Use ECP, not ECC, for Hard Failures in Resistive Memories

Stuart Schechter, Gabriel H. Loh†, Karin Strauss, Doug Burger
Microsoft Research, Redmond, WA
†Georgia Institute of Technology, Atlanta, GA
{stus,kstrauss,dburger}@microsoft.com, loh@cc.gatech.edu

ABSTRACT

As leakage and other charge storage limitations begin to impair the scalability of DRAM, non-volatile resistive memories are being developed as a potential replacement. Unfortunately, current error-correction techniques are poorly suited to this emerging class of memory technologies. Unlike DRAM, PCM and other resistive memories have wear lifetimes, measured in writes, that are sufficiently short to make cell failures common during a system's lifetime. However, resistive memories are much less susceptible to transient faults than DRAM. The Hamming-based ECC codes used in DRAM are designed to handle transient faults with no effective lifetime limits, but ECC codes applied to resistive memories would wear out faster than the cells they are designed to repair. This paper proposes *Error-Correcting Pointers* (ECP), a new approach to error correction optimized for memories in which errors are the result of permanent cell failures that occur, and are immediately detectable, at write time. ECP corrects errors by permanently encoding the locations of failed cells into a table and assigning cells to replace them. ECP provides longer lifetimes than previously proposed solutions with equivalent overhead. Furthermore, as the variance in cell lifetimes increases – a likely consequence of further process scaling – ECP's margin of improvement over existing schemes also increases.

Categories and Subject Descriptors

B.3.4 [**Hardware**]: Memory Structures—*Reliability, Testing, Fault-Tolerance, Error-Checking*

General Terms

Reliability

Keywords

Memory, Error Correction, Hard Failures, Resistive Memories, Phase-Change Memory

ISCA'10, June 19–23, 2010, Saint-Malo, France.

1. INTRODUCTION

The scaling of DRAM technology to smaller feature sizes is in jeopardy, as physical limitations – particularly limited charge – may prevent DRAM scaling beyond 30nm [1, 12]. The small number of electrons that can be stored on these shrinking capacitors, particularly in the presence of sub-threshold leakage, may limit further scaling. Resistive memories, which arrange atoms within a cell and then measure the resistive drop through the atomic arrangement, are promising as a potentially more scalable replacement for DRAM and Flash. These technologies include spin-torque-transfer magnetoresistive memory (STT-MRAM), ferroelectric memory (FRAM), memristors, and phase-change memories (PCM). Of these emerging technologies, PCM has received the most research attention in the architecture literature [7, 11, 21, 22, 26, 27, 28], as it is closest to commercialization [18, 24].

Instead of representing information as the presence or absence of electrical charge, PCM encodes bits in different physical states of a chalcogenide material [2, 3, 4, 10, 14, 18, 20, 23]. Through the application of different programming currents, the phase of the material can be melted and then re-solidified into either a crystalline or amorphous state, each with a distinct electrical resistance.

Since the state of the material is effectively a static configuration of atoms, the material, once programmed, retains its state for long periods of time. This characteristic obviates the need for leakage control and refresh operations. PCM technology is predicted to scale well to smaller feature sizes, with 9nm devices having been demonstrated. While PCM is slower than DRAM to read (two to three times) and considerably slower and more power intensive to write [13], the write latency and power shrinks as PCM cells scale down, since the total volume of phase-change material per cell also decreases. Memory architectures have been proposed to address PCM's latency and power issues to make PCM competitive with DRAM [13, 22, 28].

The major limitation of PCM as a DRAM replacement is its limited write endurance. Next-generation PCM designs can only endure 10^7 to 10^9 writes before the cell permanently fails [1, 8]. In contrast, DRAM cells can be written up to 10^{15} times before failure, which is effectively unlimited. Additionally, the failure mode of PCM cells more closely resembles Flash than DRAM. The heating and cooling process required to write a cell, and the expansion and contraction that results, eventually cause the heating element to detach from the chalcogenide. Detachment of the heating element results in a "stuck-at" hard fault that can be subsequently

read but not rewritten. Unlike charge-based DRAM, the material state of PCM cells is not susceptible to particle-induced soft errors [15, 17, 29]. While resistive drift, resulting from gradual atomic motion at high temperatures, can eventually lead to soft errors, PCM cells are expected to hold their state for years at typical operating temperatures.

Making PCM technology a viable DRAM replacement will require mitigating wear-related failures through architectural enhancements such as write buffers [13, 22], compression schemes [26], wear-leveling mechanisms [21, 22, 26, 27, 28], error-correcting codes [27], or operating system page remapping [11, 27, 28]. These techniques will reduce the total write traffic, spread the writes more uniformly over the memory cells, and cope with failures after they occur. An additional, highly effective technique that can both alleviate wear and quickly detect failures is a "read-write-read" pattern for write operations. Initially, a read is performed from the row buffers or the PCM array to access the prior value of the memory, which is bitwise compared to the write data. The write is then performed, with only the changed bits being written to the array. Subsequently, a final read checks to ensure that the data were correctly written. If the checking read returns an incorrect result, the write operation may be reissued, or a correction action must be taken.

The standard ECC implementations used in DRAM are less than ideal given these three aspects of resistive memories: the strong need to reduce writes, the dominance of hard failures, and the ability to identify failures at the time of the write. Hamming-based ECC codes modify the error-correction bits whenever any data in the protected block change, resulting in high entropy that increases wear. These codes face an unappealing choice when selecting the region size; protecting a larger region induces more wear, as the codes must be rewritten when any data in that region are changed. Protecting finer-grain regions reduces the ECC bits' wear but makes each page fail when enough cells fail within any of the small protected regions. This early failure problem can be exacerbated by cell lifetime variation. Many of the previously proposed techniques assume that all of the PCM memory cells have the same write endurance, with each cell failing after exactly W writes. In real systems, however, parametric variations in the manufacturing process create a non-uniform distribution of cell characteristics [5, 6, 19, 26]. As resistive memories are scaled to smaller dimensions, lifetime variability may become more pronounced, making it crucial for error-correcting schemes to handle numerous early cell failures gracefully.

This paper proposes *Error-Correcting Pointers* (ECP) that work to minimize write wear, handle permanent faults rather than soft errors, and improve overall memory lifetime in the presence of high parametric variation and corresponding early cell failures. Whereas error-correcting codes associate a number of coded bits with each block of data bits, ECP encodes and stores the addresses of failed cells and allocates additional cells to replace them. This ECP encoding scheme can also correct failures in the correction cells themselves without additional error-correcting schemes or cells.

ECP is also complementary to previously proposed approaches that reduce the number of writes to the PCM (*e.g.*, write combining), enabling those approaches to be composed with ECP for additional longevity. As resistive memories scale to smaller geometries, and the parametric variations within them likely increase, ECP will better tolerate clustered errors without exacerbating wear-out, and may be necessary to permit further device scaling.

As designers increase ECP entries, at some point the extended lifetime obtained from additional ECP entries reaches diminishing returns, since the rate of cell failures grows over time. This paper evaluates two additional approaches to extend lifetime further. First, a layered ECP approach provides small pointers for each row and a row of larger pointers per page, balancing both the number of pointers using a given overhead and the reach of each pointer. Layered ECP is able to reduce lifetime over the best-performing ECP organization with equivalent overhead. Second, we observe that both ECP and layered ECP eventually reach a point where adding more pointers provides less lifetime benefit than simply adding more pages. We show how to obtain that point, and, thus for a given correction scheme, demonstrate how to compute the minimum storage needed to allow a memory size M to be available for W writes, assuming a lifetime coefficient of variance V. This result will permit memory vendors to reason about the overhead and cost of supporting higher variances for target capacities and lifetimes as resistive memories are scaled to near-atomic dimensions.

2. BACKGROUND

2.1 Phase-Change Memory Failure Model

While the longer access latencies and write power pose some challenges, the limited write endurance of PCM may prove to be the greatest obstacle to widespread adoption of PCM as a DRAM replacement. Writing PCM requires elevating the temperature of the phase-change material to 650 degrees Celsius. After enough write cycles (on the order of 10^8), the mechanical stresses of repeated thermal expansion and contraction cause the heating element to separate from the phase-change material, rendering the cell unmodifiable. Without any additional protection or enhancements, this limited write endurance can render a PCM memory useless in less than one month of operation [13]. The useful lifetime of PCM must be extended to many years for it to be a practical main memory technology. Because of good thermal insulation between cells, we expect that cell failures will be independent and identically distributed, and that cell lifetimes will follow a normal distribution in keeping with other sorts of parametric variation. Finally, while PCM cells' values can decay (due to atomic motion), resulting in soft errors, the refresh period at typical operating temperatures is measured in years. System designers must ensure that the ambient PCM temperatures do not exceed a sustained level that will result in soft errors not caught by an occasional (daily?) refresh operation.

2.2 Error Correction

Error-correcting codes have been studied and applied in a variety of contexts for decades. Single-error-correcting (SEC) Hamming codes [9], which include commonly used dual-error-detecting SECDED varieties, are best known for providing error protection for DRAM as well as on-chip structures such as caches [16]. Error codes are specified as "(n,k)" where k bits of actual data are encoded into $n > k$ bits of redundant/protected data. For example, Hamming introduced a (7,4) code that can correct a single error and detect (but not correct) up to two errors by appending the k=4 original data bits with three additional error-correcting

bits. The coding is generalizable to larger blocks of data, such as the (72,64) code used to provide SECDED protection on 64-bit DRAM data.

To provide multiple-bit error correction in PCMs, more complex coding schemes may be considered. Polynomial-based codes, such as Reed-Solomon, and Bose, Ray-Chaudhuri, Hocquenghem (BCH) have already been employed in Flash storage devices and optical media to deal with multiple bit failures within a block. BCH has also been proposed for correcting for PCM bit failures [27].

From the Hamming Bound we can derive a theoretical lower limit on the space overhead S required to enable up to n errors to be corrected while encoding d data bits.

$$S_{min} \geq \frac{\left\lceil \log_2 \sum_{e=0}^{n} \binom{d\cdot(1+S_{min})}{e} \right\rceil}{d}$$

3. ERROR-CORRECTING POINTERS

Traditional Error-Correcting Codes (ECC) store sufficient information to derive the source of errors in locations undetermined at encoding time, allowing them to correct soft errors discovered when a block of data is read. Conversely, Error-Correcting Pointers (ECP) directly store the address of memory cells determined to have permanently failed during the verification of a memory write. ECP operates within each memory chip at the row level.

The ECP_n scheme uses n *correction pointers* to specify the addresses of failed cells, and pairs each pointer with a replacement memory cell. Together, the pointer and replacement cell form a *correction entry*.

Figure 1a illustrates the simplest ECP implementation, ECP_1, where a single correction entry corrects up to one bit. The example uses a row with 512 data bits. When no errors are present in the data, the correction pointer is empty, the full bit is set to 0 (false). This indicates the entry is *inactive* as there are no errors to correct. When a bit fails, for example bit 2 in Figure 1a, the correction entry is marked full (or *active*), the correction pointer is set to point to bit 2, and the replacement cell now stores the value that belongs in bit 2. Henceforth, when the row is written, the value to be written to the failed cell identified by the correction pointer is instead written to the replacement cell. When the row is read, the value in the replacement cell supersedes the value read from the defective cell.

Generalizing ECP_1 to n entries (ECP_n) is illustrated in Figure 1b, using ECP_5 as an example. The full bit is now set only when *all* error-correction entries are in use. Otherwise, the full bit is set to 0 and the bits of the last correction entry ($n-1$=4) contain a unary-encoded counter denoting how many of the other $n-1$ correction entries (entry 0 to $n-2$=3) are active. In the illustration, the full bit is set to false (0) and the two lowest order bits in entry 4 are set, indicating that correction entries 0 and 1 are active. As before, the first bit to fail (bit 2) is replaced by correction entry 0. The availability of a second correction entry (entry 1) enables us to correct a second failure (bit 509).

Errors in replacement cells are less likely to occur than errors in the original data cells, as they do not begin to wear until they are put into use to replace a failed cell. ECP can still correct these errors. When two correction entries point to the same cell, the correction entry at the higher index takes precedence over the one with the lower index, just as the correction entry at the lower-index takes precedence over the failed bit in the data array. For example in Figure 1c, the replacement cell in correction entry 0 has failed. To compensate for this, we activate correction entry 1 and have it point to the same failed cell. The replacement cell in entry 1 supplants both the original failed cell and the failed replacement cell in entry 0.

Precedence rules also make possible the correction of errors in correction pointers. Such errors are even less likely than errors in replacement cells, as these are written to at most once (twice for the cells in entry $n-1$ that are also used to activate other cells). Almost all errors in the pointers themselves will be cells that failed upon manufacture. An error in a correction pointer, as illustrated in Figure 1d, effectively replaces a working cell with a working replacement cell, doing no harm but also failing to repair the failure for which it was allocated (bit 2). We thus allocate an additional correction entry to correct the original failure of bit 2. Overall, two bits have failed and two error-correction entries have been consumed.

ECP can correct errors both in data cells and in its own data structures, while allocating only enough bits per correction pointer to address data bits. ECP_n can correct *any* n cell failures, regardless of whether they are in data cells or correction entries.[1] Because the scheme works at the cell level, it is equally effective for use in multi-level cell (MLC) memories that store more than one bit per cell.

ECP requires 1 full bit, n replacement bits, and n pointers large enough to address the original data bits. Thus the fractional space overhead $S(\text{ECP}_n)$ for a row with $d = 512$ data bits is:

$$S(\text{ECP}_n) = \frac{1 + n + n \cdot \lceil \log_2 d \rceil}{d} = \frac{1 + n \cdot (1 + \lceil \log_2 512 \rceil)}{512}$$

$$S(\text{ECP}_6) = \frac{1 + 6 \cdot 10}{512} = \frac{61}{512} = 11.9\%$$

4. EXPERIMENTS

Memory failure simulation presents special challenges, as it is impractical to perfectly simulate the real operation of a memory over a full lifetime or to simulate all possible wear patterns. We make a number of simplifying assumptions in our simulation. First, we assume that existing wear-leveling techniques (*e.g.*, stop-gap [21], fine-grained wear leveling [22]) already spread writes evenly over the memory. Second, we assume that memory chips store data in 512-bit rows, and that each contiguous block of memory is spread over eight chips. Third, we assume that writes modify a single region of bits randomly located within the page. When evaluating competing schemes that divide memory rows into smaller blocks, we assume that writes that are narrower than a block touch only one block – maximizing the endurance of the competing schemes. Each bit within the region modified by the write is assumed to change value with probability 0.5.

For each scheme, our simulator lays out a full page of memory and allocates a cell for each bit, including both data bits and meta-data structures such as correction en-

[1]The one extraordinarily rare exception to this rule occurs when a failure activates a high-precedence correction entry, its replacement cell fails, and no higher-precedence correction entry is available to repair it. For example, such a failure could be caused if the full bit is manufactured stuck at 1 (activating all correction entries) and the replacement bit in the highest-precedence correction entry fails.

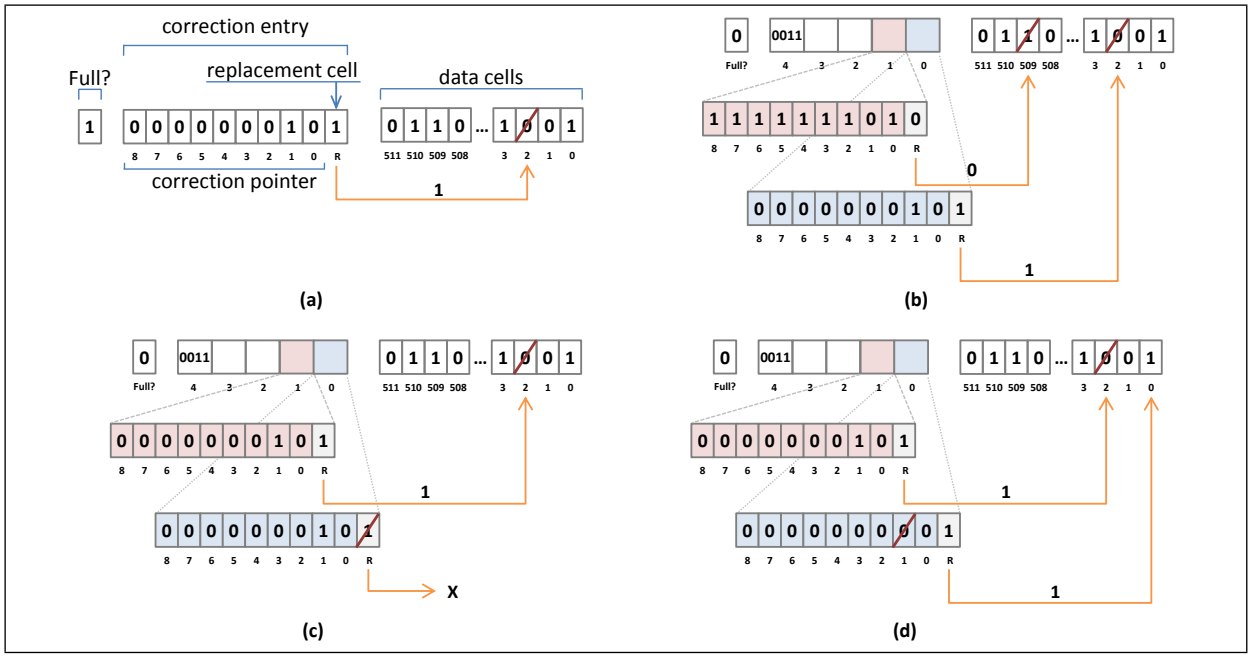


Figure 1: Correction entries enable permanent replacement of failed memory cells. (a) A simple ECP_1 scheme that corrects up to a single bit error. The correction pointer specifies the location of a dead cell to be supplanted by the replacement cell. (b) The ECP_5 scheme that corrects up to five failed cells. (c) A failure in the replacement cell can be handled by allocating an overriding correction entry at the same address, and similarly (d) a rare cell fault within a correction pointer can harmlessly cause a still-operational cell to be replaced by another working cell, requiring an additional correction entry to be allocated to replace the data cell that the faulty correction entry was intended to replace.

tries. Wear-rates are then assigned to each cell based on the calculated expected bit changes per page write. For example, for a write modification width of 2 bytes to the 4096 byte page, the expected wear on each data bit would be $0.5 \cdot \frac{2}{4096}$. The initial wear-rate for an unused replacement bit would initially be set to 0.

Next, for each simulation run, the simulator assigns a random lifetime to each cell using a normal distribution with a mean of 10^8 bit-writes-until-fail and a variance of 0.25 (unless specified otherwise). Each cell's expected remaining lifetime in page-writes-until-fail is calculated by dividing the remaining bit-writes-until-fail by the cell's wear rate. The next bit to fail is identified by finding the bit with the lowest page-writes-until-fail.

When a bit fails, the model determines whether the failure is terminal to the page and, if it isn't, simulates the action taken within the page to correct it. For example, when a cell dies in the ECP_n design, the cell that replaces it begins to encounter wear.

We assume a memory architecture in which 4KB logical pages are mapped to 4KB physical pages. When a physical page encounters an uncorrectable error, the page dies and is mapped out by the OS. Each page death reduces the size of the physical memory, which increases wear on the remaining pages. In other words, when a physical page dies there is one fewer page over which to spread the wear placed on the logical memory. The surviving pages will collectively absorb the wear that had previously been incurred by the newly deceased page. We assume each surviving page absorbs an equal amount of this increased wear. We simulate the impact of dying pages by, upon each page death, decreasing each of the survivors' remaining lifetimes by the fraction of the additional wear that each will now incur.

Page size	4KB	(32768 bits)
Row size	32B	(512 bits)
Rank	1	
Chips per rank	8	
Bit lines per chip	x8	
Mean cell lifetime	10^8	
Lifetime variance	0.25	

Table 1: Default architectural assumptions.

We use the architectural parameters shown in Table 1. For each configuration we simulate at least 2,000 physical page lifetimes. We present results using the metric of mean-writes-per-page.

5. COMPARISON TO EXISTING SCHEMES

Today's DRAM memories use single-error-correction (SEC) Hamming codes. Eight memory chips with eight data lines, providing a total of 64 data bits per bus cycle, are paired with a ninth chip. SEC requires 7 additional bits to correct 64 data bits, and so the eighth bit on the spare chip allows for detection, but not correction, of a second error (SECDED). As we assume all errors are detectable following a write, the additional detection is of no value.

We assume that our schemes should have at most 12.5% space overhead so that they, like SEC, fit in a ninth memory chip (for schemes that correct blocks spanning chips) or in the equivalent overhead (for ECP and other row-based schemes that operate on rows within a chip).

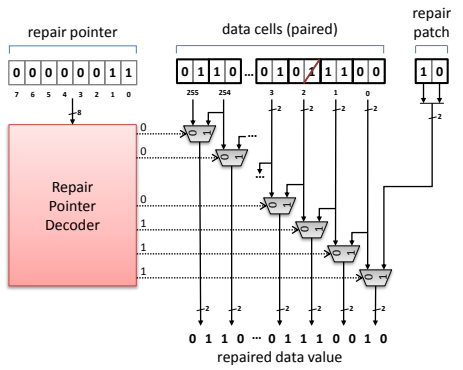


Figure 2: The "bit-fix" scheme proposed by Wilkerson *et al.*

5.1 Schemes Compared

SEC$_{64}$

SEC$_{64}$ simulates the correction scheme in today's DRAM. It chunks memory into 64-bit blocks, each divided evenly over eight chips, and corrects for up to one error in each block. A second error within a block is terminal. For all schemes, we assume that memory is deallocated at the page level, and so a second error within a block will cause the entire page to be deallocated. We simulate SEC$_{64}$ by assuming that each SEC bit within a block will change with probability 0.5 any time one or more data bits are written to the block. The space overhead of SEC$_{64}$ is 7 bits per 64-bit block, or 10.9%.

Wilkerson$_4$

The most closely related error-correcting scheme to ECP was introduced by Wilkerson *et al.* for use in caches in which certain cells may fail to operate correctly at desired voltages [25]. Wilkerson's "bit-fix" scheme, illustrated in Figure 2, pairs up cells and provides an extra replacement pair, called a repair patch. Whereas ECP directly substitutes failed cells with replacement cells, Wilkerson's scheme locates replacement cells on the logical edge of the data line, shifting as much of the row as necessary to fill vacancies left by the failed cells. This seemingly small architectural difference makes it difficult for Wilkerson's scheme to create entries that correct failures in other correction entries – the replacement cells are not themselves addressable. Instead, each 10-bit correction entry (an 8-bit pointer to select one of the 256 bit pairs, and the 2-bit replacement pair) in the bit-fix scheme requires five dedicated SECDED bits (not shown in the figure). In addition to this 50% space overhead, these SECDED bits are a potential source of wear failure, as their values change whenever the replacement bit in the correction entry changes.

To be an appropriate comparison with ECP, we extended Wilkerson's scheme to target a PCM implementation of main memory, using state stored within each memory row. Specifically, we optimized Wilkerson's scheme for PCM by using 4-bit SEC within correction entries instead of 5-bit SECDED; write-time error detection obviates the need for the double-error detection. Although the SEC code covers the entire 10-bit repair entry, it is only the repair bits that will experience frequent modifications, and so we strategically place these two bits within the 10-bit entry so that the minimum number of SEC bits toggle on updates to the repair patch.

In our implementation, Wilkerson's scheme requires 1 full bit, and contains n entries each with 2 replacement bits, 8 address bits, and 4 SEC bits. The 40% overhead that SEC bits add to each entry limits Wilkerson to four error-correction entries (Wilkerson$_4$) within our 12.5% overhead constraint.

$$S(\text{Wilkerson}_n) = \frac{1 + n \cdot (2 + 8 + 4)}{512}$$

$$S(\text{Wilkerson}_4) = \frac{1 + 4 \cdot 14}{512} = \frac{57}{512} = 11.1\%$$

Pairing$_8$

Ipek *et al.* recently introduced a hardware/software hybrid scheme to tolerate block failures within a page [11]. Their scheme assigns a parity bit to every 8-bit block, though the scheme generalizes to blocks of arbitrary size (Pairing$_n$). A block dies when any bit within it, or its parity bit, fails. Dead blocks are recognizable by their non-matching parity bit. In the rare event that a second bit failure occurs at the same time as the first, any non-dead bit within the block (including the parity bit) may be flipped to ensure a parity failure continues to be detectable. One parity bit per eight data bits results in an overhead of exactly 12.5%.

When the first failed bit within a 4KB page causes a block to die, the page is then *paired* with another page that is selected to ensure that the set of failed block indices do not intersect. In other words, if a block at index i is dead in one page, it must not be dead in the other page. If future failures cause this invariant to be violated, the affected pages must be taken offline until new matches can be identified for them. Ipek *et al.* show that the Pairing$_8$ scheme is viable for up to 160 block failures in each page. We treat the 161st block failure as terminal.

Perfect_Code$_9$

To address the primary limitation of SEC$_{64}$ – the failure of a page should two errors happen to fall within the same 64-bit block – we also evaluated a multi-bit scheme that corrects errors over a larger block. Specifically, we consider correction of multiple errors within a 512-bit block. Rather than test a specific multi-error-correction scheme, we test against the theoretical limit: a perfect n-error-correcting code over the entire block. The number of code bits $S(\text{Perfect\_Code}_n)$ required is dictated by the Hamming Bound:

$$S(\text{Perfect\_Code}_9) = \frac{\lceil \log_2 \sum_{e=0}^{9} \binom{512+64}{e} \rceil}{512} = \frac{64}{512} = 12.5\%$$

To simulate Perfect_Code$_9$ we allocate 64 error-correcting bits per 512-bit row, each of which changes value with probability 0.5 any time one or more data bits are written to the line. We simulate Perfect_Code$_9$ not because it is realistic, but because it provides a theoretical limit on traditional error-correcting schemes. In reality, decoding multi-bit error-correction schemes is expensive and correcting errors at the granularity of an entire block prevents any critical-word-first optimization, as reads cannot complete until the full line is read and errors decoded. The alternative to off-memory-chip correction is to apply the multi-bit error-correcting codes at the row level on the memory chip, but doing so results in disproportionately high, lifetime-limiting wear to the error-correcting bits.

5.2 Results

Table 2 summarizes the error-correction schemes evaluated in this paper.

	overhead	failure unit	failures survivable per unit
SEC_{64}	10.9%	64b block	1
$Pairing_8$	12.5%	4KB page	160
$Wilkerson_4$	11.1%	512b row	4
$Perfect\_Code_9$	12.5%	512b block	9
ECP_6	11.9%	512b row	6

Table 2: Overheads for error-correction schemes in this paper. Blocks span memory chips while rows are contiguous bits within a single chip.

Figure 3 shows the fraction of pages that survive a given number of page writes with a coefficient of variation of 0.25 in the mean cell lifetime and modified region widths of 128, 256 and 512 bits (equal to one physical row). Figure 4 fixes the modified region width to 512 bits and presents these page-survival fractions for coefficients of variance of 0.2, 0.25, and 0.3 (Figure 3c and Figure 4b are the same). Employing no correction results in the worst lifetimes. Almost all pages see one early fault and so this scheme's lifetime curve drops quite early and very sharply.

The relative endurance of $Perfect\_Code_9$ is worst for small modifications, as is illustrated in Figure 3a, and best for wider modifications that cover the entire block (Figure 3c). When modification widths are small, error-correction bits receive more wear than data bits and so the first bits to fail are likely to be the error-correction bits themselves. The same effect would be seen in SEC_{64} for writes that modify regions smaller than 64 bits within a 64-bit block, such as might occur when manipulating strings. One might even see this effect in $Pairing_8$ for writes that modify regions smaller than eight bits of an 8-bit block, such as those from writes to a Bloom filter.

In the middle of the endurance curves are the SEC_{64} and $Pairing_8$ schemes. As pages in the $Pairing_8$ scheme encounter their first bit error, they are paired with other pages and so once all pages have encountered their first bit error the number of available pages is cut in half. This capacity reduction doubles the effective wear on each page as there are half as many physical pages to spread the wear placed on the logical memory space. The SEC_{64} scheme suffers because the first occurrence of two errors within a 64-bit block is fatal.

The results for SEC_{64} differ from the SECDED results reported in Ipek *et al.* [11]. Like us, they implement $Pairing_8$ under the assumption that writes are followed by a verifying read. However, they compare to standard SECDED, which must deallocate a page after the first error is detected and corrected, since standard SECDED does not assume writes are re-read and verified. As a result, Ipek *et al.*'s standard implementation of SECDED with no verifying reads performs closer to our simulations of no error-correcting codes with verifying reads. When we extend SECDED to model a verifying read after each write, enabling deallocation only after the second error in a region, SEC_{64} outperforms $Pairing_8$ for lower coefficients of variance. As the variance grows, the lifetime of $Pairing_8$ improves relative to SEC_{64}, eventually exceeding it.

Wilkerson *et al.*'s scheme differs from ECP primarily in its use of a single-error-correction (SEC) Hamming code, instead of precedence rules, to correct errors in its own correction entries. Since the precedence rules incur no bit storage overhead there is no reason not to implement both. We thus enhance the $Wilkerson_4$ scheme with ECP's precedence rules. We find that the benefits of these Hamming codes over precedence rules alone are undetectable: when we graphed ECP_6 and $Wilkerson_6$ (not shown), their curves always overlapped completely; any difference in lifetime was so small as to be undetectable. Yet, Wilkerson *et al.*'s scheme can store only four correction entries within the 12.5% storage overhead constraint, whereas ECP can store six. ECP_6 outperforms the $Wilkerson_4$ scheme with similar storage overhead.

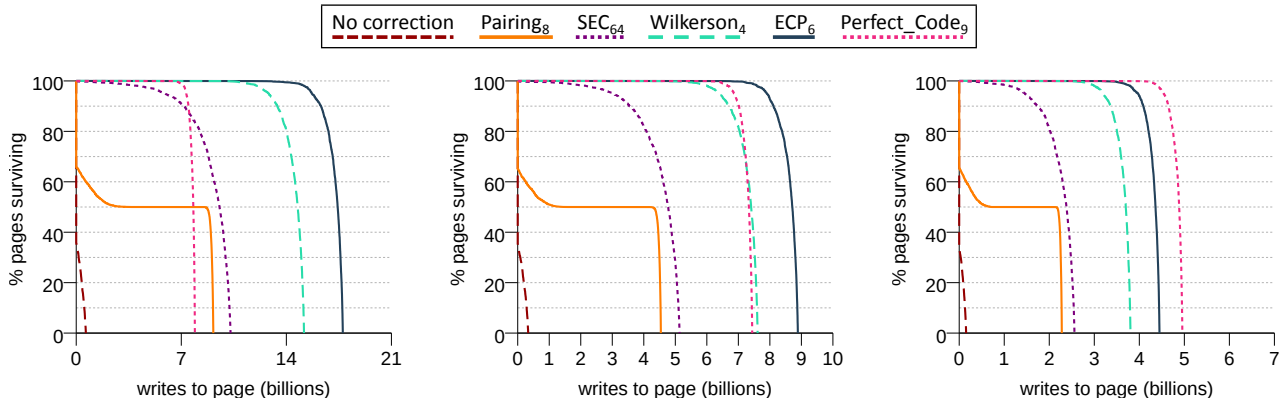
The ECP_6 scheme corrects two thirds of the errors possible with a perfect multi-error-correction Hamming code ($Perfect\_Code_9$) under the same space constraint. When the region of bits modified during a write is significantly smaller than 512 bits, pages using ECP_6 actually outlive those using a perfect multi-error-correction Hamming code. This advantage relative to a "perfect code" is possible because these write-modification widths cause Hamming codes to suffer more wear than the bits they are intended to correct (smaller average writes show larger relative wear on ECC bits). In contrast, ECP's correction pointers suffer near-zero wear and ECP's replacement bits suffer only as much wear as the data bits they replace.

6. INTRA-ROW WEAR LEVELING

The endurance of $Perfect\_Code_9$ suffers when writes to a block contain regions that are unmodified, as the correction bits suffer heavier wear than data bits and may fail first. We could compensate for this shortcoming by leveling wear throughout the block's data and correction cells. While throughout this paper we assume that writes are already wear-leveled across rows and pages, external wear-leveling mechanisms can only level wear within externally visible data cells; they cannot spread wear among the internal meta-data cells used for error-correction structures.

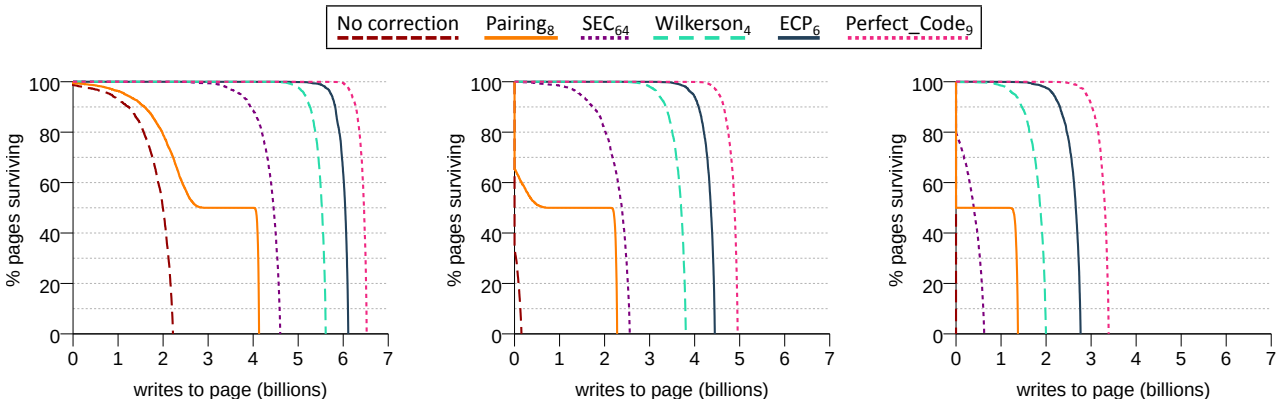
To address uneven wear between correction and data cells, we could architect rows (or blocks) to periodically rotate the positions of the logical row by a random offset to place them into different physical cells. This rotation would spread the impact of wear on the logical error-correction bits over all of the physical cells, presumably making $Perfect\_Code_9$ more competitive with ECP for write modification widths that do not span a full block.

In the next experiment, we introduce such a rotating wear leveler into $Perfect\_Code_9$ and ECP_6. The wear leveler rotates all logical structures (except its own) around random positions over a single set of physical cells, as shown in Figure 5a. We do not track the number of writes since the last rotation as doing so would incur additional space overhead and wear; we instead assume a scheme initiates rotations at random intervals with a uniform probability at each write. For a mean cell lifetime of 10^8 bits, we select a rotation probability of 10^{-4}. Since the probability that a given bit will change during a rotation is 0.5, the expected wear induced by the wear-leveling on each bit is $0.5 \cdot 10^{-4}$ bit-writes per write to the row. While infrequent enough to minimize the



(a) 128 bits (b) 256 bits (c) 512 bits

Figure 3: Page lifetimes in writes-per-page. A write of a 512-bit row may only modify a subset of the bits. The graphs assume writes that span (a) 128, (b) 256, and (c) 512 bits. Each bit within the span of modification changes value with probability 0.5.



(a) Cell lifetime variance = 0.2 (b) Cell lifetime variance = 0.25 (c) Cell lifetime variance = 0.3

Figure 4: Page lifetimes for various error-correction schemes for a row with 512 data bits, write width of 512 bits and various variances: (a) 0.2, (b) 0.25, and (c) 0.3.

probability of wear-failure within this structure, the rotator is expected to visit each possible rotation approximately twenty times over a period of 10^8 writes to a 512-bit row.

For a row of m data and repair bits, the rotating wear leveler requires $\lceil log_2 m \rceil$ bits to represent the rotation position. ECP_6 and $Perfect\_Code_9$ use rows/blocks of 512 data bits and have overhead of at most 64 bits (12.5%), and so 10 bits will be required to store the rotation position. The very low frequency of wear-leveling ensures that cell failures within the wear leveler itself are extremely rare. Rather than correct for these outliers, we simply accept that a negligible fraction of pages will reach the end of their lifetime as the result of a cell failure within a wear leveler, and not after n failures within a row. This assumption will not affect the experimental comparison, as the same wear leveler is used for all architectures and so a failure within the wear leveler is equally improbable for each of them.

In addition to using the rotating wear leveler to spread wear, we add an additional physical data cell to the space that the wear leveler can address. Thus, at any given time one cell is rotated out of use. This extra cell enables the scheme to repair one additional failed cell. Figure 5b shows a line using ECP_5 where all five correction entries have already been used up and a sixth error has been detected. By rotating this final error out of use, this scheme tolerates up to $n + 1$ failures per block, though after the final failure the wear leveler will be stuck in a single position until the terminal failure occurs. This cessation of rotation is likely to hasten the arrival of the terminal failure, as pointer bits may no longer absorb their share of wear, but it provides additional lifetime at the cost of only one extra bit.

Following a rotation, ECP implementations may need to re-discover failed bits that had been located in a correction pointer in the previous rotation, as ECP does not need to store these locations explicitly to correct them.

In Figure 6 we present the comparative page lifetimes for $Perfect\_Code_9$ and ECP_6, both equipped with rotating wear levelers. While wear-leveling significantly improves the relative endurance of $Perfect\_Code_9$ for smaller write-modification regions, ECP dominates $Perfect\_Code_9$ whenever the region of modified bits within a write does not span the full block; even when the extra error-correct-ing-code wear can be spread out, the wear has a significant impact on the overall block. ECP's relative endurance actually im-

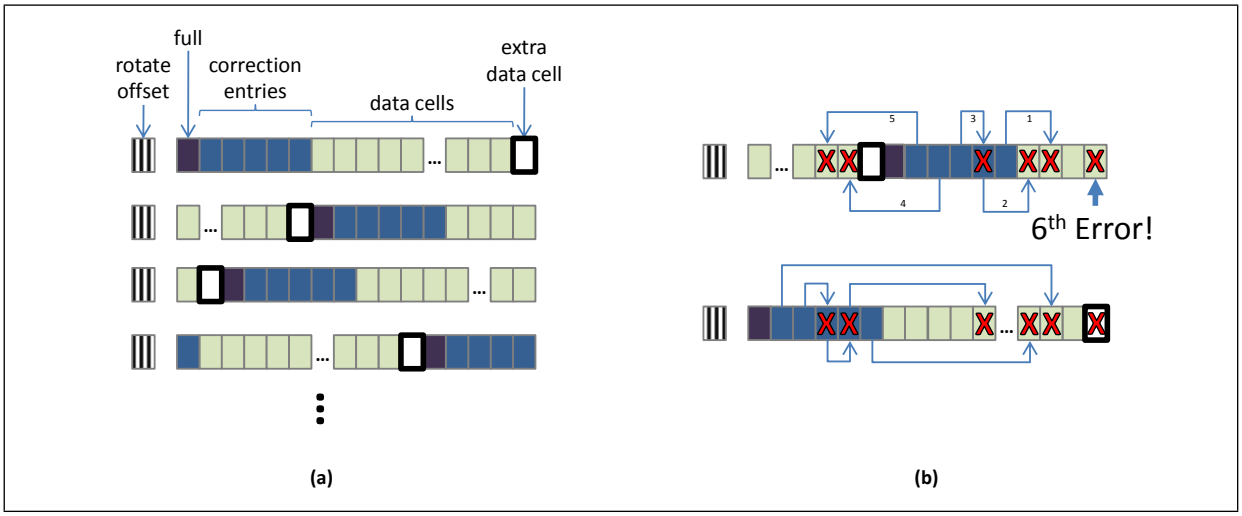


Figure 5: (a) The rotating wear leveler periodically rotates the entire PCM row including data and correction bits by a random offset. (b) The inclusion of one extra data cell provides a slight increase in expected lifetime and enables the tolerance of one additional bit failure. For ECP$_n$, the first n failures are handled by the correction entries, and the $n{+}1^{st}$ is covered by rotating the row such that the extra data bit aligns with the failed cell, although this comes at the cost that the wear leveler can no longer be used.

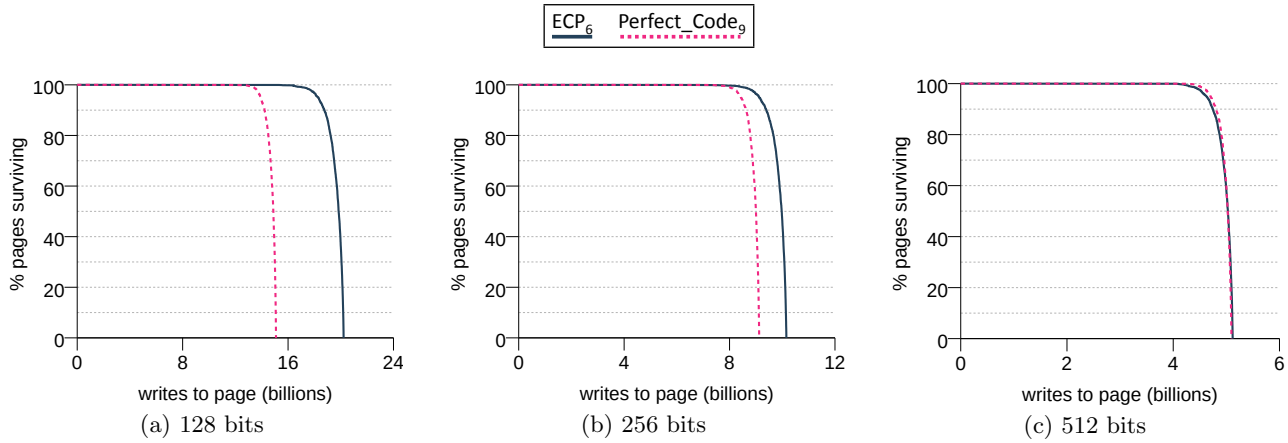


Figure 6: The comparative endurance of Perfect_Code$_9$ and ECP$_6$, using rotating wear leveling to spread wear evenly over both data bits and correction structures. The graphs assume writes that span (a) 128, (b) 256, (c) 512 bits.

proves when writes modify the whole block, as wear to the data and replacement bits is now spread to the address bits.

7. COMPARISON TO OPTIMAL ECP

A space-optimal version of ECP could store the locations of failed cells using a more compact representation than ECP. ECP does contain redundancy; for example, a two-pointer ECP organization could correct bit 4 in entry 0 and bit 5 in entry 1, or vice versa, resulting in two encodings to correct the same two bit failures. Ignoring precedence rules, ECP$_n$ has $n!$ different representations of the same n data bit failures (permutation of n error pointers).

An optimal replacement-cell scheme would ignore the unlikely event of failures in the low-wear error-locating mechanism, which is written to at most n times, and only correct failures of data and replacement bits. The optimal encoding only needs to identify failures in the $d{+}n{-}1$ repairable cells experiencing wear: the d data cells and the first $n-1$ replacement cells (pointing to a defect in the nth replacement cell would serve no purpose as there are no cells to replace it). The number of combinations of 0 to n error locations in these $d+n-1$ cells is:

$$\sum_{e=0}^{n} \binom{d+n-1}{e}$$

For a row size of 512 data bits ($d = 512$), a perfect replacement-cell scheme using the smallest possible representation of the failed bit locations would thus require sufficient space to represent the error location combinations above and to store the n replacement cells:

$$S(\text{Perfect\_Replacement}_n) = \frac{n + \lceil \log_2 \sum_{e=0}^{n} \binom{512+n-1}{e} \rceil}{512}$$

Decoding such a scheme would require significantly more overhead than decoding ECP; a naive approach would require multiple divisions. Since this scheme represents the

most storage-efficient pointing mechanism possible, it provides an important point of comparison to illustrate the room for improvement in compressing ECP. Table 3 shows the comparative storage overheads for schemes to correct one to ten errors. The relative overhead of ECP increases with the number of errors, since the number of redundant representations grows as $n!$. The overhead is acceptably small for most practical values of n.

8. LAYERING ECP

In the final experiment, we explore a multi-level ECP scheme, which attempts to provide the best of having many small restricted pointers and larger pointers that can correct any fault in a region. The small *row-level* pointers correct errors within a row, and the larger pointers can correct errors throughout a 4KB page. This extension allocates an extra row of 15-bit correction entries for each page – wide enough to correct any error in the page. These *page-level* correction entries would be allocated for errors within those rows that have exhausted their row-level correction entries.

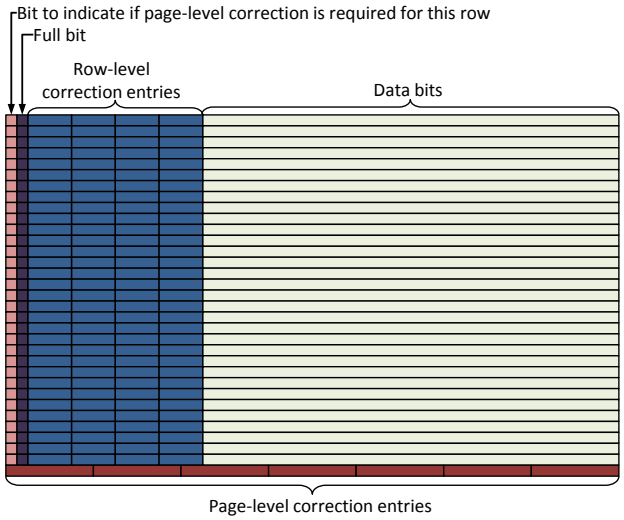


Figure 7: Hardware organization for layered ECP including both row-level ECP and page-level ECP. Row-level correction entries are applied after all reads. If the leftmost bit of the row-to-be-read is set, a second row containing page-level correction entries must also be read and will contain at least one entry that points within the row-to-be-read. All entries pointing within the row-to-be-read must be applied.

We call this scheme Layered_ECP$_{n,m}$, which has n row-level correction entries and m page-level correction entries. Since the page-level correction entries consume one row of physical memory cells, and the row width is a function of the number of row-level correction entries, m is a function of n. Figure 7 shows the organization of the original data rows (with per-row ECP) along with the additional page-level ECP entries (bottom row in the figure). Each row contains one extra bit that indicates whether any page-level entries have been allocated to correct errors in this row (*i.e.*, all of the row-level entries have already been activated and more errors need to be fixed). Reading/writing of the row storing the page-level correction entries is only needed when this bit is set.

Table 4 compares the endurance and space overhead of ECP$_x$ with Layered_ECP$_{x-1,y}$ (which includes the extra row for page-level correction entries). At similar overheads, Layered_ECP$_{x-1,y}$ has better relative endurance than ECP$_x$. However, the endurance gap narrows as the number of entries grow, and Layered_ECP$_{x-1,y}$ is more complex to implement and may incur additional performance and power overheads when accessing rows that require page-level correction.

9. OPTIMIZING MEMORY FOR CORRECTION

As resistive memories scale down to smaller geometries, and frequent faults become more prevalent with higher variation, it will be useful to determine how much overhead, for a specified lifetime variance, will be required to provide a desired capacity for a predetermined lifetime (measured in writes). This capability will allow architects to reason about cost and overheads for different failure rates. We next formulate the problem and walk the reader through the process of solving it.

Stated formally, we wish to build the smallest possible physical memory to store M bits of data (the *effective memory*) while withstanding W writes to the memory system (the *effective lifetime*).

The next step is to determine the *survival function* $s_n(w)$ for ECP$_n$: the fraction of memory with variation V surviving after w total writes, which can be done with simulations like those presented in this paper. Starting from m bits of physical storage, the fraction of physical memory available to store data after w writes, the *fractional effective memory*, is the product of the data fraction times the survival function.

$$f_n(m, w) = d_n \cdot s_n(w)$$

Figure 8 shows this fractional effective memory function plotted for ECP$_5$ with variance 0.3.

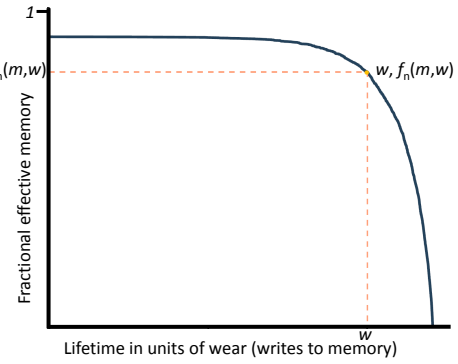


Figure 8: The fractional effective memory function plotted for ECP$_5$ with coefficient of variance 0.3.

The *effective memory*, the portion of the memory used to store data bits and that has survived w writes, is simply m times the fractional effective memory $e_n(m, w)$:

$$e_n(m, w) = m \cdot f_n(m, w) = m \cdot d_n \cdot s_n(w)$$

Architects have two tools with which to scale up the effective memory and lifetime to meet the target goals. They

Percentage space overhead for correction of n errors										
Errors correctable (n)	1	2	3	4	5	6	7	8	9	10
ECP$_n$	2.1%	4.1%	6.0%	8.0%	10.0%	11.9%	13.9%	15.8%	17.8%	19.7%
Perfect_Replacement$_n$	2.1%	3.9%	5.5%	7.0%	8.6%	10.0%	11.3%	12.7%	14.1%	15.4%

Table 3: Space overheads to correct 1 to 10 errors, comparing ECP with a storage-optimal replacement-cell encoding.

ECP											
Number of errors tolerated	0	1	2	3	4	5	6	7	8	9	10
Space overhead	0	2.1%	4.1%	6.0%	8.0%	10.0%	11.9%	13.9%	15.8%	17.8%	19.7%
Writes before 5% capacity drop (10^9)	0	0.6	1.9	2.6	3.2	3.6	3.9	4.2	4.4	4.7	4.9
Writes before 50% capacity drop (10^9)	0	1.6	2.6	3.2	3.7	4.1	4.4	4.6	4.8	5.0	5.2
ECPL											
Number of errors tolerated (row)		0	1	2	3	4	5	6	7	8	9
Number of errors tolerated (page)		32	32	33	33	34	35	35	36	37	37
Space overhead	-	1.8%	3.7%	5.7%	7.7%	9.7%	11.7%	13.6%	15.6%	17.6%	19.6%
Writes before 5% capacity drop (10^9)	-	2.6	3.5	4.1	4.4	4.8	5.0	5.2	5.4	5.6	5.7
Writes before 50% capacity drop (10^9)	-	2.8	3.7	4.2	4.6	4.9	5.1	5.3	5.5	5.7	5.8

Table 4: Space-overhead-equivalent comparison of ECP and layered ECP (ECPL).

can increase n, which increases the survival function but also increases the ECP overhead, reducing the fraction of memory used to store data. They can also increase the size of the physical memory by a multiplier k (to a total of km), for which we might wish to calculate the effective memory $e_n(km, W)$. This has two effects. First, the total amount memory is increased to km and so whatever fractional memory remains after page deaths is increased by a factor of k. Second, increasing the memory size by a factor of k spreads wear over k times as much memory, effectively dividing the wear by k. Thus, for a choice of n and k, the effective memory at lifetime W, $e_n(km, W)$, is equal to:

$$e_n(km, W) = km \cdot f_n(m, \frac{W}{k}) = k \cdot e_n(m, \frac{W}{k})$$

The optimal configuration is thus the one that results from finding the pair (n, k) that minimizes km and meets our constraint $M \leq e_n(km, W)$.

10. DISCUSSION

10.1 Hardware Implementation

The implementation of ECP (and the derivative schemes) can introduce additional delay in reading and writing the PCM. For ECP$_1$, the read operation needs the equivalent of a 9-to-512 bit row-decoder (for a 512-bit data block) to align the replacement bit with the dead cell being corrected. Figure 9a illustrates the logic for ECP$_1$. Instead of explicitly storing the replacement value, we use a differential encoding where the replacement bit conditionally inverts the failed bit. If the failed bit is stuck at the wrong value, then the replacement bit performs a correction by inverting the bad data value. The row decoder also requires an "enable" input such that all outputs are zero when the corresponding ECP has not yet been allocated.

A 512-way decoder has a reasonably small latency; consider that modern processors have branch prediction tables with thousands of entries that need to be accessed in a single cycle and that these row-decoders only account for a fraction of their overall access latencies. Furthermore, the access latency of the PCM array itself is already slow enough that the addition of some extra combinatorial logic will only have a minor effect on the final PCM access latency as well as on overall system performance.

Figure 9b shows the hardware organization for ECP$_5$; the logic in the shaded box is detailed in Figure 9c. One decoder per ECP entry is required. The chain of OR gates computes a prefix operation indicating whether any ECP entries to the left (*i.e.*, higher precedence) have been activated for this row. If a higher precedence entry exists, then the multiplexer will pass the corresponding replacement bit through to the right. At the end of the logic chain, if any ECP entry was activated for this row, the OR chain will evaluate to true and allow the highest-precedence replacement bit to pass through. If none of the row decoders are activated for this row, then the final AND gate outputs a zero; the differential encoding interprets this to not invert the corresponding data bit. Without the differential encoding, all of the XOR gates in the right portion of Figure 9b would have to be replaced by muxes and additional routing would be needed for the mux control inputs.

Although Figure 9c shows a linear chain of logic, for large n, the circuits can be replaced with a log-depth parallel prefix circuit. The total gate delay overhead would then be $O(\log d)$ for a d-way row decoder, $O(\log n)$ for a log-depth version of the logic illustrated in Figure 9c, and another $O(1)$ for the final AND and XOR gates. The total overhead is $O(\log n + \log d)$ which scales gracefully for both the number of ECP entries used and the width of a data block covered by each entry.

10.2 Orthogonality and Composability

The ECP scheme provides resilience to PCM cell failures within a memory row, allowing many more writes before the row is forcibly decommissioned. There have been several other recent proposals that can extend the lifetime of phase-change memories. Qureshi *et al.* proposed a hybrid DRAM/PCM organization where a write-tolerant DRAM

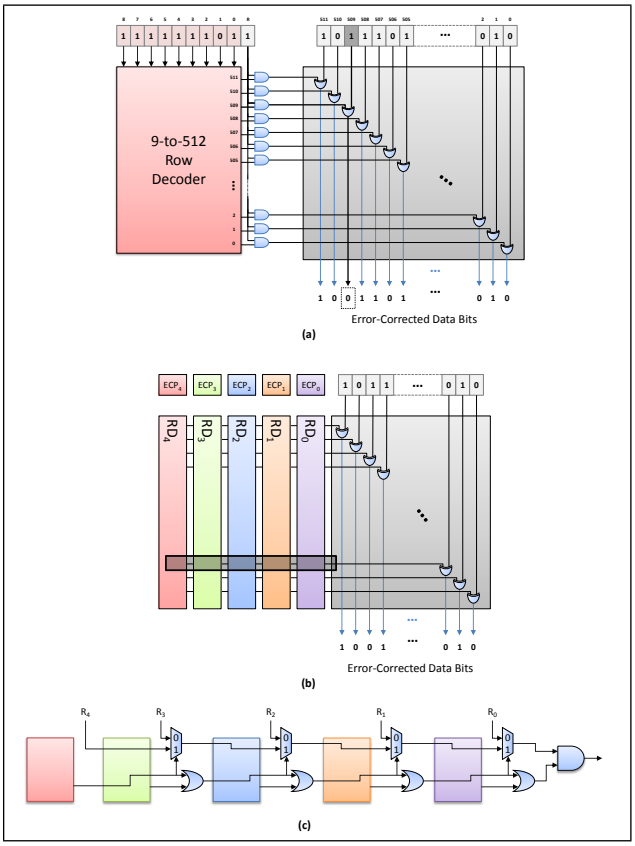


Figure 9: Hardware implementation for (a) ECP_1, (b) ECP_5, and (c) a close-up of one row of logic for ECP_5.

buffer can act as a large write combining buffer (among other optimizations) to condense many write commands from the processor into a single bulk write operation to the PCM [22]. The proposed ECP correction scheme does not come into play until the final write operations are presented to the PCM chips themselves, therefore techniques that reduce external write traffic before reaching the PCM can be directly composed with ECP to further extend the expected lifetime of PCM. These schemes do not compete with ECP but rather they complement each other.

Existing PCM wear-leveling schemes modify the mapping of rows within memory [21, 22, 28] so that the writes destined for frequently written rows are spread out over time across all of the available rows. In this fashion, no single row receives a disproportionately higher amount of the write traffic. Although the writes are now uniformly distributed across rows, the writes within a row may still be clustered. In the worst case, *all* of the writes may be destined for only a single bit (*e.g.*, a Boolean variable placed in memory using cache line padding to prevent false sharing). Each of the ECP and derivative schemes can be applied in conjunction with any memory-wide wear leveling scheme so that non-uniform write patterns are both *evenly distributed* across rows and *tolerated* within rows.

10.3 Transient Errors

The ECP approach assumes that soft errors are not a problem for phase-change memories; PCM cells are expected to reliably maintain their state for years under typical operating conditions. If transient faults occur with any reasonable probability (*e.g.*, operation in a high-temperature environment or multi-level cells that are much more sensitive to thermal drift), additional error correction beyond ECP will be required, which brings back all of the problems associated with traditional ECC codes.

If transient errors are not spatially located, one way to address them would be to layer a traditional error correction scheme (*e.g.* SEC_{64}) on top of ECP, using wear leveling to distribute the additional wear. The lower-level ECP would do the heavy lifting of correcting for multiple wear-induced failures while the SEC_{64} would correct for up to one transient error. SEC_{64} adds a 7 bit modification region (3.5 bit values are expected to flip) for each 64 bit block that is modified. If each modification spans all 64 bits of a modified block (32 bits are expected to flip), then adding SEC_{64} will result in an 11% increase in wear, spread over an additional 11% increase in storage bits.

11. CONCLUSIONS

The traditional approach for handling faults in main memories (SECDED ECC) works well for correcting rare transient faults, seen in DRAM, but not for correcting multiple wear-induced cell failures. ECC codes over small blocks cannot correct for multiple errors with reasonable space overhead. Using multiple-error correction Hamming codes over a larger block size reduces space overhead but, when data is written in small chunks, writing the large error correction codes can become a source of wear–inducing more errors.

Error-Correcting Pointers (ECP) function much better than Hamming codes for correcting multiple wear-induced hard faults. Like Hamming codes, ECP is able to efficiently correct errors in both data and in the correction structures themselves. Whereas Hamming codes must be updated each time a block of data is written, causing additional wear, ECP adds wear only in the rare case in which a new cell failure must be accounted for. The results show that ECP provides close to ideal correction capabilities and improved lifetime over previously proposed solutions at equivalent overheads. What's more, we provide with ECP the mathematical tools with which optimally allocate precious storage space between error-correction entries and spare memory when building a memory system to meet constraints of effective memory size, wear lifetime, and cell lifetime variance.

Acknowledgments

We are indebted to David Molnar and Thomas Moscibroda for their help in brainstorming and in providing feedback. Ed Nightingale was invaluable in helping us to understand prior work. We especially appreciate the insights of our anonymous reviewers, whose comments inspired numerous improvements. We are also thankful to Sean Eilert from Numonyx for providing us information about PCM architecture and failure models.

References

[1] Emerging research devices. In *International Technology Roadmap for Semiconductors*, 2007.

[2] S. J. Ahn, Y. J. Song, C. W. Jeong, J. M. Shin, Y. Fai, Y. N. Hwang, S. H. Lee, K. C. Ryoo, S. T. Lee, J. H. Park, H. Horii, Y. H. Ha, J. H. Yi, B. J. Kuh, G. H. Koh, G. T. Jeong, H. S. Jeong, K. Kim, and B. I. Ryu. Highly manufacturable high density phase change memory of 64Mb and beyond. In *International Electron Devices Meeting*, 2004.

[3] G. Atwood and R. Bez. Current status of chalcogenide phase change memory. In *Device Research Conference*, 2005.

[4] F. Bedeschi, R. Fackenthal, C. Resta, E. M. Donze, M. Jagasivamani, E. C. Buda, F. Pellizzer, D. W. Chow, A. Cabrini, G. Calvi, R. Faravelli, A. Fantini, G. Torelli, D. Mills, R. Gastaldi, and G. Casagrande. A multi-level-cell bipolar-selected phase-change memory. In *International Solid-State Circuits Conference*, 2008.

[5] A. J. Bhavnagarwala, X. Tang, and J. D. Meindl. The impact of intrinsic device fluctuations on CMOS SRAM cell stability. *IEEE Journal of Solid-State Circuits*, 36(4), Apr. 2001.

[6] S. Borkar, T. Karnik, S. Narendra, J. Tschanz, A. Keshavarzi, and V. De. Parameter variation and impact on circuits and microarchitecture. In *Proceedings of 40th Design Automation Conference*, June 2003.

[7] G. Dhiman, R. Ayoub, and T. Rosing. PDRAM: A hybrid PRAM and DRAM main memory system. In *Proceedings of 47th Design Automation Conference*, June 2009.

[8] R. Freitas and W. Wickle. Storage-class memory: The next storage system technology. *IBM Journal of Research and Development*, 52(4/5):439–447, 2008.

[9] R. Hamming. Error detecting and error correcting codes. *Bell System Technical Journal*, 29(2), April 1950.

[10] H. Horii, J. H. Yi, J. H. Park, Y. H. Ha, I. H. Baek, S. O. Park, Y. N. Hwang, S. H. Lee, Y. T. Kim, K. H. Lee, U.-I. Chung, and J. T. Moon. A novel cell technology using N-doped GeSbTe films for phase change RAM. In *Symposium on VLSI Technology*, 2003.

[11] E. Ipek, J. Condit, E. Nightingale, D. Burger, and T. Moscibroda. Dynamically replicated memory: Building resilient systems from unreliable nanoscale memories. In *To appear at The Fifteenth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS 2010)*, Mar. 2010.

[12] K. Kim. Technology for sub-50nm DRAM and NAND flash manufacturing. In *International Electron Devices Meeting*, 2005.

[13] B. Lee, E. Ipek, O. Mutlu, and D. Burger. Architecting phase-change memory as a scalable DRAM alternative. In *International Symposium on Computer Architecture*, June 2009.

[14] K.-J. Lee, B.-H. Cho, W.-Y. Cho, S. Kang, B.-G. Choi, H.-R. Oh, C.-S. Lee, H.-J. Kim, J.-M. Park, Q. Wang, M.-H. Park, Y.-H. Ro, J.-Y. Choi, K.-S. Kim, Y.-R. Kim, I.-C. Shin, K.-W. Lim, H.-K. Cho, C.-H. Choi, W.-R. Chung, D.-E. Kim, Y.-J. Yoon, K.-S. Yi, G.-T. Jeong, H.-S. Jeong, C.-K. Kwak, C.-H. Kim, and K. Kim. A 90nm 1.8V 512Mb diode-switch PRAM with 266 MB/s read throughput. *Journal of Solid-State Circuits*, 43(1), January 2008.

[15] T. May and W. Woods. Alpha-particle-induced soft errors in dynamic memories. *IEEE Transactions on Electronic Devices*, 26(2):2–9, 1979.

[16] C. McNairy and R. Bhatia. Montecito: A dual-core, dual-thread itanium processor. *IEEE Micro Magazine*, 25(2):10–20, March/April 2005.

[17] S. S. Mukherjee, J. Emer, and S. K. Reinhardt. The soft error problem: An architectural perspective. In *Proceedings of 11th International Symposium on High-Performance Computer Architecture*, pages 243–247, San Francisco, CA, USA, February 2005.

[18] Numonyx. The basics of phase change memory technology. In *Numonyx White Paper*, 2007.

[19] M. Orshansky, L. Milor, P. Chen, K. Keutzer, and C. Hu. Impact of spatial intrachip gate length variability on the performance of high-speed digital circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 21(5), May 2002.

[20] F. Pellizzer, A. Benvenuti, B. Gleixner, Y. Kim, B. Johnson, M. Magistretti, T. Marangon, A. Pirovano, R. Bez, and G. Atwood. A 90nm phase change memory technology for stand-alone non-volatile memory applications. In *Symposium on VLSI Circuits*, 2006.

[21] M. K. Qureshi, M. Fraceschini, V. Srinivasan, L. Lastras, B. Abali, and J. Karidis. Enhancing lifetime and security of phase change memories via start-gap wear leveling. In *International Symposium on Microarchitecture*, December 2009.

[22] M. K. Qureshi, V. Srinivasan, and J. A. Rivers. Scalable high performance main memory system using phase-change memory technology. In *International Symposium on Computer Architecture*, June 2009.

[23] S. Raoux, G. W. Burr, M. J. Breitwisch, C. T. Rettner, Y.-C. Chen, R. M. Shelby, M. Salinga, D. Krebs, S.-H. Chen, H.-L. Lung, and C. H. Lam. Phase-change random access memory: A scalable technology. *IBM Journal of Research and Development*, 52(4/5), Jul/Sept 2008.

[24] Samsung. Samsung introduces the next generation of nonvolatile memory - pram. In *Samsung News Release*, Sept. 2006.

[25] 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 *The 35th Annual International Symposium on Computer Architecture*, June 2008.

[26] W. Zhang and T. Li. Characterizing and mitigating the impact of process variations on phase change memory systems. In *International Symposium on Microarchitecture*, December 2009.

[27] W. Zhang and T. Li. Exploring phase change memory and 3d die-stacking for power/thermal friendly, fast and durable memory architectures. In *International Conference on Parallel Architectures and Compilation Techniques*, Sept. 2009.

[28] P. Zhouand, B. Zhao, J. Yang, and Y. Zhang. A durable and energy efficient main memory using phase change memory technology. In *International Symposium on Computer Architecture*, June 2009.

[29] J. F. Ziegler and G. R. Srinivasan. Terrestrial cosmic rays and soft errors. *IBM Journal of Research and Development*, 40(1), 1996.