A CASE FOR NONUNIFORM FAULT TOLERANCE IN EMERGING MEMORIES

Contributor

Moinuddin K. Qureshi
Georgia Institute of Technology

As DRAM systems face scalability challenges, the architecture community has started investigating alternative technologies for main memory. These emerging memory technologies tend to suffer from the problem of limited write endurance. This problem is exacerbated because of the high variability in lifetime across different cells, resulting in weaker cells failing much earlier than nominal cells. Ensuring long lifetimes under high variability requires that the design can correct a large number of errors for any given memory line. Unfortunately, supporting high levels of error correction for all lines incurs significantly high overhead, often exceeding 10 percent of overall memory capacity. We propose to reduce the storage required for error correction by exploiting the observation that only a few lines require high levels of hard-error correction. Therefore, prior approaches that uniformly allocated a large number of error correction entries for all lines are inefficient, as most (more than 90 percent) of these entries remain unused. We propose Pay-As-You-Go (PAYG), an efficient hard-error resilient architecture that allocates error correction entries in proportion to the number of hard faults in the line. We describe a storage-efficient and low-latency organization for PAYG. Compared to uniform error correction, PAYG requires one third the storage overhead and yet provides 13 percent more lifetime.

Introduction

As DRAM-based memory systems get limited by power and scalability challenges, architects are turning their attention towards emerging memory technologies for building future systems. Phase Change Memory (PCM) has emerged as one of the most promising technologies suitable for incorporation into main memory.[3] While PCM has several desirable attributes such as improved scalability and nonvolatility, the physical properties of PCM dictates that only a limited number of writes can be performed to each cell. On average, PCM devices are expected to last for about 10 to the 7th and 10 to the 8th, writes per cell.[1] Once a cell reaches its end of life, it gets stuck in one of the states, manifesting itself as a hard error. The problem of limited lifetime is further exacerbated by the high variability in lifetime across different cells due to process variations. This means a small percentage of cells that have a significantly lower than average lifetime end up determining the overall lifetime of the system.

"…a small percentage of cells that have a significantly lower than average lifetime end up determining the overall lifetime of the system."

Ensuring reasonable system lifetime under high variability requires that the design provision large amounts of error correction for PCM lines. As we are concerned with lifetime failures that manifest themselves as hard errors, we focus only on hard-error correction in this article. Recent studies have proposed write-efficient error correction schemes such as *Error Correction*

Pointers (ECP)[5] and *SAFER*[6] to tolerate a large number of hard faults in memory lines. While our analysis is applicable to any hard-error correction scheme, we discuss ECP for our studies owing to its simplicity.

ECP corrects a failed bit in a memory line by recording the position of the bit in the line and its correct value. For example, a 64-byte (512-bit) line needs a 9-bit pointer plus 1 replacement bit resulting in a total of 10 bits for each ECP entry. Our evaluations show that correcting six errors per line can provide a lifetime of about 6.5 years for our baseline (the configuration is described in the section "Experimental Methodology"). Provisioning for 6 bits of error correction requires an overhead of 61 bits (60 bits of ECP plus one full bit to indicate that all ECP entries are used) per line, which translates to a total storage overhead of 12 percent. Note that this level of error correction would not be an optional feature in future PCM systems but rather something that would be essential to enable meaningful operation of the PCM array. Given that the memory market is low margin and highly cost-sensitive, it is important that the storage overhead of such necessary error correction be minimized, while retaining the desired levels of reliability. Thus, the 12 percent storage overhead of ECP may very well prove to be too high for wide-scale adoption of PCM.

To reduce the storage overhead of error correction, we begin by pointing to the inefficiency with the ECP approach that uniformly allocates six ECP entries per line. Our analysis shows that very few lines are weak, and more than 95 percent of the lines require no more than one ECP entry per line. Therefore, we would expect that with uniform ECP-6, the majority of the ECP entries would remain unused. Table 1 shows the distribution of lines that use a given number of ECP entries at different aging levels (age normalized to the lifetime under ECP-6, or 6.5 years). The average number of ECP entries used is also shown.

"Our analysis shows that very few lines are weak, and more than 95 percent of the lines require no more than one ECP entry per line."

Number Writes (Normalized Age)	Number of ECP Entries Used per Line				Average Number of ECP Entries Used
	0	1	2	3–6	
50%	99.02%	0.97%	0.00%	0.00%	0.010
90%	84.76%	14.02%	1.16%	0.07%	0.165
95%	79.63%	18.14%	2.06%	0.17%	0.228
100%	73.24%	22.82%	3.55%	0.40%	0.311

Table 1: Inefficiency of Uniform ECP-6. On average, only 0.3 out of six entries eventually gets used
(Source: Moinuddin K. Qureshi, 2013)

As the number of writes increase, the rate of faults increases, and hence more and more of the allocated ECP entries get used. However, even at the end of the expected system lifetime under ECP-6, less than 5 percent of the lines utilize more than one ECP entry. On average, only 0.3 entries out of the allocated six entries of ECP get used, indicating significant inefficiency with

uniform ECP. If we could allocate ECP entries only to lines that need those entries, we would reduce the required ECP entries by almost 20X. Ideally, we want to allocate more ECP entries to weak lines (lines with large number of errors) and fewer ECP entries to other lines. Unfortunately, uniform ECP allocates a large (and wasteful) number of ECP entries with each line a priori, being agnostic of the variability in lifetime of each line.

"…Pay-As-You-Go (PAYG, pronounced as "page"), an error correction architecture that allocates error correction entries in response to the number of errors in the given memory line."

We propose Pay-As-You-Go (PAYG, pronounced as "page"), an error correction architecture that allocates error correction entries in response to the number of errors in the given memory line. To maintain low latency of error correction, PAYG splits the correction entries into two parts: first, a per-line Local Error Correction (LEC) that can correct up to one error per line and is sufficient for 95 percent of the lines; and second, a Global Error Correction (GEC) pool that contains tagged ECP entries and provides error correction entries for lines that have more errors than can be handled by the LEC.

We describe several versions of PAYG, each with varying effectiveness, storage overhead, and latency overhead. Our evaluations show that PAYG reduces the storage overhead of error correction by a factor of 3.1X compared to ECP-6 (19.5 bits per line vs. 61 bits per line) while still obtaining 13 percent longer lifetime. Thus, PAYG obtains the best of both worlds in that it achieves the lifetime corresponding to strong levels of error correction while maintaining the low storage overhead that is sufficient for most of the lines.

Background

The problem of limited write endurance is common to many of the emerging memory technologies. Without loss of generality, this article analyzes Phase Change Memory (PCM) as an example of emerging memory technology. PCM suffers from the limited endurance in that the memory cells cease to have the ability to store data after a certain number of writes. Such cells get stuck to one of the states and manifest themselves as hard errors.[5] Designing a robust PCM system that can last for several years requires carefully architecting the system to tolerate such errors.

Problem: Variability in Lifetime

ITRS[1] projections (and various other studies) indicate that PCM cells can be expected to have an average endurance in the range of 10^7–10^8 writes. While this range of endurance is much lower than the ~10^{15} endurance of DRAM, it is still sufficient to architect a system with several (more than five) years of lifetime. Unfortunately, the lifetime of PCM cells is not uniform, and process variability results in significant variations even within adjacent cells in the same die.[2][5] This causes certain cells to have much lower endurance than the average population. Such weak cells fail much earlier than the typical cell and can reduce the lifetime of the system significantly to the tune of a few weeks.

The variation in lifetime is typically expressed as normalized standard deviation (COV) around the mean. Previous studies on variability of PCM endurance have used COV values between 10–30 percent of the mean.[2][5][6] In our

analysis, we use a default COV value of 20 percent. With a COV of 20 percent, the cell failure probability at the very start is in the range of 10^{-6}. Given that a typical main memory system contains tens of billions of cells, even this small failure probability would result in several thousand cells having bit failures, which in turn would result in a drastic reduction in the overall system lifetime because of variability.

"…a typical main memory system contains tens of billions of cells, even this small failure probability would result in several thousand cells having bit failures,…"

Prior Work

The lifetime of a PCM system can be increased to a useful range if the system can tolerate errors. Hamming code-based error correction, which is typically employed in memory systems, can tolerate transient errors as well as hard errors. Unfortunately, such codes are write intensive and can further exacerbate the endurance problem in PCM. Fortunately, identifying the endurance-related write failures is easy as it can be done by simply performing a verify read after completing a write.[1] If the two values do not match then the nonmatching bit is likely to be a hard error.

Recent studies[5][6] have focused on developing write-efficient methods to provide error correction of hard faults, relying on this simple detection property of endurance-related failures.

One such proposal is Error Correcting Pointers (ECP).[5] ECP performs error correction by logging bit errors in a given line. For example, for a line of 64 bytes (512 bits), a 9-bit pointer is used to point to the failing bit and an additional bit to indicate the correct value. This scheme can correct one error and is referred to as ECP-1. The concept can be extended to correct multiple bits per line. Intelligent precedence rules allow correction of errors even in the ECP entries. A generalized scheme that can correct N errors per line is called ECP-N. A full bit per line indicates if all the ECP entries associated with the line are used. Thus, the storage overhead of ECP-N is $(10N + 1)$ bits per line.

Need to Correct Several Errors per Line

Given that transient faults are rare, a typical memory system is designed to handle at most one or two transient faults per line. However, unlike transient faults, endurance-related hard errors accumulate over time. Therefore, we need to provide large amount of error correction per line in order to obtain reasonable system lifetime. Figure 1 shows the mean time to first uncorrectable error for our baseline system, where the number of ECP entries per line is varied from 1 to 12. All lifetime numbers are normalized to the case of zero variance. To show dependence of lifetime on variance, we show data for different COVs. For COV = 20 percent, ECP-6 obtains 35 percent of ideal

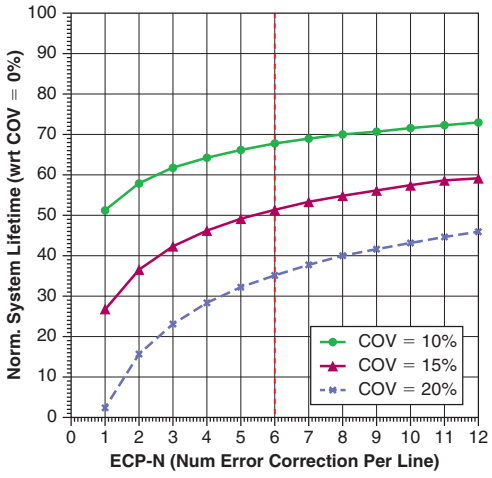

Figure 1: Normalized value of system lifetime (defined as the mean time of first uncorrectable error) as a function of the ECP strategy. The system lifetime is normalized with respect to a memory cell with 0% COV. Note that at COV of 20%, ECP-6 obtains 35% of theoretical maximum lifetime
(Source: Moinuddin K. Qureshi, 2013)

[1] If the system supports some amount of transient fault protection with each line, then we can identify the hard faults without performing the verify read. For example, the position of a bit that causes a failure with a transient fault protection mechanism can be tracked. Given that transient faults are rare, if the same bit position is causing frequent errors then that bit is likely to be a hard fault. Such a bit can then be corrected using a hard-error correction mechanism. This article assumes that an efficient means of detecting endurance-related failures exists and focuses only on correcting such failures.

"ECP-6 incurs storage of 61 bits per line, which translates to 12 percent storage overhead. Given that memory chips are extremely cost sensitive, such overhead may be too high for practical use."

lifetime. For our baseline, this translates to a lifetime of 6.5 years, which is in the desired range of 5–7 years for a typical server. ECP-6 incurs storage of 61 bits per line, which translates to 12 percent storage overhead. Given that memory chips are extremely cost sensitive, such overhead may be too high for practical use.

Inefficiency of Traditional Approach

For a memory of N lines, a PCM system would provision a total of $6N$ ECP entries to implement ECP-6. The problem with such an approach is that it results in significantly underutilized ECP entries. Because weak lines are few, only a few lines require high levels of error correction. Most of the other lines do not use the allocated ECP entries. Figure 2 shows the failure probability as the number of writes is increased (under COV = 20 percent), normalized to a system that has zero variance. The failure of line (or system) occurs when there is at least one uncorrectable error for a given amount of ECP. The expected time to failure is computed as the time at which the failure probability is 50 percent.

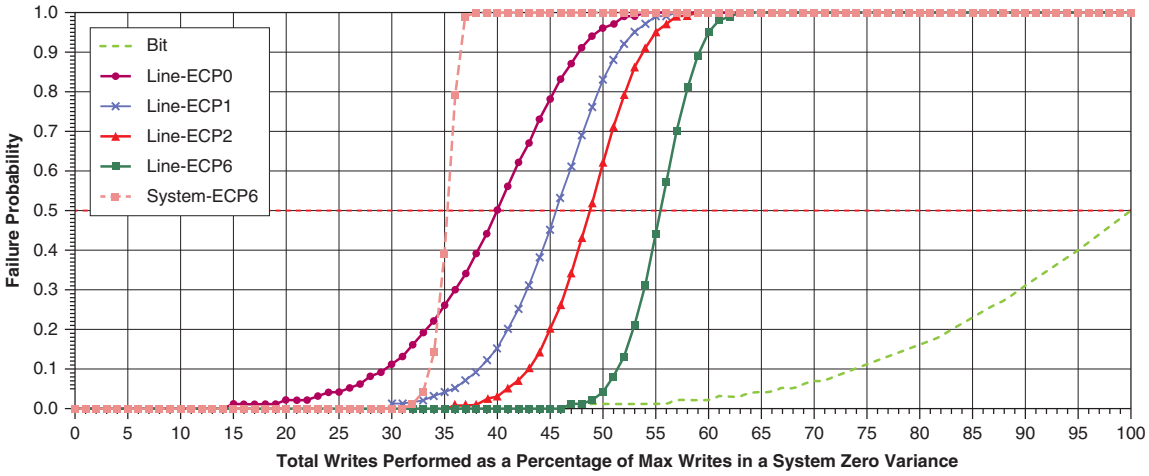

Figure 2: Failure probability vs. the percentage of writes assuming a COV = 20%, normalized to a system that has zero variance. "Bit" shows probability of failure of a single bit. "Line-ECPN" shows the probability of failure of a single line if N bits can be corrected. "System-ECP6" shows the probability that "at least one line fails out of all the lines" when each line has ECP-6. Observe that when the system failure is expected to occur under ECP-6, the probability of line failure with ECP-1 is approximately 3.5%
(Source: Moinuddin K. Qureshi, 2013)

Given that memory has millions of lines, the line failure probability must be very low (much less than 10^{-6}) to achieve a low system failure probability. When the system failure is expected to occur under ECP-6, the probability of line failure with ECP-1 is approximately 3.5 percent. This implies that fewer than 5 percent of the lines have more than one failed bit at the time of system failure, indicating significant inefficiency in the traditional approach that allocates six ECP entries for all lines. We note that ECP-1 is sufficient in the common case, and we need higher levels of ECP for very few lines. Ideally,

we would like to retain the robustness of ECP-6 while paying the hardware overhead of only ECP-1.

We base our solution on the insight that hard errors are quite different from transient faults. We need to allocate the storage for the error detection of transient faults up-front—before the error occurs. However, for hard errors, we can detect the error using a separate mechanism and allocate the error correction entry only when the error occurs. We discuss our experimental methodology before describing our proposal.

Experimental Methodology

The following section describes our experimental methodology.

Baseline Configuration

We assume a memory configuration that is designed with PCM banks each with 1 GB memory. Each bank has one write port and the write operation can be performed with a latency of 1 microsecond. The size of the line in the last-level cache is 64 bytes, which means there are 2^{24} lines in each bank. All operations on memory occur at line-size granularity. Given that each bank is a separate entity and can be written independently, we focus on determining the lifetime of one bank. We assume that each line has an endurance of 2^{25} writes. If endurance variance was 0 percent, we would expect the baseline to have a lifetime of 18 years.[2] ECP-6 obtains 35 percent of this lifetime, which translates to 6.5 years.

Assumptions

We are interested in evaluating the lifetime of memory, which is typically in the range of several years. Modeling a system for such a long time period inevitably involves making some simplifying assumptions. We make the following assumptions in order to evaluate memory lifetime:

- We assume the lifetime of each memory cell to follow a normal distribution without any correlation between neighboring cells. We assume a mean lifetime of 2^{25} writes[4] and a COV of 20 percent of the mean.
- We assume perfect wear-leveling to focus only on the impact of the error correction schemes. This implies that all the memory lines will receive the same number of writes.
- A write request to memory is converted into a sequence of write requests followed by a read request to detect hard faults. We assume that this technique can identify hard faults with 100 percent accuracy.

Figure of Merit

The endurance-limited lifetime of the system can be defined as the number of writes performed before encountering first uncorrectable error. Thus, for a given scheme, lifetime is determined by the first line that gets more errors than

[2] Each of the 224 lines can be written 225 times, for a total of 249 writes. With write latency of 1 microsecond, we can perform 106 writes/second or 244.8 writes per year, hence, a lifetime of 18 years, even under continuous write traffic.

can be corrected. ECP-6 obtains a lifetime of 6.5 years, which is in the range of 5–7 years of lifetime for a typical server. We want a lifetime in this range; hence all lifetime numbers in our evaluation are normalized to ECP-6. We define Normalized Lifetime (NL) as follows, and use this as the figure-of-merit in our evaluations:

$$NL = \frac{\text{Total Line Writes Before System Failure} \times 100\%}{\text{Total Line Writes Before System Failure With ECP 6}} \quad (1)$$

Pay-As-You-Go Error Correction

"…PAYG, provides storage-efficient on-demand error correction while incurring negligible latency overhead."

We can architect an efficient and robust design by allocating error correction entries only on demand, as and when an error occurs. In fact, one can reduce the percentage of unused ECP entries to zero by having a fully associative structure where each entry contains one tagged ECP-1 unit. Unfortunately, such a design would incur intolerable latency as each memory access would need to search through hundreds of thousands of error-correction entries. Our proposed design, PAYG, provides storage-efficient on-demand error correction while incurring negligible latency overhead. In this section, we first start with a naive design for PAYG, identify its shortcomings, and then propose the robust design.

Architecture of Naive PAYG

When failure occurs under ECP-6, we observe that 73 percent of the lines have 0 errors (Table 1). Hence, error correction overhead could be decreased by ~4x, by allocating ECP-6 only for lines that have at least one error. This simplified architecture is called Naive-PAYG, and is shown in Figure 3.

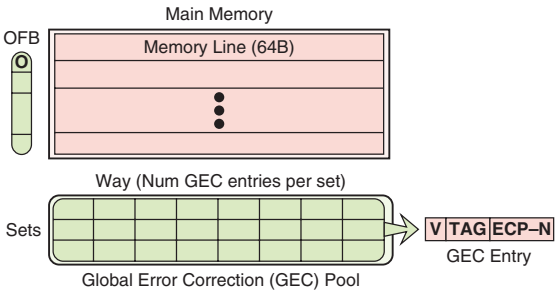

Figure 3: Architecture of Naive-PAYG (newly added structures are shaded)
(Source: Moinuddin K. Qureshi, 2013)

Each line contains an overflow bit (OFB) to indicate if the line has at least one failed bit.[3]

A Global Error Correction (GEC) pool provides error correction entries for such lines. Each GEC entry contains a valid bit, a tag (to identify the owner

[3] A stuck-at-zero OFB can be a single point of failure. Under COV = 20 percent, the probability that a bit will fail at first write is 0.3×10^{-6}. Given 16 million lines in memory, 4.8 lines are expected to have such a failure on average. We avoid this problem by using two-way replication for the OFB bit. We assume that OFB is set to 1, if any of the replicated bits is 1. The probability that both the replicated bits of OFB are stuck-at-zero is negligible (10^{-13}).

line), and one or more ECP entries (ECP-6 in our case). GEC is organized as a set-associative structure. Given that memory designs are highly optimized for a given array size, we want to use a line-size granularity for GEC as well. Therefore one set of GEC is sized such that it fits in 64 bytes, translating to seven GEC entries per set. We found that such a design is noncompetitive compared to even uniform ECP-6 because it suffers from three problems:

- The set associative organization needs a much larger number of entries (~8x) than a fully associative structure to reach the same level of effectiveness.
- Even with the filtering provided by the OFB, 25 percent of the lines can still incur a latency of two accesses (one for main memory and second for GEC), resulting in significant slowdowns.
- Most ECP entries remain unused as six ECP entries are allocated for lines with even one error.

We now describe efficient solutions to each of these problems, leading up to our final design.

Addressing Problem 1: Shortcoming of Set Associative Structure for GEC Pool

In a set-associative organization, each set has only a fixed number of ways, which means that the first set to exceed its allocation causes an uncorrectable failure. So, an important question in determining efficiency of the set-associative structure is to analyze the number of GEC entries occupied before one of the sets overflows. Given that most of the efficient wear-leveling algorithms[4][7] randomize the address space in PCM, we assume that failures occur at random lines in memory, and that any access pattern gets spread over the entire memory (due to remapping from wear leveling). Based on this randomized address space property, we can analyze the effective capacity utilization of a set-associative structure using an analogous buckets-and-balls problem, where a bucket represents one of the sets and a ball represents one of the occupied ways. If there are N buckets, each of which can hold B balls, then the collection can hold a maximum of NB balls. However, if balls are thrown at random, then how many balls can be thrown before one of the buckets overflows? Our Monte Carlo simulations indicate that a 7-way or 8-way GEC pool is only about 12 percent occupied when one of the sets overflows, indicating about 8x inefficiency with a set-associative structure.

Ideally, we want the efficiency of a fully associative structure (where all entries get used) and latency of set-associative structure (single low-latency index). To handle these contradictory requirements, we use a hash-table-with-chaining structure. It consists of two tables: first, the *Set Associative Table (SAT)* and second, the *Global Collision Table (GCT)*. SAT provides a single-index low-latency access to the GEC pool, while GCT provides flexibility in placement. Both SAT and GCT are structurally identical and differ only in the way they

"Ideally, we want the efficiency of a fully associative structure (where all entries get used) and latency of set-associative structure (single low-latency index)."

are indexed. Each GEC set (both in SAT and GCT) also contains a pointer (GCTPTR) that points to a location in the GCT.[4] The proposed GEC structure is shown in Figure 4.

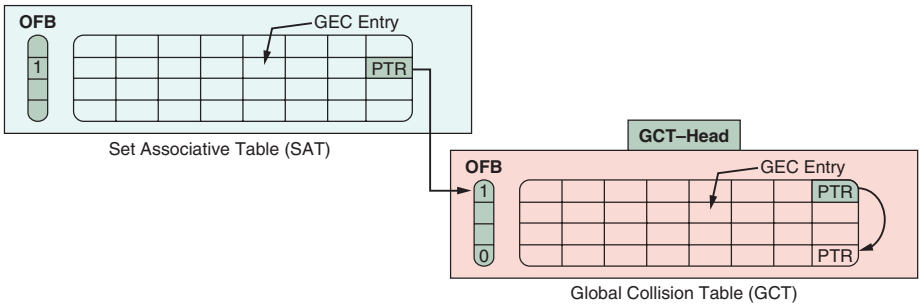

Figure 4: Architecture of scalable GEC pool (Set Associative Table + Global Collision Table) (Source: Moinuddin K. Qureshi, 2013)

Reading GEC Entries

For obtaining a GEC entry, SAT is accessed first in a set that is indexed by some bits of the line address. If there is no tag match in the set, then the GCTPTR of that set identifies the GCT set that must be checked. GCT can be indexed only in this manner. If there is a tag match in the GCT row, then GEC entries can be obtained. If there is no match, the GCTPTR in that set identifies the next GCT set that must be checked. The traversal continues until a GCT entry with matching tag (or a set with OFB = 0) is found.

Allocating GEC Entries

Initially, all GCT sets remain unallocated. These sets get allocated to a set of SAT only on overflow. To aid this allocation, a register called GCT-Head keeps track of the number of GCT entries that have been allocated. When one of the set of SAT or GCT overflows, the GCTPTR of that set is initialized to GCT-Head and the OFB associated with that set is set to 1. The newly allocated set of GCT provides as many GEC entries as the associativity of GCT. The GCTPTR of this newly allocated entry is marked invalid and OFB is set to 0 (to indicate end of traversal).

The GCT-Head is incremented after every GCT allocation. When the value of GCT-Head reaches the number of sets in GCT, it indicates an uncorrectable error.

We use a GCT that has half as many sets as SAT. Table 2 shows the effective capacity if there are N sets in SAT and $0.5N$ sets in GCT, as the associativity of SAT is varied. For an 8-way SAT, our organization obtains an effective capacity of more than 70 percent of the allocated $1.5N$ entries, much higher than the 12 percent with a set-associative structure.

"…our organization obtains an effective capacity of more than 70 percent of the allocated 1.5N entries, much higher than the 12 percent with a set-associative structure."

[4] We use two-way replication for GCTPTR for tolerating errors. We force the GCTPTR with a single stuck-at-bit to point to either location all-zeros or all-ones (both locations are reserved). On mismatch between the two copies of GCTPTR, the entry pointing to the reserved location is ignored. The probability of two bits stuck in GCTPTR is negligible (10^{-12}).

Associativity of SAT	1	2	4	8
Effective Capacity	1.19*N*	1.15*N*	1.11*N*	1.08*N*

Table 2: Effective capacity utilization (of 1.5N entries) with proposed (SAT+GCT) organization
(Source: Moinuddin K. Qureshi, 2013)

In the common case, we want the access to be satisfied by SAT and not the GCT, as GCT incurs higher latency due to multiple memory accesses. Our Monte Carlo simulations show that until about half the entries in SAT get occupied, the probability of single GCT access remains low (less than 1 percent). Thus, the proposed design has a good storage efficiency as well as low latency.

Addressing Problem 2: Local Error Correction for Low-Latency

One of the shortcomings of the naive design is that it accesses the GEC for a line with even one error. We can reduce latency and storage requirements for GEC by allocating a small amount of error correction with each line. For example, we observe that with ECP-1, the likelihood of failure is less than 4 percent even at the end of system lifetime. Therefore if we allocate ECP-1 with each line, we can reduce the GEC access rate as well as demand significantly. We propose to have such Local Error Correction (LEC) with each line. When the number of errors in the line exceeds what can be corrected by LEC, the OFB associated with that line is set and an entry from GEC is allocated. With ECP-1 in LEC, each GEC entry would need to store only ECP-5, which means the GEC can be an 8-way structure in a 64-byte space.

Addressing Problem 3: Fine-Grained On-Demand Allocation for Improved Efficiency

Another source of inefficiency in the naive design is that it allocates a large number of ECP entries for each assignment of a GEC entry. While this amortizes the tag overhead, it results in severe inefficiency, as most of the allocated ECP entries remain unused. The utilization of ECP entries can be increased by reducing the number of ECP entries in each GEC entry. For example, if each GEC entry contained only ECP-1, it would result in significant increase in utilization of ECP entries, even if it would mean relative increase in tag overhead. With ECP-1 in GEC entry, we can fit approximately 24 entries in the space of 64 bytes, therefore the associativity of GEC (SAT as well as GCT) would be 24. As there can be multiple tag hits in a given GEC set, we use the same precedence rule as used in the ECP proposal, that is, GEC entries are allocated from right to left, and younger entries have precedence over older entries. Our design restricts that all GEC entries of a given line must be placed in the same set. If a line needs more GEC entries and that set is full, then all ECP entries of the line are invalidated from the GEC set and relocated into a new set in GCT.

Proposed PAYG: Tying It All Together

PAYG obtains both high storage efficiency and low latency by leveraging the flexible structure for GEC, a hybrid LEC-GEC organization, and fine-grained allocation. Figure 5 shows the overall architecture of our proposed PAYG design.

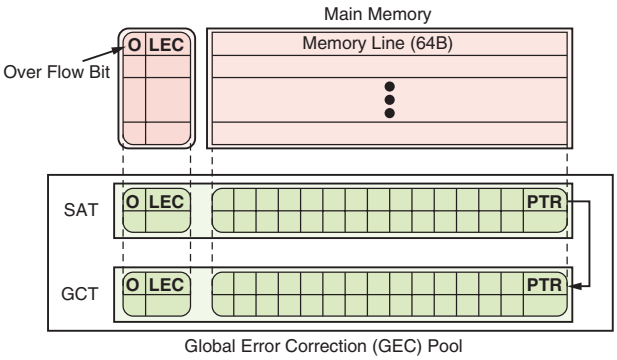

Figure 5: Proposed Architecture of PAYG
(Source: Moinuddin K. Qureshi, 2013)

"The LEC handles the common case of one-or-zero errors in a line for more than 95 percent of the lines. The GEC provides a storage-efficient low-latency on-demand allocation of ECP entries…"

The LEC handles the common case of one-or-zero errors in a line for more than 95 percent of the lines. The GEC provides a storage-efficient low-latency on-demand allocation of ECP entries for lines that have more than one error. Each GEC entry would contain only ECP-1 for high utilization of ECP entries. To reduce the array design overhead, we assume the same memory array for GEC (SAT and GCT) as the main memory, and provision the LEC + OFB for GEC as well, to maintain uniformity (this also allows the GEC size to be changed freely at runtime by the OS). An access to main memory with OFB = 0 is satisfied by single access. When OFB = 1, the GEC is accessed, one or more memory lines are read, matching GEC entries are obtained, ECP information is retrieved, and the line or lines get corrected.

Unlike uniform ECP-6, PAYG does not have to limit the maximum error correction allocated to a line. Thus, a weak line can use as many ECP entries as needed (limited only by the number of GE entries per line). This allows PAYG to outperform even ECP-6. The only real limiter of lifetime with PAYG is the number of GEC entries, as the likelihood of 24 or more errors per line is negligible for our system.

Results and Analysis

Our proposed design has three key components: the scalable structure for the GEC pool, Fine Grained Allocation (FGA), and Local Error Correction (LEC). In this section, we present the key results highlighting the importance of each of these components. We then analyze the storage and latency overheads, and also the impact of different variability scenarios on the effectiveness of our proposal.

Importance of Scalable GEC Pool

The key component of PAYG that provides scalability and efficiency is the architecture of the GEC pool. The first set of results we present are to emphasize the need for such a scalable structure. For this analysis, we assume a version of PAYG that has LEC implemented as ECP-1. The GEC does not have fine-grained allocation, which means each GEC entry contains ECP-5, and each set of GEC (in both SAT and GCT) contains 8 GEC entries. We call this configuration PAYG-NoFGA. Figure 6 compares the normalized lifetime of uniform ECP to that with PAYG-NoFGA. The left sets of bars are for ECP where the level of ECP is varied from 1 to 6. The middle sets of bars are for PAYG-NoFGA without GCT, where the number of sets in SAT is varied from 32K to 1024K. The right sets of bars are for PAYG-NoFGA with 128K sets in SAT and GCT sets vary from 2K to 64K.

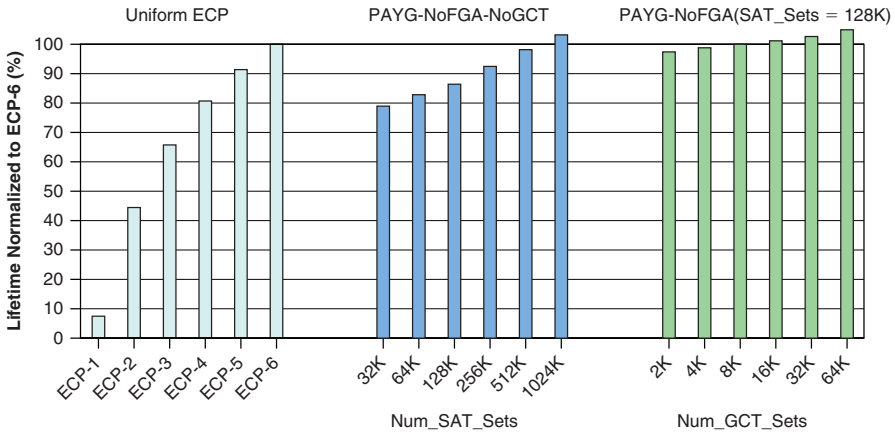

Figure 6: Lifetime of uniform ECP and PAYGNoFGA. Without GCT, PAYG-NoFGA needs 1024 sets (6.25% storage overhead) for lifetime comparable to ECP-6. With GCT, this reduces to (128K + 64K = 192K), 5x lower
(Source: Moinuddin K. Qureshi, 2013)

The first observation is that ECP-6 improves lifetime compared to ECP-1 by more than 10x. Unfortunately, ECP-6 incurs a storage overhead of 12 percent of memory capacity. The second observation is that PAYG-NoFGA needs a large number of sets (1 million) to achieve the lifetime as ECP-6, resulting in significantly high storage overhead (6.25 percent). However, the presence of GCT decreases storage requirement significantly. Combining 128K sets in SAT with 64K sets in GCT can provide a lifetime slightly higher than ECP-6 (this occurs because PAYG does not cap maximum error correction entries to six per line, so a few lines end up using ECP-7). The storage overhead of this combination would be 128K + 64K = 192K sets (1.2 percent overhead), which is 5x lower. Thus, a SAT-GCT based architecture is much more storage efficient than a simple set-associative structure. Unless specified otherwise, we will use 128K-set SAT combined with 64K-set GCT for the rest of the article.

"…SAT-GCT based architecture is much more storage efficient than a simple set-associative structure."

Importance of Fine-Grained Allocation

PAYG-NoFGA allocates five ECP entries with each GEC entry, most of which remain unused. FGA improves the utilization of ECP entries by reducing the number of ECP entries in each GEC entry. Table 3 shows the number of GEC entries that can be packed in one set (64 bytes), when the number of ECP entries in each GEC entry is varied from one to five. The tag size for our GEC structures is 7 bits, and we replicate the valid bit in GEC entry for fault tolerance. We also reserve 32 bits for GCTPTR (16 bits, 2-way replicated), which means only 480 bits per line are available for GEC entries. As the number of ECP entries per GEC entry decreases, the total number of GEC entries per each set increases.

Number of ECP in each GEC entry	1	2	3	4	5
Number of tag bits + valid bits	9	9	9	9	9
Number of bits for ECP	11	21	31	41	51
Size of 1 GEC entry (bits)	20	30	40	50	60
Number of GEC entries per set	24	16	12	9	8
Number of ECP entries per set	24	32	36	36	40

Table 3: Tradeoff between the number of ECP entries per GEC entry vs. ECP entries per set. Note that 24 GEC entries can be packed in one GEC set if each GEC entry contains ECP-1
(Source: Moinuddin K. Qureshi, 2013)

Figure 7 shows the normalized lifetime of PAYG as the number of ECP entries in GEC is varied from six to one. PAYG is implemented with LEC of ECP-1, SAT contains 128K sets, and GCT contains 64K sets. As the number of ECP entries in each GEC entry is reduced, there is a gradual increase in relative lifetime indicating that the effective utilization of ECP entries outweighs the relative increase in tag-store overhead.

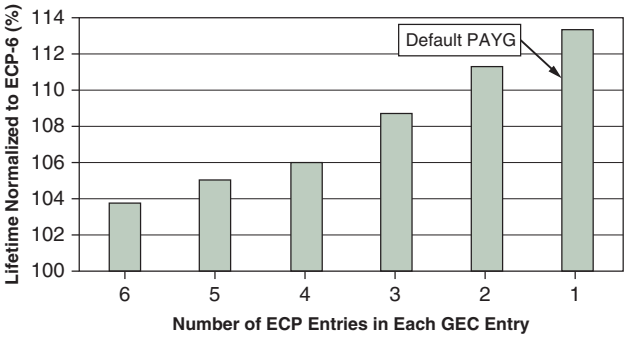

Figure 7: Effect of fine-grained allocation on effectiveness of PAYG. Note that having ECP-1 in GEC provides the highest lifetime and is the default PAYG configuration
(Source: Moinuddin K. Qureshi, 2013)

With only ECP-1 in each GEC entry, PAYG obtains a lifetime 13 percent higher than ECP-6, which is similar to that obtained with uniform ECP-8. Given the

efficiency of such fine-grained allocation, we assume that PAYG is implemented with ECP-1 in each GEC entry. The Default PAYG configuration used in our study is: 128K sets in SAT, 64K sets in GCT, LEC with ECP-1, and FGA with ECP-1 in each GEC entry. This configuration incurs a storage overhead of 3.8 percent of memory capacity and provides 13 percent more lifetime than uniform ECP-6.

"This configuration incurs a storage overhead of 3.8 percent of memory capacity and provides 13 percent more lifetime than uniform ECP-6."

Importance of Local Error Correction

The LEC provides the first line of defense for error correction in PAYG and is designed to handle the common case of zero or one failure per line. Figure 8 shows the normalized lifetime with PAYG as the level of ECP in LEC is varied from zero to six. Note that each ECP in LEC accounts for storage of approximately 2 percent of overall memory capacity, so having higher levels of ECP in each LEC entry incurs significant storage overhead. As expected, the lifetime increases with increasing ECP in LEC. A version of PAYG that has LEC containing ECP-5 has storage similar to uniform ECP-6 and provides a lifetime improvement of 43 percent. Thus, PAYG can not only be used to obtain a given amount of lifetime for reduced storage but can also be used to enhance lifetime at a given storage budget.

For the PAYG configuration without LEC (NoLEC), the given number of GEC entries are insufficient to handle the error rate, hence it obtains a lifetime lower than ECP-6. This can be avoided by simply increasing the number of GEC entries. The right set of bars in Figure 8 shows the lifetime of PAYG without LEC, when the GEC entries are doubled or quadrupled. We observe that simply doubling the entries (storage overhead of 2.4 percent) has lifetime equivalent to ECP-6, and when we double the GEC entries further to overhead of 4.8 percent, this combination can provide a lifetime significantly higher than with uniform ECP. However, the key problem of the PAYG configuration without LEC is the increased access latency. Because the line of defense of LEC is absent, all lines that have even a single error will experience increased latency because of GEC accesses.

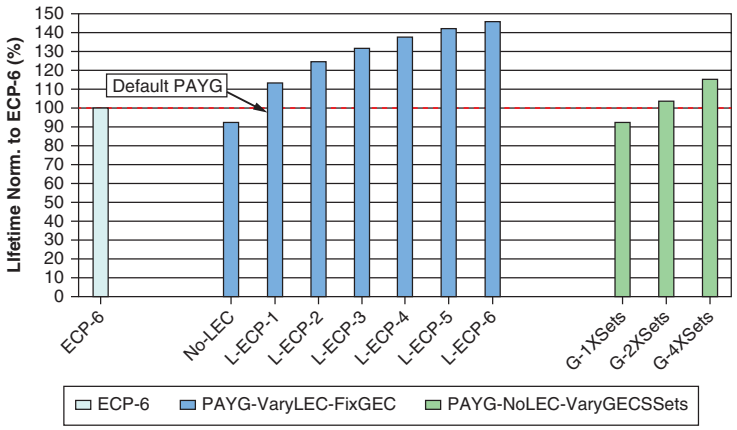

Figure 8: Lifetime impact of LEC: the middle set of bars vary ECP in each LEC entry from zero to six. To get lifetime comparable to ECP-6, we either need at least ECP-1 in LEC, or twice as many sets in GEC (Source: Moinuddin K. Qureshi, 2013)

"The storage overhead of PAYG is 3.13x lower than ECP-6."

Storage Overhead of PAYG

The storage overhead of PAYG consists of two parts: LEC and GEC. The overhead of LEC is incurred on a per-line basis, whereas the overhead of GEC gets amortized over all the lines. Table 4 computes the storage overhead of Default PAYG, given that the bank in our baseline contains $N = 2^{24}$ lines. The LEC incurs 13 bits/line (2-way replicated OFB bits + (1+10) bits for ECP-1). The storage overhead of PAYG is 3.13x lower than ECP-6. On average, PAYG needs 19.5 bits/line vs. 61 bits/line for ECP-6.

	PAYG
LEC (2 OFB + ECP-1)	13 bits/line
SAT (2^{17}) sets	2^{17} lines × 64 B = 8 MB
GCT (2^{16}) sets	2^{16} lines × 64 B = 4 MB
Total overhead of LEC	13 bits ($2^{24} + 2^{17} + 2^{16}$) = 26.9 MB
Total overhead of PAYG	26.9 MB + 8 MB + 4 MB = 38.9 MB
Total overhead of ECP-6	61 bits/line × 2^{24} = 122 MB
Ratio of (ECP-6/PAYG)	122 MB/38.9 MB = 3.13x

Table 4: Storage overhead of PAYG (PAYG obtains 13% more lifetime than ECP-6)
(Source: Moinuddin K. Qureshi, 2013)

Effective Latency with PAYG

Correcting an error with PAYG may require multiple accesses to memory. The main access simply gets broken down into multiple memory accesses (each of which takes deterministic time). The structures SAT and GEC are organized at a granularity of memory line, and we assume that an access to them incurs similar latency as access to main memory. When a GEC access occurs, the SAT is indexed and the memory line obtained is searched for a GECP entry with a matching tag. This incurs one extra memory access. If a match is not found, then the GCT is accessed, which incurs yet another memory access for each GCT access. However, this occurs rarely, given that GEC access happens only when the number of errors in a given line exceeds what can be corrected by the LEC. Figure 9 shows the percentage of demand accesses that require one extra access (satisfied by SAT) and two extra accesses (one for SAT and one for GCT). The probability of one extra access remains 5 percent or less throughout the expected lifetime under ECP-6 (6.5 years under continuous write traffic). Only after that does it increase significantly, reaching 17 percent at the end of lifetime with PAYG. In fact, for the first five years of system lifetime there is on average only 0.4 percent extra access per memory access, which means the performance impact is negligible (less than 0.4 percent) during the useful lifetime. The probability of two extra accesses remains very low throughout the lifetime.

"…for the first five years of system lifetime there is on average only 0.4 percent extra access per memory access…"

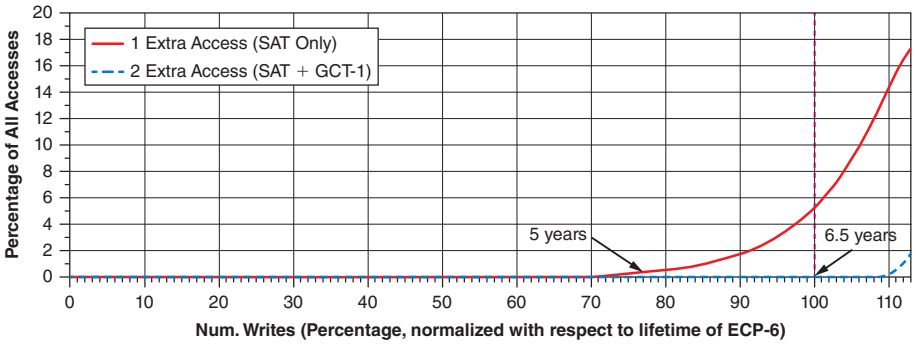

Figure 9: Extra accesses for each demand accesses with PAYG. Note that 100% of ECP lifetime is 6.5 years. PAYG incurs one extra access for less than 0.4% of memory accesses during the first five years of machine lifetime. The latency increases to noticeable range only after 6.5 years. The probability of three or more extra accesses as it remains negligible (< 0.01%) throughout the lifetime
(Source: Moinuddin K. Qureshi, 2013)

Summary

Emerging memory technologies suffer from the problem of limited write endurance. Such systems need high levels of error correction to ensure reasonable lifetime under high variability in device endurance. Uniformly allocating large amounts of error correction entries to all the lines results in most of them remaining unused. We can avoid the storage overhead of such unused entries by allocating the entries in proportion to the number of faults in the line. Based on this key insight, our article makes the following contributions:

- We propose Pay-As-You-Go (PAYG), an efficient hard-error–resilient architecture that allocates error correction entries on-demand, as and when errors occur.
- We propose a storage-efficient, low-latency organization for searching through large number of global error correction (GEC) entries.
- We reduce the latency for accessing error correction entries further by allocating a small amount of Local Error Correction (LEC) per line. Our analysis shows that one bit of LEC per line is sufficient to balance the tradeoff between storage overhead and latency impact.

PAYG can be implemented with any hard-error correction technique and is highly effective compared to line sparing. While we have evaluated the concept of nonuniform fault tolerance in the context of PCM systems, this concept is applicable to other memory technologies as well.

References

[1] Int'l Technology Roadmap for Semiconductors (ITRS). http://www.itrs.net/Links/2008ITRS/Home2008.htm.

[2] E. Ipek et al. "Dynamically replicated memory: building reliable systems from nanoscale resistive memories." ASPLOS-15, 2010.

[3] M. Qureshi et al. "Scalable high performance main memory system using phase-change memory technology." In ISCA-36, 2009.

[4] M. K. Qureshi et al. "Enhancing lifetime and security of pcm-based main memory with start-gap wear leveling." In MICRO-42, 2009.

[5] S. Schechter et al. "Use ECP, not ECC, for hard failures in resistive memories." In ISCA-2010.

[6] N. H. Seong et al. "SAFER: Stuck At Fault Error Recovery for Memories." In MICRO-2010.

[7] N. H. Seong et al. "Security refresh: Prevent malicious wear-out and increase durability for phase-change memory with dynamically randomized address mapping." In ISCA-37, 2010.

Author Biography

Moinuddin Qureshi is an Associate Professor in the School of Electrical and Computer Engineering at Georgia Institute of Technology. His research interests include computer architecture, scalable memory systems, fault-tolerant computing, and analytical modeling of computer systems. Prior to joining Georgia Tech, he was a research staff member at IBM T.J. Watson Research Center from 2007 to 2011. He was awarded the IBM outstanding technical achievement award for his studies on emerging memory technologies for server processors. He received his PhD (2007) and MS (2003), both in Electrical Engineering, from the University of Texas at Austin, and Bachelor of Electronics Engineering (2000) degree from University of Mumbai. He is a recipient of the NetApp Faculty Fellowship (2012) and Intel Early Career Faculty Award (2012). He can be reached at moin@ece.gatech.edu.