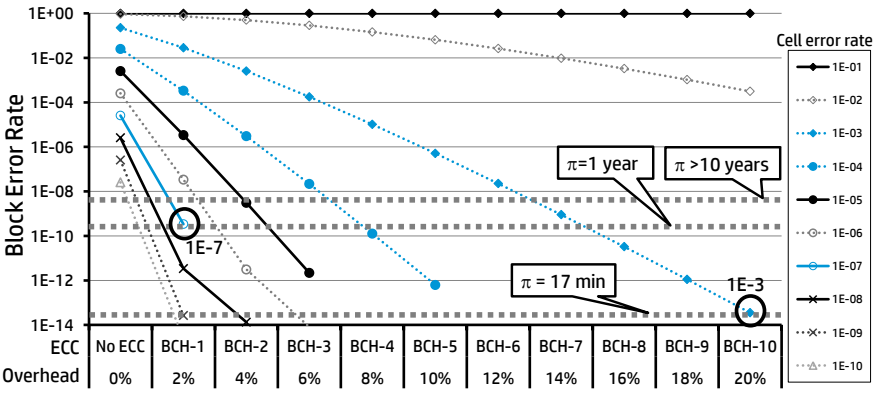


Figure 5: Block error rate as a function of cell error rate (2 bits per cell) and ECC. π is a refresh interval. Dotted horizontal lines represent target BLERs when refresh intervals are >10 years, 1 year, and 17 minutes.

5 Drift Error Resilient MLC-PCM

Given high drift rates, managing drift-induced errors is key to enabling practical MLC-PCM. In some applications, hours to months of retention time is good enough. However, PCM's unique advantage is nonvolatility with byte-addressable read/write; hence, enabling years of retention has numerous practical advantages in real systems.

In order to mitigate drift errors, we first apply a set of optimization techniques to the conventional four-level-cell (4LC) design. We then propose a new three-level cell (3LC) design that substantially reduces drift error rates.

5.1 A resilient four-level-cell design

As shown in Figure 3, only two states cause drift errors in the conventional 4LC design. Optimization techniques that leverage asymmetry in information statistics can reduce drift errors; we apply two techniques to the conventional 4LC design (4LC^n).

Smart cell encoding: We estimate cell error rate in 4LC^n assuming equal probability of occurrence, 25% in each state. Since the error rates in S3 is an order of magnitude higher than that of S2 or others, the cell error rate is roughly 25% of the cell error rate for cells in S3.

Often, however, some cell states occur less frequently than the others. A smart encoding system reduces the number of cells programmed to vulnerable states (S2 and S3). Both Helmet [40] and symbol-based value encoding in [35] do this.

Clearly, the effectiveness of this technique depends on having a nonuniform probability of state occurrences. In this study, we assume a skewed probability of occurrence: 35% for S1 and S4, and 15% for S2 and S3. This is the scheme (4LC^s). We believe the probability of occurrence in 4LC^s is quite optimistic compared to prior work [40, 35].

Optimal state mapping: The cell resistance distributions shown in Figure 1 use a naïve, uniform distribution of nominal values and thresholds in the log-resistance domain. Given that drift only increases cell resistance and drift error rates are much higher in S3 than the other states, re-assigning each state's nominal resistance and thresholds can mitigate the problem.

For the optimization, we use the following notations: μ_n denotes the nominal value of state Sn (n is 1, 2, 3, or 4); τ_n is a threshold value between state Sn and Sn+1; σ is standard deviation for resistance distribution; and δ is minimum margin between a threshold and a (left or right) tail of resistance distribution (hence, left and right tails of two distributions are spaced by at least $2 \times \delta$ apart).

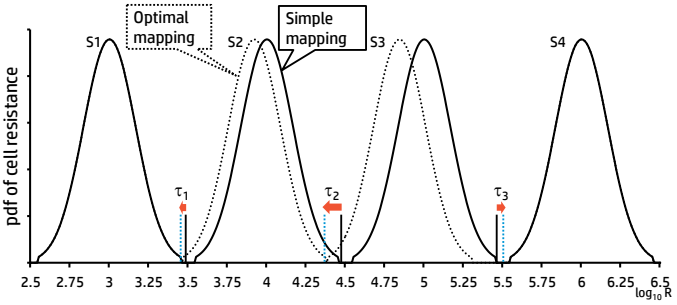


Figure 6: Four-level cell: simple and optimal mapping.

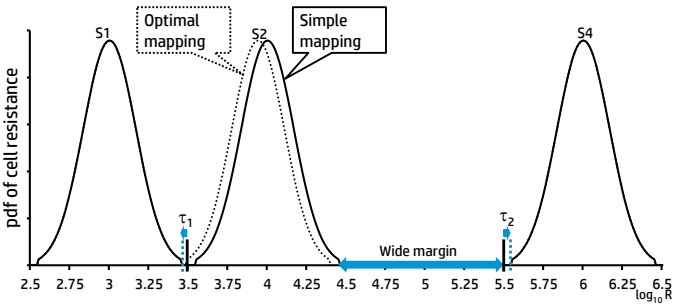


Figure 7: The proposed three-level cell: simple and optimal mapping.

δ is a guard band for sense amplifier noise, possible long-term drift that may decrease resistance very slowly, and other faults. We use a very small δ (0.05σ) in this study. Note that real systems may have additional constraints.

First, we fix μ_1 and μ_4 to $10^3\Omega$ and $10^6\Omega$, respectively, since the resistance of fully crystalline and amorphous states is determined by process technology. We then optimize μ_2, μ_3, and τ_1 to τ_3 as follows.

$$\begin{array}{ll} \text{minimize} & CER(\mu_2, \mu_3, \tau_1, \tau_2, \tau_3), \\ \text{subject to} & \mu_i + 2.75\sigma + \delta < \tau_i < \mu_{i+1} - 2.75\sigma - \delta, \\ \text{for} & i = 1, 2, 3. \end{array}$$

The objective function $CER()$ estimates cell error rate at time $t = 2^{15}$ s using Monte Carlo simulation (we use 10^6 cells for faster evaluation). We use 2^{15} s as an example time to evaluate drift error rates; since drift error rates monotonically increase over time,

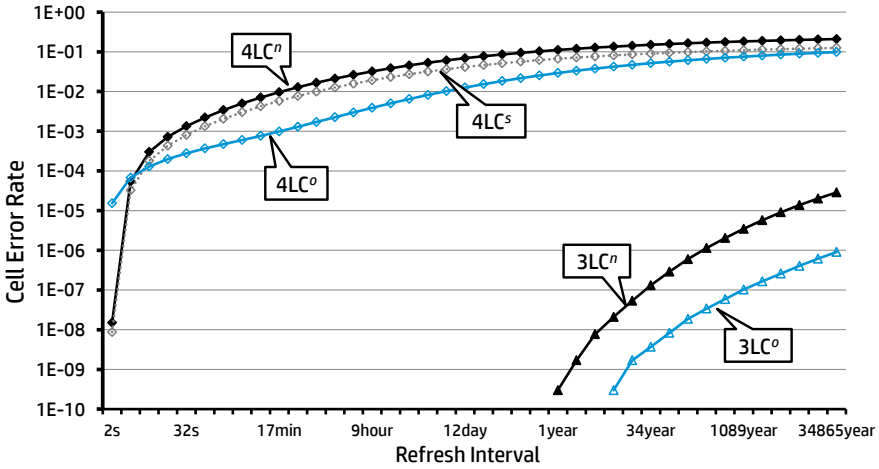


Figure 8: Cell drift error rates: four-level cell and three-level cell.

the optimization result is not sensitive to the choice of this value. The constraints in the optimization procedure are due to the relative order among μ_n and τ_n.

Figure 6 compares the simple mapping and the resulting optimal mapping. The nominal resistance values of S2 and S3 are shifted left, and the threshold value between S3 and S4 is shifted right. As a result, the drift error margin for S3 (the gap between the right tail of S3 and the threshold for S4) is widened significantly. We refer the 4LC design with both the optimal mapping and a smart encoding to as 4LC^o. Because the excess vulnerability of S3 has largely been removed, the smart data encoding adds to resilience only marginally.

Limitations: We show the achieved cell error rates from the described optimizations in 4LC-PCM in Section 5.3. These techniques (4LC^s and 4LC^o), however, are not a game changer; they achieve a modest reduction in drift errors and allow some increase in the refresh interval. With strong ECC, use of 4LC^o as volatile memory becomes possible.

In order to make MLC-PCM nonvolatile, we must reduce drift error rates by several orders of magnitude. Because $\log R$ grows only as $\log t$, we can substantially prolong the time-to-error by widening the gap between states.

5.2 Three-level cell design

We propose a three-level cell (3LC) design. Even with optimized mapping, 4LC packs too many levels into the available log-resistance range. Our proposal instead reduces the pressure by backing off to three levels: a ternary cell. Figure 7 illustrates both a simple state mapping (3LC^n) and an optimal mapping (3LC^o).

3LC^n just removes S3 from the naïve 4LC mapping (Figure 1). For 3LC^o we used the optimization described in the previous subsection. Note that we denote the three states S1, S2, and S4 (not S3, because this state is basically equal to the S4 in Figure 1).

This simple change dramatically reduces the drift-caused transient error rate. Without S3, S2 is the only state that causes drift errors. In the proposed 3LC design, S2 has a wide safety margin, as shown in Figure 7. A cell programmed to S2 takes a long time to drift into the S4 region.

For simplicity, we do not use any smart encoding technique for the 3LC designs—we assume that all three states are equally frequent.

5.3 Drift error rates of 4LC and 3LC

We compare the per period cell error rates of 4LC and 3LC in Figure 8, using Monte Carlo simulation of 10^9 cells. For 3LC designs, when the resistance of an S2 cell drifts and reaches $10^{4.5}\Omega$ (the original τ_2 in 4LC^n), we apply a different drift rate (using S3's drift rate parameters: $\mu_\alpha = 0.06$). This is to conservatively account for the effect of possible increase in drift rate as the resistance increases.

Four-level cell: 4LC^n is practically useless due to high error rates. The cell error rate is $1E-3$ at a very frequent refresh interval of 30 s, but which is impractical—the device would be almost always busy for refresh and unavailable to users. At the more practical refresh interval of 17 minutes or longer, the cell error rates are too high ($> 1E-2$). At this error rate, nearly every block will have uncorrectable errors even with impractically strong ECC (say, a 20-bit correcting code). This result corroborates the pessimistic estimates in [38];

Smart encoding (4LC^s) lowers the error rate curve, but this technique alone is not sufficient. The curve may further lower depending on the value locality of input data, but equally, it may go up. Our assumption of only 15% for S2 and S3 is already optimistic compared to the reported statistics in prior work [40, 35].

The optimal mapping (4LC^o) achieves approximately an order of magnitude lower cell error rates than those of 4LC^n. For the initial four seconds, 4LC^o experiences higher cell error rates than those of 4LC^n, mainly due to errors in the S1 state; with S2 squeezed closer to S1, S1 has reduced drift error margin and may cause some errors. But, after four seconds 4LC^o achieves much lower cell error rates than 4LC^n. The cell error rate at 17-minute retention time is around $1E-3$, reaching a barely feasible level for a real system: a strong, but realizable, 10-bit correcting ECC (BCH-10) can keep the BLER lower than the target ($1.20E-14$), appropriate for a 17-minute refresh interval.

Proposed three-level cell: The 3LC designs achieve orders of magnitude lower cell error rates than 4LC. Even a simple mapping (3LC^n) has negligible cell error rate until one year. Consider the optimal mapping (3LC^o). Its error-free period exceeds 16 years, so for ten-year nonvolatility it needs no ECC at all (according to the drift error models). It would need no ECC if used as volatile memory; nor would it need refresh unless one envisages an application running for more than 16 years non-stop! For longer data retention, note that it achieves cell error rate of $1E-8$ even after 68

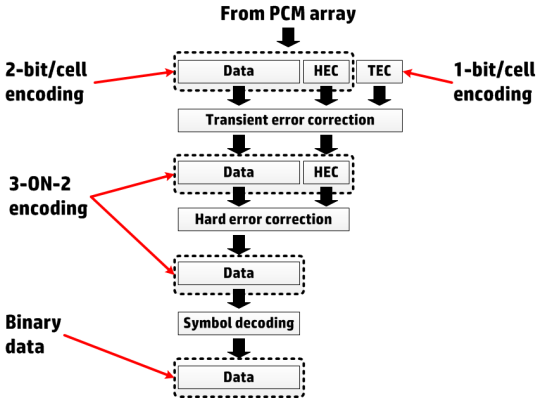
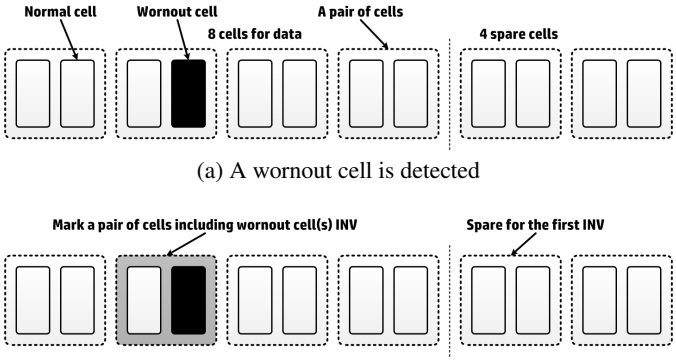


Figure 9: Read data path in the proposed PCM architecture. HEC is hard error correction information, and TEC is transient error correction information.

Table 2: Example 3-ON-2 encoding

First cell state	second cell state	3-bit data
S1	S1	000
S1	S2	001
S1	S4	010
S2	S1	011
S2	S2	100
S2	S4	101
S4	S1	110
S4	S2	111
S4	S4	INV



(a) A wornout cell is detected

(b) A pair with a failure is marked as INV, and a spare is used instead

Figure 10: Mark-and-spare example with eight data cells and four spare cells. A real system will have 342 data cells and 12 spare cells for tolerating six wearout failures.

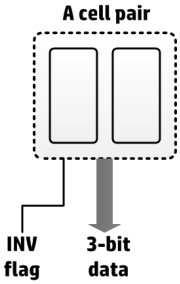


Figure 11: A cell pair with an INV flag and three-bit data.

years, and at this rate, a simple, one-bit correcting ECC (BCH-1) is enough.

In summary, the proposed (3LC) design is simple, yet powerful. It effectively avoids drift errors. Unlike the 4LC designs that demand strong ECC and frequent refresh, the 3LC designs allow simple or no ECC and can store data reliably for years without power. The caveats for the 3LC designs are (i) storing binary information in ternary cells is tricky, and (ii) storage density is reduced.

6 Three-Level-Cell Phase Change Memory

In this section, we develop an information encoding scheme for the proposed 3LC^o. We store three bits of information in a pair of ternary cells. A pair of cells has nine states, so there is one state in addition to the eight used to encode three bits. Using this remaining state, we design a low-storage-overhead mechanism for the proposed 3LC^o to tolerate wearout failures. We then qualitatively compare the proposed 3LC^o to other designs.

6.1 High-level design

Figure 9 illustrates how to retrieve data in our proposal. We do not show a write-back path since it is the reverse of data retrieval. The data retrieval is composed of several steps: a PCM array read, transient error correction, hard error correction, and symbol decoding.

A PCM array stores information for hard error correction and transient error correction (HEC and TEC in Figure 9) along with data. A read operation (or row activation in current DRAM terminology) fetches data and its associated HEC and TEC from the PCM array.

A drift error can occur on any memory cell, including those that store hard error correction information. Without protection from drift errors, the hard error correction procedure may use incorrect information and may corrupt healthy memory cells (instead of correcting failed memory cells). Hence, we need to correct transient errors before hard error correction.

We use different encoding schemes for transient error correction and hard error correction (described later in detail). The final step is symbol decoding to construct binary information before sending the data to the requester (e.g., a memory controller).

For main memory, we assume a 64B data block as an access unit (hence, data / ECC encoding uses this block size). We also expect that the hard error correction mechanism should be able to tolerate up to six wearout failures per block [27].

6.2 Information encoding

We store three bits in two ternary cells—*3-ON-2*. This mechanism has information capacity of 1.5 bits per cell (close to the ideal capacity of ternary cells—1.58 bits per cell). A 64B data block is stored in 342 cells.

An example 3-ON-2 encoding is shown in Table 2; nine different states from two ternary cells encode three bits (in eight valid states) and an invalid state (*INV*). We leverage the 9th (INV) state for tolerating wearout failures.

Different encodings are possible for the eight valid states: e.g., 3-bit Gray coding, or encoding schemes shown in [18]. The INV state, however, should be the cell state [S4, S4] (highest resistance in both cells), since we leverage this particular state for handling wearout failures as discussed later in this section.

A caveat is that a drift error may turn a valid state into the INV state. As we discuss later, we use the INV state to mark a cell pair with failures, but an unexpected transition from a valid state to the

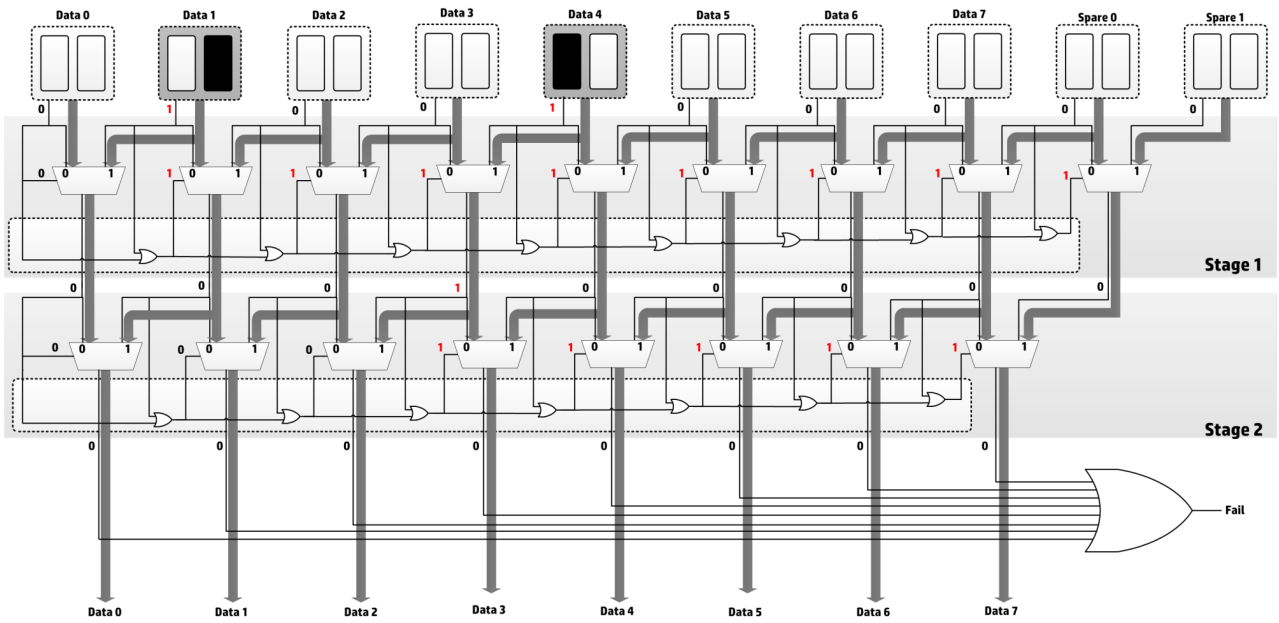


Figure 12: Mark-and-spare correction. Each stage replace a cell pair with failure(s) with a spare pair. At stage 1, the second to the last MUXes choose the right input, ignoring Data 1. At stage 2, the fourth to the last MUXes choose the right input, ignoring Data 4. Two spare pairs are used to replace faulty pairs, Data 1 and Data 4. Note that MUX select signals are autonomously generated from INV flags and OR-gate chains.

INV state will break the proposed wearout tolerance mechanism. To avoid this, we apply transient error correction before correcting hard errors as shown in Figure 9. Transient error correction for 3-ON-2 must be able to detect and correct drift errors that convert a valid state to the INV state.

6.3 Transient error correction

If we extract three bits from two cells and construct ECC from these bits, we cannot represent the INV state and consequently, ECC cannot detect and correct an error that converts a valid state to the INV state. Hence, we use a different encoding scheme for detecting and correcting transient errors (due to resistance drift). For transient error correction, each cell is treated to have 2 bits – S1: 00, S2, 01, and S4: 11. As in Gray coding, a drift error can cause only 1 bit error in this encoding. Note that this encoding does not change cell states encoded for 3-ON-2. But the way cell states are interpreted by ECC logic is different.

For constructing an ECC codeword, the message length is 708 bits—2 bits per cell from 342 data cells (512 bits using 3-ON-2) and 12 spare cells for hard error correction (more details in the next subsection). The proposed 3LC has very low drift error rates, so we use a single bit-error correcting code, BCH-1 (or equivalently, a Hamming [13] or a Hsiao [15] code), that requires additional 10 check bits over a 64B block. We store the check bits in SLC mode (1 bit per cell) to prevent drift errors on the check bits.

6.4 Hard error correction

MLC-PCM has a much shorter lifetime than SLC-PCM (10^5 cycles vs. 10^8 cycles), and iterative write-after-verify will increase variation among cells. Tolerating wearout failures in MLC-PCM is therefore even more critical than in SLC-PCM.

Though a BCH code can correct both hard and transient errors, a simpler error correction mechanism dedicated to wearout failures is more desirable (e.g., ECP [27]); BCH decoding becomes very complex as the number of bits corrected increases.

We propose a cost-effective wearout tolerance mechanism for the 3-ON-2 encoding, leveraging the INV state. Figure 10(a) shows an example of eight data cells with four spare cells, using 3-ON-2. When a wearout failure is detected (typically, through write-after-verify), we mark the cell pair that includes the worn-out cell as INV, after which a spare pair is used, as shown in Figure 10(b). Figure 11 shows a cell pair with two outputs: one is an INV flag, which is true if the cell pair is the INV state, and the other is three-bit data, which is valid only if the INV flag is false.

This mechanism, *marking worn-out cells and using a spare* (or *mark-and-spare*, in short), has very low storage overhead: only two spare cells for each wearout failure. Tolerating six wearout failures requires 12 spare cells.

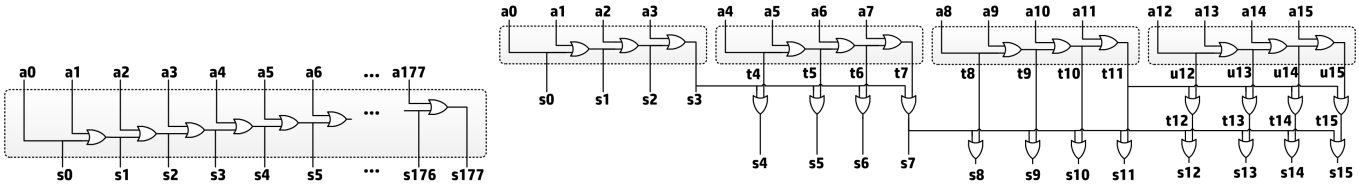
Marking of a pair of cells with failure(s) is possible since the INV state uses S4 in both cells. PCM has two failure modes: stuck-reset and stuck-set [6]. The former is a cell stuck at the highest resistance state (S4), and the latter is a cell that cannot be RESET to the highest resistance state. Fortunately, we can revive the stuck-set cell by applying a reverse current [12], forcing a stuck-set cell into S4. Even when a stuck-set cell cannot be forced into S4, the 1-bit correcting ECC can hide it. Note that we apply ECC-1 before mark-and-spare (TEC and HEC in Figure 9). If there are more non-revivable stuck-set cells than the ECC-1 can correct, we can combine the current design with fine-grained block remapping [39] to provide end-to-end protection.

Error correction using mark-and-spare: Figure 12 illustrates how to correct wearout failures using mark-and-spare. To tolerate n wearout failures, we need n correction stages. Each correction stage comprises a set of multiplexors (MUXes) and throws an INV cell pair (with failures) out.

In the example shown in Figure 12, there are eight data pairs (data 0 through data 7) and two spare pairs (spare 0 and spare 1). Two INV pairs (darkened ones, data 1 and data 4) include wornout cells (black cells).

Table 3: Qualitative comparison.

	Storage mechanism	64B Data	Wearout failure correction	Drift error correction	ECC enc/dec	Refresh period	Density
4LC^o	2 bits per cell	256 cells	ECP-6 (5 cells per failure)	BCH-10	18 / 569 [FO4]	17 minutes	1.52 bits per cell
Permutation	11 bits per 7 cells	329 cells	ECP-6 in SLC (10 cells per failure)	permutation coding and BCH-1	N/A	> 37 days	1.29 bits per cell
3-ON-2	3 bits per 2 cells	342 cells	Mark-and-spare (2 cells per failure)	BCH-1	18 / 68 [FO4]	> 68 years	1.41 bits per cell



(a) $O(n)$ design, similar to ripple carry adders (b) $O(\log(n))$ design, similar to Sklansky adders

Figure 13: OR gate chain.

At the stage 1, data 1 is thrown out, data 2 is shifted to data 1 position, data 3 to data 2, and so on. MUX select signals are generated by the OR-gate chain and INV flags; the select signals for the second to the last MUXes are set to 1. Similarly, stage 2 throws data 4 out. This procedure is similar to the bit-fix technique presented in a recent cache reliability study [36], but the mark-and-spare correction logic autonomously generates MUX select signals from INV flags.

Though this correction mechanism is simple, a long OR gate chain in each stage may take a long time to evaluate. The OR-gate chain length can be 177 gates for 64B blocks as shown in Figure 13(a): a_k is an input to the OR-gate chain, which is an INV flag from a cell pair or a MUX in the previous stage; and S_n is an output of the OR-gate chain, which is fed into a MUX to select the proper input. We can apply optimization strategies used in building fast, prefix adders since S_k is a prefix of S_n if $k < n$; e.g., $S_2 = a_0 + a_1 + a_2$, and $S_3 = a_0 + a_1 + a_2 + a_3$, but S_3 can be re-written as $S_2 + a_3$. Optimizing a long OR-gate chain using a prefix circuit is also described in [27].

Figure 13(b) shows a fast implementation of a 16-bit long OR-gate chain using a structure similar to Sklansky adder [30], reducing the delay of the OR-gate chain from $O(n)$ to $O(\log n)$. Other design styles (e.g., Kogge-Stone adder [20]) is also possible.

6.5 Storage capacity

In summary, the 3-ON-2 design uses 342 cells for 512 data bits, 12 spare cells for tolerating six wearout failures, and 10 cells for single bit-error correcting ECC. The storage density is 1.406 bits/cell, close to the data-only capacity of 1.5 bits/cell or the ideal capacity 1.58 bits/cell, thanks to the low-storage-overhead mark-and-spare and simple ECC enabled by years of retention time in 3LC.

6.6 Qualitative comparison

We compare the proposed 3-ON-2 encoding to the conventional 4LC^o design and a coding-based drift error tolerance technique (permutation coding [22]). We first briefly review encoding and error correction in prior work.

Four-level cell: The 4LC^o design (ideally) stores 2 bits per cell – only 256 cells are required for storing a 64B block. However, the 4LC^o design still needs transient and hard error correction, so the effective capacity is less than ideal.

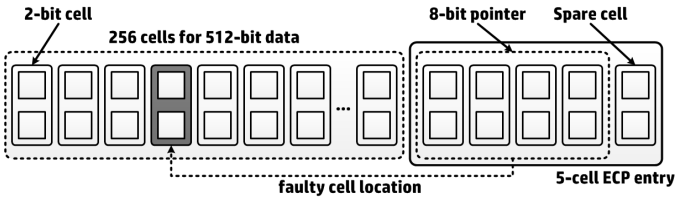


Figure 14: ECP for MLC.

Due to high cell error rates (around $1E-3$ at a 17-minute refresh period), at least 10-bit correcting BCH code is needed to keep the BLER down to practical levels. A 64B data block needs 10 check bits per transient error; so total 100 check bits are used, stored in 50 cells.

To tolerate wearout failures, we use an error correcting pointer (ECP) technique [27], a widely accepted wearout failure correction mechanism. The original ECP design is for SLC-PCM. We adapt ECP to 4LC-PCM as shown in Figure 14. An 8-bit pointer (for 256-cell data) is stored in four cells, and an additional cell is for spare; therefore an ECP entry of five cells is required for correcting a cell failure. To tolerate six wearout failures, a total of 31 cells (5×6 for six ECP entries and one additional cell for full flag) are needed.

We use Gray coding for the information stored in 4LC-PCM so that a drift error manifests as a one-bit error. Similar to the 3LC^o design, we apply transient error correction before correcting hard errors. To incorporate a smart encoding (as discussed in Section 5.1) that opportunistically reduces the number of S2/S3 states, the final symbol decoding is applied after the hard error correction step.

Permutation coding: The original patent description [22] omits how to handle wearout failures. The permutation coding only achieves cell error rates of $1E-5$ after 37 days. So, we add ECP-6 (stored in SLC mode since it is unclear how to handle wearout failures in the

Table 4: Comparison with tri-level cell PCM [29].

	Data	Wearout failure correction	Drift error correction	Density
4LC in [29]	512 bits / 256 cells	N/A	BCH-32: 320 bits /160cells	1.23 bits /cell
4LC^o in our work	512 bits / 256 cells	ECP-6: 31 cells	BCH-10: 100 bits /50 cells	1.52 bits /cell
3LC in [29]	8 bits / 6 cells	N/A	N/A	1.33 bits /cell
3LC^o in our work	512 bits / 342 cells	mark-and-spare: 12 cells	BCH-1: 10 bits /10 cells	1.41 bit s/cell

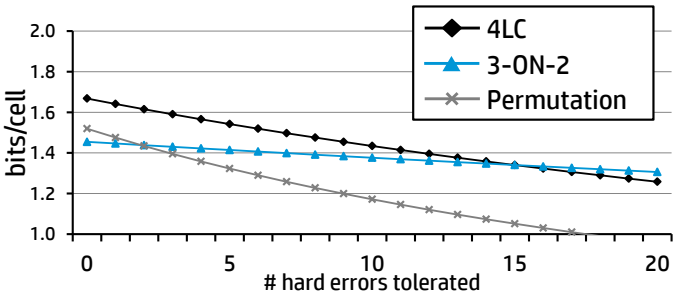


Figure 15: Capacity vs hard error.

context of permutation coding) and an additional 1-bit correcting BCH code.

Comparison: Table 3 compares 3-ON-2, 4LC, and permutation. The capacity of the proposed 3-ON-2 (considering both wearout failure and drift error correction) is only 7.4% lower compared to the 4LC design. This difference is much lower than the initial bit density difference would imply (2 bits/cell for 4LC vs. 1.58 bits/cell for 3LC).

Considering only data storage, permutation coding has higher capacity than the 3-ON-2 ($11/7$ vs. $3/2$). Including hard and transient error correction overhead, however, the 3-ON-2 outperforms permutation coding.

So far, we assumed six wearout failures per 64B blocks. This requirement was assumed in the context of SLC-PCM [27], but in general, MLC-PCM has lower reliability and may need stronger wearout failure tolerance. Figure 15 shows the storage capacities of the three designs, as we increase hard error correction capability. 3-ON-2's mark-and-spare has low storage overhead per wearout failure; hence, its capacity overhead grows much slower than those of the other schemes as we increase hard error correction capability.

We also compare ECC encoding and decoding latencies in FO4 delay using a logic model described in [32]. Both BCH-1 and BCH-10 have comparable encoding latency, where the number of message bits is the dominant factor. In terms of timing-critical decoding speed (including error correction), however, BCH-1 is more than 8x faster than BCH-10.

6.7 Comparison with Tri-Level Cell PCM

As discussed earlier, Seong *et al.*'s recent work, tri-level cell PCM [29], is very close to our work. Both studies make the same observation that most drift errors occur in the second largest resistance level in 4LC and propose to use 3LC to avoid the error-prone state (S3 in our notation). Table 4 compares 4LC and 3LC designs in our work and tri-level cell PCM.

4LC Designs: The authors of the tri-level cell PCM paper claimed that resistance drift makes 4LC-PCM impractical. In order to tolerate high drift error rates in the 4LC design, they use an unreasonably lengthy ECC (320-bit BCH-32), resulting low cell density (1.23 bits/cell). While we agree that 4LC-PCM is not usable for long-terms storage, we show that this broad conclusion is overly pessimistic: The optimized 4LC design (4LC^o) with a strong, but realizable ECC (BCH-10) at a reasonable refresh period (17 minutes) can achieve our reliability goal (MTBF 10 years). Although refresh can still impact power and performance (evaluated in the next section), we show that 4LC is usable for volatile memory and provides the highest density (1.52 bits/cell).

3LC Designs: The 3LC designs in both studies are conceptually the same, but different in cell state and boundary mapping: the design in the tri-level cell paper with the straightforward mapping is equivalent to 3LC^n in our notation; and our design uses the optimal cell mapping (3LC^o), yielding even lower drift error rates.

Tri-level cell PCM also uses 3-ON-2 (<3,2> in their notation). For 8-bit access granularity, they store eight bits in six cells: two sets of 3-ON-2 (four cells) and two one-bit cells. A PCM device, however, internally uses 512-bit (or larger) buffers similar to row-buffers in DRAM. So we store the whole 512-bit data in 342 cells, maximizing 3LC capacity. Note that BCH-1 in our design is a safety net; drift error rates in 3LC^o are low enough to retain data reliably for more than 10 years without ECC. One of our unique contributions is the mark-and-spare wearout tolerance mechanism enabled by non-power-of-two levels per cell. The authors of the tri-level cell PCM paper did not consider wearout failures in their designs, but they focused on relaxed writes to S2 in order to improve write latency and bandwidth (Bandwidth-Enhanced 3LC).

7 Impacts of Refresh

We have shown that the proposed 3LC-PCM has long data retention time and is genuinely nonvolatile, but the conventional 4LC-PCM, even with many optimizations, fails to meet the nonvolatility requirement (years of retention time without power). Still, we can use the 4LC-PCM with periodic refresh as volatile main memory, providing a slight capacity advantage (but only around 7%).

Refresh, however, may impact performance and energy. PCM has relatively low write bandwidth; normal writes and refresh need to share this finite write bandwidth. Reads can stall while refresh is going on the target PCM bank.

We use a cycle-based simulator [1] to evaluate performance, energy, and power impacts of the 3LC and 4LC designs. We use applications from SPEC CPU 2006 [31] and STREAM microbenchmark [21]. We include memory intensive applications (**STREAM**, **mcf**, **libquantum**, **bzip2**, and **lbm**), penalized by the performance and energy overheads of refresh, as well as compute intensive one (**namd**), where the effect of refresh is not critical. Table 5 describes key simulation parameters used in the evaluation. Since a detailed MLC-PCM device model is not available as of today, we adapt the simulator with a simple latency-based PCM model. We assume a 3D-stacked device with a bottom logic die that implements BCH encoding/decoding, wearout tolerance, and symbol encoding/decoding. Write throughput is set to 40MB/s, translating to a maximum of four writes (including refresh) for four-write-window of 6.4 us (similar to four-activation-window in DDRx DRAM).

We compare four designs: 4LC-REF, 4LC-REF-OPT, 4LC-NO-REF, and 3LC. The two 4LC^o implementations (4LC-REF and 4LC-REF-OPT) have periodic per-bank refresh with the 17-minute re-

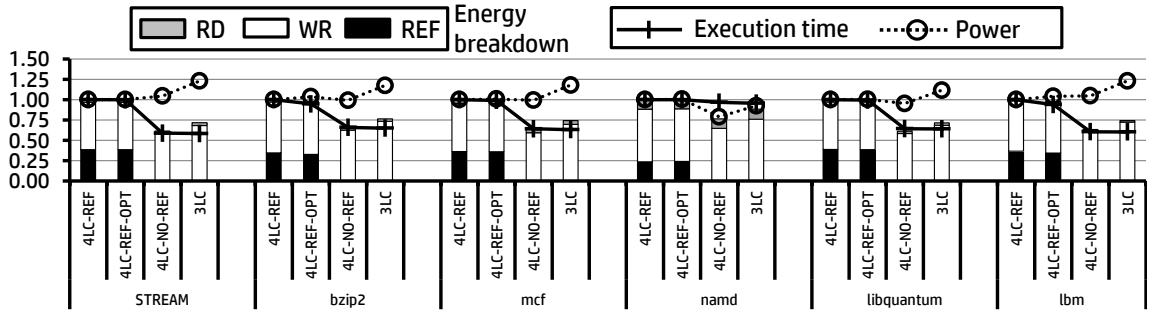


Figure 16: Normalized execution time, energy, and power (the lower, the better).

Table 5: Simulation parameters.

Processor	an out-of-order core running at 3.2GHz
L1 cache	16kB instruction and data caches 64B line size
L2 cache	512kB unified cache, 64B line size
MLC-PCM	16GB, 8 banks, 64B blocks read: 200 ns write: 1us write throughput: 40MB/s

fresh interval and adds 36.25 ns BCH-10 overhead in addition to the 200 ns read latency. A refresh operation makes the corresponding bank unavailable for 1 us, but it does not in 4LC-REF-OPT. Instead, 4LC-REF-OPT emulates an ideal intelligent refresh scheme that can avoid all refresh-related contention. To further remove the refresh overhead on limited write bandwidth in both 4LC-REF and 4LC-REF-OPT, we model an impractical no-refresh scheme 4LC-NO-REF. The 3LC is 3LC^o with 3-ON-2 encoding and mark-and-spare. We consider all practical overheads for the 3LC model: it does not use refresh, but adds 5 ns to the PCM read latency.

Figure 16 compares the performance, energy, and power of 4LC and 3LC designs, normalized to those of 4LC-REF. Both 4LC-NO-REF and 3LC show much lower energy and execution time than those of 4LC-REF and 4LC-REF-OPT, because refresh operations use a considerable fraction of limited write throughput, limiting performance and consuming energy. The only exception is benchmark **namd**, which is compute-intensive and insensitive to memory latency and bandwidth. 3LC's performance improvements also imply higher activity factors hence higher power, but the increase in power is related to the memory sub-system and much lower compared to the speedup. The significant performance and energy benefits demonstrate that the 3LC design is not only adequate for nonvolatile storage but also efficient for volatile main memory usage.

8 Discussion

Resistance drift poses a significant challenge for the practical adoption of MLC-PCM. Drift-induced error rates are too high and demand frequent refresh to reliably retain cell values. The need for refresh makes two-bit-per-cell MLC-PCM volatile, losing one of the most unique PCM advantages. We performed a model-driven analysis on resistance drift and proposed a new three-level-cell (3LC) PCM that lowers drift error rates by several orders of magnitude. Our technique restores nonvolatility of MLC-PCM, making it practically usable in both main memory and nonvolatile storage.

We developed an encoding technique (3-ON-2) that stores binary information in ternary memory cells. We also devised a low-storage-overhead wearout tolerance technique (mark-and-spare) for 3-ON-2. Considering both transient and wearout correction information, the proposed 3LC design has only 7.4% less capacity than the conventional four-level-cell PCM, yet achieving many practical advantages: nonvolatility, low-latency reads, *etc*.

Although we used specific PCM parameters in this study, our techniques, including three-level cells and mark-and-spare wearout tolerance, are applicable to any PCM that suffers from resistance drift. Future advances in materials and devices may mitigate the resistance drift problem; 4LC-PCM can retain data reliably for hours to days. Yet it requires refresh – it cannot be used as nonvolatile storage. Our simple solution makes MLC-PCM truly nonvolatile.

We have shown that widening the safety margins is the key to drift error control. Absent a change in the material that widens the dynamic range, we can best improve storage density by reducing the variability of the log-resistance of written cells. We can generalize the proposed technique to support other non-power-of-two-level cells for the higher density MLC-PCM—e.g., five-level or six-level cells. We can combine the described optimal state mapping, information encoding, and error correction techniques with the generalized non-power-of-two-level cells to practically enable high density MLC-PCM with further density improvements.

9 Acknowledgment

This research was partially supported by the Department of Energy under Award Number DE - SC0005026.
See `http://www.hpl.hp.com/DoE-Disclaimer.html` for additional information.

References

[1] McSim. `http://cal.snu.ac.kr/mediawiki/index.php/McSim`.

[2] M. Awasthi, M. Shevgoor, K. Sudan, B. Rajendran, R. Balasubramonian, and V. Srinivasan. Efficient scrub mechanisms for error-prone emerging memories. In *Proc. the 18th Int'l Symp. High-Performance Computer Architecture (HPCA)*, Feb. 2012.

[3] R. Azevedo, J. D. Davis, K. Strauss, P. Gopalan, M. Manasse, and S. Yekhanin. Zombie memory: Extending memory lifetime by reviving dead blocks. In *Proc. the 40th Int'l Symp. Computer Architecture (ISCA)*, Jun. 2013.

[4] M. Boniardi et al. Statistical and scaling behavior of structural relaxation effects in phase-change memory (PCM) devices. In *Proc. the IEEE Int'l Reliability Physics Symp. (IRPS)*, Apr. 2009.

[5] R. C. Bose and D. K. Ray-Chaudhuri. On a class of error correcting binary group codes. *Information and Control*, 3:68–79, 1960.

[6] 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. *J. Vacuum Science and Tech. B.*, 28(2):223–262, 2010.

[7] Y. Choi et al. A 20nm 1.8V 8Gb PRAM with 40MB/s program bandwidth. In *Proc. the IEEE Int'l Solid-State Circuits Conf. (ISSCC)*, Feb. 2012.

[8] J. Coburn, A. M. Caulfield, A. Akel, L. M. Grupp, R. K. Gupta, and S. Swanson. NV-Heaps: Making persistent objects fast and s afe with next-generation, non-volatile memories. In *Proc. the 16th Int'l Conf. Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, Mar. 2011.

[9] J. Condit, E. B. Nightingale, C. Frost, E. Ipek, B. Lee, D. Burger, and D. Coetzee. Better I/O through byte-addressable, persistent memory. In *Proc. the ACM 22nd Symp. Operating Sys. Principles (SOSP)*, Oct. 2009.

[10] T. M. Cover. Enumerative source encoding. *IEEE Transactions on Information Theory*, 19:73 – 77, Jan. 1973.

[11] X. Dong, N. Muralimanohar, N. P. Jouppi, R. Kaufmann, and Y. Xie. Leveraging 3D PCRAM technologies to reduce checkpoint overhead for future exascale systems. In *Proc. the Int'l Conf. High Performance Computing, Networking, Storage, and Analysis (SC)*, Nov. 2009.

[12] L. Goux et al. Degradation of the reset switching during endurance testing of a phase change line cell. *IEEE Trans. Electron Devices*, 56(2):354–358, 2009.

[13] R. W. Hamming. Error correcting and error detecting codes. *Bell System Technical J.*, 29:147–160, Apr. 1950.

[14] A. Hocquenghem. Codes correcteurs d'erreurs. *Chiffres (Paris)*, 2:147–156, 1959.

[15] M. Y. Hsiao. A class of optimal minimum odd-weight-column SEC-DED codes. *IBM J. Res. and Dev.*, 14:395–301, 1970.

[16] 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 *Proc. the Symp. VLSI Technology (VLSIT)*, Jun. 2010.

[17] D. Ielmini, D. Sarma, S. Lavizzari, and A. L. Lacaita. Reliability impact of chacogenide-structure relaxation in phase-change memory (PCM) cells - part I: Experimental study. *IEEE Trans. Electron Devices*, 56(5):1070–1077, May 2009.

[18] L. Jiang, Y. Zhang, and J. Yang. ER: Elastic RESET for low power and long endurance MLC based phase change memory. In *Proc. the ACM/IEEE Int'l Symp. Low Power Electronics and Design*, Aug. 2012.

[19] S. Kang et al. A 0.1-um 1.8-V 256Mb phase-change random access memory (PRAM) with 66-MHz synchronous burst-read operation. *JSSC*, 42(1):210–218, 2007.

[20] P. Kogge and H. Stone. A parallel algorithm for the efficient solution of a general class of recurrence equations. *IEEE Trans. Computers*, C-22(8):786–793, Aug. 1973.

[21] J. D. McCalpin. STREAM: Sustainable memory bandwidth in high performance computers. `http://www.cs.virginia.edu/stream/`.

[22] T. Mittelholzer, N. Papandreou, and C. Pozidis. Data encoding in solid-state storage devices. U.S. Patent, Application 2011/0296274.

[23] T. Nirschl et al. Write strategies for 2 and 4-bit multi-level phase-change memory. In *IEDM Tech. Digest*, Dec. 2007.

[24] M. K. Qureshi. Pay-As-You-Go: Low overhead hard-error correction for phase change memories. In *Proc. the 44th IEEE/ACM Int'l Symp. Microarchitecture (MICRO)*, Dec. 2011.

[25] M. K. Qureshi, M. Franceschini, and L. A. Lastras-Montano. Improving read performance of phase change memories via write cancellation and write pausing. In *Proc. the Int'l Symp. High-Performance Computer Architecture (HPCA)*, January 2010.

[26] M. K. Qureshi, J. Karidis, M. Franceschini, V. Srinivasan, L. Lastras, and B. Abail. Enhancing lifetime and security of PCM-based main memory with start-gap wear leveling. In *Proc. the Int'l Symp. Microarchitecture (MICRO)*, Dec. 2009.

[27] S. Schechter, G. H. Loh, K. Strauss, and D. Burger. Use ECP, not ECC, for hard failures in resistive memories. In *Proc. the Int'l Symp. Computer Architecture (ISCA)*, June 2010.

[28] N. H. Seong, D. H. Woo, V. Srinivasan, and J. A. R. H.-H. S. Lee. SAFER: Stuck-at-fault error recovery for memories. In *Proc. the Int'l Symp. Microarchitecture (MICRO)*, Dec. 2010.

[29] N. H. Seong, S. Yeo, and H.-H. S. Lee. Tri-level-cell phase change memory: Toward an efficient and reliable memory system. In *Proc. the 40th Int'l Symp. Computer Architecture (ISCA)*, Jun. 2013.

[30] J. Sklansky. Conditional-sum addition logic. *IRE Trans. Electronic Computer*, EC-9:226–231, Jun. 1960.

[31] Standard Performance Evaluation Corporation. SPEC CPU 2006. `http://www.spec.org/cpu2006/`, 2006.

[32] D. Strukov. The area and latency tradeoffs of binary bit-parallel BCH decoders for prospective nanoelectronic memories. In *Proc. Asilomar Conf. Signals Systems and Computers*, Oct. 2006.

[33] S. Venkataraman, N. Tolia, P. Ranganathan, and R. Campbell. Consistent and durable data structures for non-volatile byte-addressable memory. In *Proc. the 9th USENIX Conf. File and Storage Technologies (FAST)*, Feb. 2011.

[34] H. Volos, A. J. Tack, and M. Swift. Mnemosyne: Lightweight persistent memory. In *Proc. the 16th Int'l Conf. Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, Mar. 2011.

[35] J. Wang, X. Dong, G. Sun, D. Niu, and Y. Xie. Energy-efficient multi-level cell phase-change memory system with data encoding. In *Proc. the 29th Int'l Conf. Computer Design (ICCD)*, Oct. 2011.

[36] C. Wilkerson, H. Gao, A. R. Alameldeen, Z. Chishti, M. Khellah, and S.-L. Lu. Trading off cache capacity for reliability to enable low voltage operation. In *Proc. the 35th Ann. Int'l Symp. Computer Architecture (ISCA)*, Jun. 2008.

[37] W. Xu and T. Zhang. Using time-aware memory sensing to address resistance drift issue in multi-level phase change memory. In *Proc. the Int'l Symp. Quality Electronic Design (ISQED)*, Mar. 2010.

[38] S. Yeo, N. H. Seong, and H.-H. S. Lee. Can multi-level cell pcm be reliable and usable? analyzing the impact of resistance drift. In *the 10th Ann. Workshop on Duplicating, Deconstructing and Debunking (WDDD)*, Jun. 2012.

[39] 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 *Proc. the Int'l Symp. High-Performance Computer Architecture (HPCA)*, Feb. 2011.

[40] W. Zhang and T. Li. Helmet: A resistance drift resilient architecture for multi-level cell phase change memory system. In *Proc. the IEEE 41st Int'l Conf. Dependable Sys. & Networks (DSN)*, Jun. 2011.