ECC Cache: A Lightweight Error Detection for Phase-Change Memory Stuck-At Faults

Chao Zhang, Khaled Abdelaal, Angel Chen, Xinhui Zhao, Wujie Wen, and Xiaochen Guo
Lehigh University, Pennsylvania, PA, 18015, USA
{chz616,kha217,anc520,xiz820,wuw219,xig515}@lehigh.edu

ABSTRACT

DRAM scaling has been slowed down. Emerging non-volatile memories (*e.g.*, Phase-Change Memory) promises higher density, better scalability, and persistence. However, endurance is a fundamental issue that hinders the broad adoption of PCM—after repeated writes, a PCM cell can get stuck at a value and be no longer programmable. The prevalence of this stuck-at fault issue requires error detection and correction mechanisms for PCM. Existing solutions such as verify-after-write adds additional latency to PCM writes, which degrades overall system performance. Other solutions like in-memory error-correcting code (ECC) requires a high storage overhead and introduces more reliability issue because ECC bits tend to wear out faster than the protected data bits.

In this paper, a novel stuck-at faults detection technique is proposed to improve performance and reliability simultaneously. Since stuck-at faults can only be detected after new writes, ECC does not need to be stored permanently and can be deleted immediately after a one-time detection, which helps to reduce ECC storage overhead. Therefore, this work proposes to use a small on-chip ECC cache to store the temporary ECC entries, which does not suffer from endurance issue. To maximize the utilization of the limited cache space, this work optimizes ECC entry insertion and deletion mechanisms and exploits memory bank-level parallelism to minimize performance impact. For the evaluated workloads, the proposed ECC cache achieves an average of 9.4% of the performance improvement over the baseline with a verify-after-write detection.

CCS CONCEPTS

• **Hardware → Memory and dense storage**.

KEYWORDS

PCM, Stuck-at-Fault, Error Detection, ECC, Verify-after-Write

ACM Reference Format:
Chao Zhang, Khaled Abdelaal, Angel Chen, Xinhui Zhao, Wujie Wen, and Xiaochen Guo. 2020. ECC Cache: A Lightweight Error Detection for Phase-Change Memory Stuck-At Faults. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD '20), November 2–5, 2020, Virtual Event, USA.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3400302.3415650

1 INTRODUCTION

As the scaling of DRAM technology to smaller feature sizes becomes increasingly difficult [19], PCM holds the potential to complement DRAM as a new memory device that blurs the boundary between the storage and the working memory or even replace DRAM as the main memory. However, PCM suffers from limited endurance. That is, after a limited number of writes to a memory cell (on average 10^6- 10^8) [13, 43], the PCM cell can get stuck at either '0' or '1'.

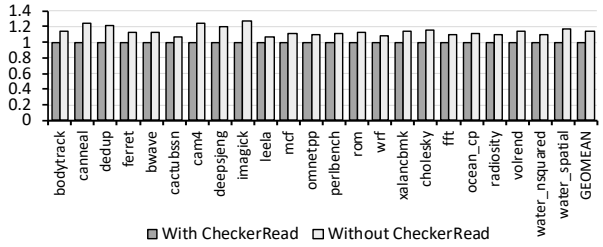

Figure 1: Performance impact of checker reads. Experimental setup is described in Section 4.

Existing fault tolerance techniques for PCM [17, 29, 33, 36, 42] focus mainly on providing error correction for stuck-at faults, while error detection has received very little attention. One common error detection mechanism is called *verify-after-write* [14], which simply issues an additional read (*checker read*) after each write. The checker read reads from the memory array after write finishes and the returned value is compared with the value intended to be written. An error is detected if the two values do not match. However, the drawback of this technique lies in the significant performance overhead incurred by checker read, as checker read can increase memory contention and block demand reads even when demand reads are prioritized over both writes and checker reads. As Fig.1 shows, verify-after-write can degrade the performance by 14.2% on average (detailed system configuration can be found in in Table 1).

Another option to detect PCM stuck-at faults is to use error-correcting code (ECC). Using in-memory ECC to detect error, no additional checker reads will be added after writes. However, there are two reasons why in-memory ECC is not commonly adopted for detecting and correcting stuck-at faults. 1) Using in-memory ECC for both hard- and soft errors can reduce the effectiveness of the ECC code protecting against soft errors when hard error rate increases [33, 42]. And 2) recent study [11] shows that ECC redundancy bits tend to have an average of 2× of bit flip rate as compared to data bits. For applications such as libquantum [6], ECC bits flip rate is 23.98× higher than it is in the data bits. Therefore, the unmatched bit flip rates make dedicated ECC storage cells wear out much faster than the data storage cells they are protecting.

The goal of the proposed design is to support PCM stuck-at-faults detection without performance impact from verify-after-write and endurance issue from in-memory ECC. The proposed mechanism is designed based on the following two observations. 1) It is unnecessary to detect stuck-at faults after every memory read. This is because stuck-at faults can be detected only after new writes. Detecting once after each write would be sufficient. 2) ECC redundancy bits do not have to be stored permanently in dedicated storage. The ECC redundancy bits can be deleted after the first read accomplishes the detection of stuck-at faults of the last write. Based on these insights, this paper proposes a new lightweight stuck-at-fault detection mechanism that uses an ECC cache to store temporary ECC redundancy bits for stuck-at fault detection. To the best of our knowledge, this is the first work addressing the

ICCAD '20, November 2–5, 2020, Virtual Event, USA

ACM ISBN 978-1-4503-8026-3/20/11…$15.00
https://doi.org/10.1145/3400302.3415650

reliability and performance trade-off for stuck-at fault detection. The key features and novelties of the purposed ECC cache are listed as follows:

- ECC cache has a low performance overhead and only issues checker read when it is not blocking demand reads.
- ECC cache does not introduce new endurance issue because it uses on-chip SRAM to store the temporary ECC bits.
- ECC cache deletes ECC entries after one-time detection, which are proactively issued when not interfering with demand reads.
- ECC cache is orthogonal and complementary to existing works [13, 17, 29, 33, 36, 42, 43] on error correction for PCM.

2 BACKGROUND

This section summarises PCM reliability issues and existing works on fault tolerance for PCM.

2.1 Hard Errors in PCM

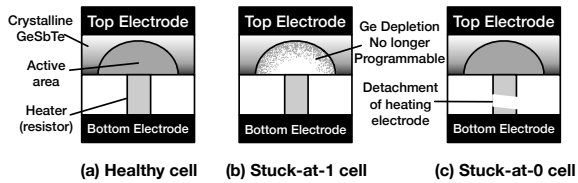

Figure 2: PCM cell and stuck-at-fault.

A PCM storage cell is a sandwich of a phase-change material (chalcogenide, typically GeSbTe) between two electrodes (Figure 2 (a)). The resistance of the cell depends on the phase of the chalcogenide material. After initial heating above the melting point, passing a high current through the heater for a relatively long time will cool down the cell slowly and hence crystallize the chalcogenide material and sets the cell into a low resistance state, which represents a logical '1'. If a short pulse is applied after the initial heating, the cell is cooled down rapidly, which results in a logical '0' state. A PCM cell can be programmed into partially amorphous and partially crystalline as well, which can be used to implement multi-level cell (MLC) that defines multiple resistance ranges for storing multiple bits in one cell. This work focuses on binary PCM cell (SLC).

The main source of hard errors is stuck-at fault in PCM, as the cell is at a certain permanent state. A PCM cell can either stuck-at high or low resistance after repeated writes. A stuck-at-1 fault (Figure 2 (b)) can be attributed to the overheating of the GST material during programming. When the generated heat is extremely high during programming, the chalcogenide material might intermix with the adjacent material and thus destroying its physical characteristics [26]. In such circumstances, Ge depletion is no longer programmable and the cell will continuously exhibit low resistance. Hence, the PCM cell will permanently have a 1 value. A stuck-at-0 fault can happen after the heating element is exposed to different amounts of electric current repeatedly. The heating element can be detached from the phase-change material after frequent expansions and contractions of the heating element, which leaves an open circuit (Figure 2 (c)). The stuck-at fault issue is a fundamental challenge for PCM. PCM capacity decays faster when used as the main memory, where it experiences more frequent writes.

2.2 Soft Errors in PCM

A PCM cell as a resistive device is immune to alpha particle-induced soft errors. But other factors can lead to soft errors in PCM. Resistance drift is identified as the major source of PCM soft errors. The reason is the meta-stable nature of the amorphous phase. Sudden cooling of the PCM cell can trigger phase-change. The resistance of the cell continues to drift high for a certain time, which is a resistance drift high error. At the same time, low crystallization of the phase change material at room temperature degrades the cell resistance over time. This type of long-term resistance drifts (typically over several days) can be easily addressed by periodically refreshing the resistance of the cells. Short-term drift can be problematic in MLC PCM, which is not the focus of this paper. For SLC PCM, resistance drift is not a major reliability concern [2, 25].

2.3 Hard Error Detection in PCM

A commonly adopted hard error detection mechanism in PCM is verify-after-write [17, 29, 33, 36, 42]. The basic idea is to issue an additional read (checker read) after each writes. If the value read from the PCM array matches with the value intended to be written, the write was successful and no hard errors were detected. This kind of checker reads introduces performance overhead. Note that checker read is more expensive than normal column read because it requires reading from the array, which is essentially an *activation* (reading from the array to row buffer) followed by a *column read* (reading from the row buffer). To ensure every checker read reads from the PCM memory array, an *auto-precharge* is used after every writes for both the baseline and the proposed architecture. An open-page after reads and close-page after write paging policy is used (see Section 4). The direct reason for the performance degradation is that it can block demand reads especially for write-intensive applications. There are two ways to implement the checker read: *integrated* and *decoupled*. The integrated checker read adds latency directly to its corresponding write and a combo memory command is issued to perform both the write and the checker read. In this case, the likelihood of encountering and blocking a demand read is higher. The decoupled checker read is implemented by inserting the check read to the end of the read queue in the memory controller to prioritize demand reads. This would add pressure to the read queue. Moreover, each write value has to be temporarily stored in the memory controller after issuing the write to wait for the comparison when the corresponding checker read value returns. The increased read queue pressure can block demand reads from entering the queue and hence degrades performance as well.

ECC is widely used to detect and correct errors in DRAM. Dedicated storage (typically 12.5% in ECC-DIMM) is required to store ECC redundancy bits. ECC syndrome is calculated to check if the value of any bits in the codeword (including the redundancy bits) has been changed. A commonly used ECC for protecting DRAM from soft errors is a single-error correction double-error detection (SECDED) code. Recent Study [44] shows the SLC PCM hard errors are orders of magnitude higher than the soft errors. Using in-memory ECC to cover both soft and hard errors requires much stronger ECC than protecting soft error only, which leads to a higher chip area overhead. Another reason that makes ECC codes in PCM inefficient is that the ECC bits tend to have a higher bit flip rate, which is costly in PCM with a limited write lifetime (ECC is a checksum to data bits). Y. Du *et al.* [11] claims that ECC bits in a single error correction double error detection (SECDED) code have nearly twice bit flip rate as compared to it is in the data bits. Therefore, PCM cells dedicated for storing ECC bits will be worn out twice as fast as the protected PCM cells storing data. After the ECC chip has worn out, the PCM will lose stuck-at-faults detection.

Both of these two reasons make in-memory ECC impractical for PCM stuck-at-fault protection.

2.4 Hard Error Correction in PCM

To mitigate the PCM lifetime limitation, some of the prior work proposed wear-leveling to spread the writes across memory cells uniformly [31, 40]. This scheme can limit the write frequency to the same subset of cells with a balanced distribution of the write traffic, and hence allows the memory capacity to gradually decay. However, the endurance limit of chalcogenide material for a PCM cell still exists, and wear-leveling alone can not solve this problem.

A common approach to mitigate the impact of the stuck-at faults is to remap the faulty memory location to a healthy cell location. The finer the granularity of this remapping, the slower the memory capacity decays. For example, error correction pointer (ECP) [33] can remap each faulty bit and correct up to 6 bits in a 64B line. Pointers can be used for fast access to an entire page as well, for which healthy cells within a faulty page might get retired early. ECP requires a 61-bit storage overhead for each 64B block. Compared to ECP, PAYG [29] provides a stronger error correction scheme, which uses a hierarchical sets of ECPs to allow sharing to support variations in the number of failures. DRM [17] and Zombie [3] pairs the faulty pages and recycles pages with stuck-at faults to form healthy pages. Other methods like SAFER [36], RDIS [24], and Aegis [13] are proposed to provide partition and inversion mechanisms for reusing the stuck-at faults in each data block based on the fact that writing a '1' to a stuck-at-1 cell is not an error and vice versa for a '0'.

3 KEY IDEAS AND OVERVIEW

This section presents the key ideas and an overview of the proposed ECC cache design.

Use checker read to reduce storage overhead, when writes are not blocking demand reads. The main drawback of the verify-after-write approach is the potential to block demand reads. However, not all checker reads block demand reads. For those that do not block demand reads, they can be issued immediately after the corresponding writes. When the read queue is empty or when the checker read is going to a bank that has no pending demand reads, the checker read can be added to the read queue. A detailed discussion about when to issue a checker reads immediately after writes is in Section 5.3. Write cancellation and write pausing [30] is also useful to prioritize the demand reads over writes and checker reads. In the proposed ECC cache, a new decoupled checker read is always inserted, and a checker read and write can be treated in two iterations, thus pending read can be served at the end of each write iteration.

Use temporary ECC to detect and correct stuck-at faults when writes are blocking demand reads. The proposed work uses a serially concatenated ECC, which can detect stuck-at faults in both data bits and in-memory ECC bits. The in-memory ECC is used for only soft error. As shown in Figure 3, a demand write will first check whether issuing a checker read will block demand reads. If it would not block demand reads, a checker read will be added to the end of the read queue. For the checker read that may block demand reads, the ECC cache will try to allocate an entry to store the ECC bits of the corresponding write value. However, the on-chip ECC cache has limited space. If the cache set is full, a checker read has to be inserted into the read queue. Every demand memory access checks the ECC cache. On a read hit, the demand read can serve as a free checker read and the returned value will be used in combination with the stored ECC bits to calculate syndrome. The ECC entry can be deleted after this read hit. Since these checker reads compare syndrome rather than values, this type of read is named as *deletion read*. On a write hit, the old ECC entry can be deleted as the new write can use the new ECC to detect stuck-at faults. This work chooses to use a hardware ECC cache instead of in-memory because a small hardware cache has shorter access latency, lower energy consumption, and no endurance issue.

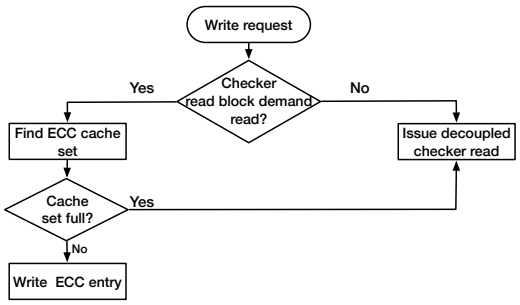

Figure 3: A flowchart of when to issue checker reads.

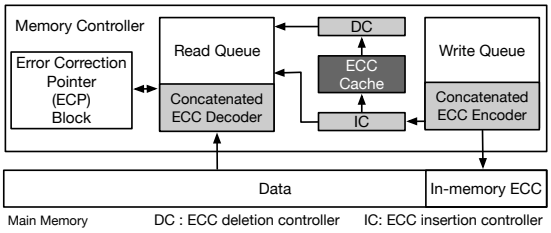

Figure 4: The proposed architecture with ECC cache.

Delete ECC entries proactively when not hurting performance. The limited size of the on-chip ECC cache can reduce the effectiveness of the proposed stuck-at fault detection scheme. Incoming writes must issue checker reads regardless of the performance overhead if the ECC cache is full. (Section 5.1). To make better use of the ECC cache, a proactive deletion scheme is required. The deletion rate managed by the deletion controller in Figure 4 should match the insertion rate such that the ECC cache can allocate space for writes when needed. Deleting an ECC entry requires issuing an extra deletion read and calculating the syndrome using the returned read value and the ECC bits. Therefore, deleting ECC entry needs to be careful not to interfere with demand reads. The proposed architecture leverages bank-level parallelism to delete ECC (Section 5.4).

Use existing techniques for remapping faulty bits after stuck-at faults are detected. The proposed failure detection mechanism is orthogonal to existing error correction mechanisms [27, 34] that remap faulty bits to healthy memory cells. In Figure 4, ECP is used for illustration without loss of generality. The proposed stuck-at fault detection can be combined with any existing stuck-at fault correction and recovery technique. Upon detecting an error, the in-cache ECC entry is used for the first-time correction. The ECC entry can be deleted after correction because the faulty bit will be remapped to a healthy cell for future uses.

As shown in Figure 4, a small ECC cache (64KB) is added into the memory controller, which is used for storing ECC entries for stuck-at fault detection and correction. Each of the cache entry is an 8B ECC code. An insertion controller (IC) determines whether to add a checker read into the read queue or to add an ECC entry into the ECC cache. Unlike conventional caches that replace cache entries when the cache set is full, the proposed ECC cache uses a deletion controller (DC) to determine when and which ECC entries to delete to minimize performance impact (Section 5.4). Section 6.5 will discuss which ECC code to use to satisfy a typical reliability requirement. This work assumes an SRAM-based ECC cache in the evaluation. PCM can be used for persistence applications. When

crush consistency is required, the ECC cache can be made persistent by using a battery to backup the ECC cache upon power failure similar to other prior work [1, 8, 45]. The ECC cache can also be implemented using non-volatile memory technologies such as embedded STT-MRAM [21] to support persistence. However, supporting persistence is not the focus of this work.

4 EXPERIMENTAL SETUP

The proposed ECC cache design choices are made according to the insights from experiments. Before presenting the details of the design, this section first describes the experimental setup.

4.1 Simulation infrastructure

This work uses ZSim [32] in combination with NVMain 2.0 [28] to perform execution-driven and cycle accurate simulation. CACTI 7.0 [4] is used for the ECC cache area, latency, and power modeling. Cadence Genus Synthesis Solution [7] is used to analyze the insertion controller and deletion controller of the ECC cache. McPAT [22] is used to estimate the entire on-chip energy consumption. The area is estimated based on FreePDK45nm [38] standard cell library, and is scaled to 22 nm by SPECTRE circuit simulations, which are the same as [15]. This work uses the Bose-Chaudhuri-Hocquenghem (BCH) code [23]. For BCH encoder and decoder, we adopt a parallel implementation of encoder from [18] and decoder from [39], which is similar in prior work [16].

Table 1: Baseline Configurations.

CPU	4 OoO cores, 2.6 GHz 32KB, 8-way private L1d and L1i cache, 128KB, 8-way private L2 cache 2MB, 16-way shared last-level cache (LLC)
Memory	4 GB PCM, DDR3 compatible 400MHz 1 channel, 1 rank per channel, 4 banks per rank tRCD/tWR/tRP = 99, 150, 70ns
Memory Controller	Scheduling policy: FR-FCFS_WQF Page policy: open-read/close-write page policy Write/Read queue size: 32/32 entries Write high/low threshold: 85%/50%

An open-page after read and close-page after write policy is used to optimize for applications with good spatial locality. The close-page after write policy uses auto-precharge, which ensures that checker reads can always read from the memory array instead of hitting in the row buffer. PCM requires a large write driver, hence the write throughput is typically limited to one column write per bank [12]. The close-page after write policy can also limit the write throughput for a realistic PCM device. An FR-FCFS_WQF scheduling policy is used with a dedicated write queue. A write queue draining mechanism is adopted to prioritize reads. The system will only switch to writes when the write queue occupancy is greater than a high threshold. And it will switch back to checker read when 1) write queue is empty or 2) read queue is not empty and write queue occupancy is below a low threshold. The PCM timing constraints come from Samsung PRAM with diode-switch cell array prototypes [9] and the adaptive precharge scheme proposed by J. Ko *et al.* [20]. A statistical simulation is performed to estimate the reliability of the proposed design (Section 6.5).

Table 2: Applications.

Suite	Type	Benchmark (write throughput in MB/s)
PARSEC 3.0	Multi-thread	bodytrack(8.34), canneal(12.14), dedup(8.35), ferret(9.19)
SPEC 2017 FP	Multi-program	bwave(12.04), cactubssn(8.43), cam4(13.08) imagick(9.83), rom(12.40), wrf(6.62)
SPEC 2017 INT	Multi-program	deepsjeng(12.00), leela(4.94), mcf(2.01) omnetpp(6.58), perlbench(8.26), xalancbmk(11.13)
SPLASH-2	Multi-thread	cholesky(7.42), fft(7.69), ocean_cp(6.12), radiosity(13.3) volrend(7.97), water_nsquared(3.02), water_spatial(7.37)

4.2 Benchmarks

A total of 22 applications are selected from PARSEC 3.0 [5], SPEC 2017 [10], and SPLASH 2 [41] with no less than 2 MB/s of average write throughput as listed in Table 2. The peak bandwidth of the PCM chip is 40MB/s. The first 100 million instructions are used to warm-up the on-chip caches. In ZSim, a dynamic binary translation is used to perform per-process fast-forwarding [32], which toggles per 100 million instructions.

5 THE PROPOSED ECC CACHE

This section first describes the challenge of storing temporary ECC bits in a small ECC cache. Section 5.2 demonstrates how this design overcomes the challenge of the limited cache space by utilizing bank-level parallelism. Section 5.3 and 5.4 discussed the proposed dynamic insertion and deletion schemes.

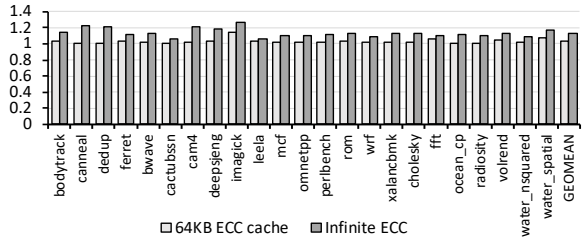

Figure 5: Finite ECC cache vs. infinite ECC cache (Speedup over no ECC cache).

5.1 Challenge of Finite ECC Cache Capacity

The proposed on-chip ECC cache is fast and consumes low energy, which is beneficial for storing temporary ECC entries. To understand the effectiveness of the proposed scheme, an ECC cache with infinite capacity could set the upper bound. The infinite ECC cache can achieve an average of 13.4% of the speedup as compared to the baseline with a verify-after-write detection, close to the ideal scenario, which does not use any checker read. The 0.8% performance difference (infinite ECC cache vs. no checker read) is due to the blocking of demand reads after a checker read is issued.

For a 64KB and 16-way set-associative ECC cache, less than 2% of speedup can be achieved because of its limited size. For write-intensive applications, the insertion rate is much higher than the hit deletion rate. The ECC cache is quickly filled up. To store more temporary ECC entries and reduce the performance impact, the number of ECC set-full cases need to be reduced. There are different ways to prolong the effectiveness of the ECC cache. Increasing the capacity of the cache is the most straightforward but also the most expensive way. Another approach is to virtualize the ECC cache by having a backup space in memory. But additional memory accesses might be needed on ECC cache misses and increases memory traffic. This work uses dynamic insertion (Section 5.3) and proactive deletion (Section 5.4) to effectively reduce the number of set-full cases when a write needs to insert an ECC entry.

5.2 Opportunity of No Pending Bank

Checker read or deletion read would have a greater performance impact if they block demand reads by going to a different row in the same bank. This work utilizes bank-level parallelism to issue checker reads and deletion reads. We define *No Pending bank (NP bank)* as the memory bank that has no pending read requests waiting in the memory controller queue.

To verify how many NP banks can be utilized, a breakdown of execution time is shown in Figure 6 based on the configuration in

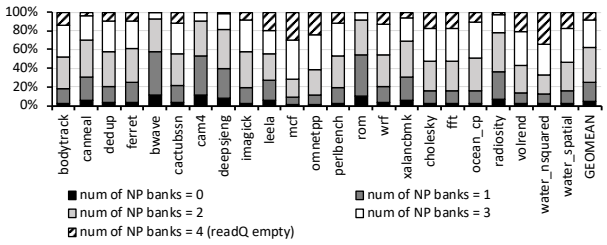

Figure 6: Execution time breakdown according to the number of NP banks.

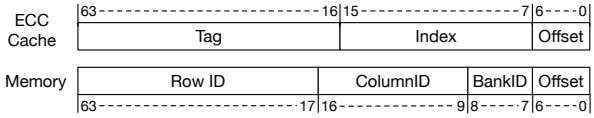

Figure 7: Address Mapping in ECC cache and Memory.

Table 1. On average, more than half of the banks are in NP state even with a block-interleaved address mapping (using LSB of the address for bankID, see Figure 7). Issuing checker reads to NP banks has a relatively low impact on demand reads because precharge and activation of different banks can be done in parallel. Exploiting bank-level parallelism will benefit both dynamic insertion and proactive deletions for the proposed ECC cache.

To monitor which banks are in the NP state, each bank has a counter in the memory controller to keep track of the total number of pending requests in the read queue (Figure 8). A bank can be identified as NP state when the corresponding counter is zero.

5.3 Inserting into the ECC cache

Using a static threshold of the read queue occupancy as an indicator for insertion is useful to reduce the total number of insertions. The higher the threshold, the lower the insertion rate. However, a high threshold also reduces performance because a greater percentage of writes will do a checker read than an ECC cache insertion, which can have a higher possibility to block demand reads.

Instead of setting a static threshold, the proposed design adopts a dynamic insertion policy, which adds a checker read to read queue when the target bank is currently a NP bank. This insertion method is named as *No pending insertion* in this paper.

For the proposed insertion scheme, the decision on not to insert to ECC cache is made under the following conditions, which requires adding a checker read to the read queue. First, a checker read is added if the read queue is empty. Second, a checker read is added if it is going to an NP bank, which can be easily known with the help of an NP bank state monitor. If neither of the above conditions is true, an ECC cache lookup is performed. On a cache hit, the ECC bits are updated and no immediate checker read will be issued. On a cache miss, a new ECC entry is inserted into the ECC cache if the cache set is not full. When the cache set is full, a checker read is added to read queue.

5.4 Deleting from the ECC cache

To reduce the set-full cases, deleting from ECC cache, and saving space for future writes is important.

5.4.1 Hit Deletion. A simple deletion policy would be to delete an entry on each demand access hit. Hit deletion is the cheapest deletion because no extra requests are generated. Directly calculating syndrome based on the data read from a demand read can serve the purpose of a checker read for the last write to the same location. For example, if a demand read to address A hits the ECC cache, there is an earlier write to address A pending to be calculated. The memory controller can delete the ECC entry after calculating the syndrome using both the returned read value and the ECC bits in the ECC cache. However, less than 5% of ECC entries can be deleted through hit deletion based on our observation in the evaluated benchmarks (Figure 12). The deletion rate can be lower than the insertion rate with hit deletion only. Most of the writes will encounter set full condition and have to add a checker read. More proactive deletion methods are required to increase the deletion rate.

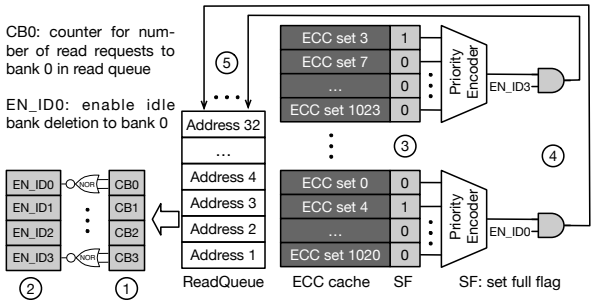

Figure 8: No pending bank deletion.

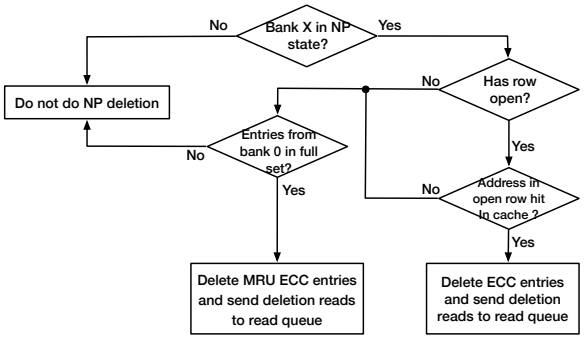

Figure 9: No pending bank deletion with row buffer hit optimization.

5.4.2 No Pending Deletion. NP bank deletion is a relatively cheaper mechanism to do proactive deletion since sending deletion read to an NP bank has a lower performance impact. Those deletion reads can be served in parallel with demand reads by utilizing bank-level parallelism. In this case, there are lots of opportunities to delete ECC entries from NP banks and those checker read latency will be hidden through bank-level parallelism.

As shown in Figure 8, ① CB0 is the counter that counts pending requests in the read queue. The value of the counter increments or decrements whenever a request arrives or departs. ② EN_ID0 is a one bit enable signal for bank 0's NP bank deletion. EN_ID0 is flipped to 1 when the CB0 is equal to zero, which allows an NP bank deletion read to be added if there is an ECC cache entry protecting a block in bank 0. When there are multiple choices, a deletion is selected according to the set full conditions. The proposed ECC cache uses bankID as part of the cache index, so the entries in the same cache set are mapped to the same memory bank. ③ Each set in ECC cache has a set full bit to indicate whether the set is full. Only full sets will be considered for NP bank deletion to make space for future ECC cache insertions. ④ If any ECC cache set is full and EN_ID0 is also true, the MRU (most recent used) entry from the full set will be deleted by sending a deletion read to the read queue. If there are N banks in total, N priority encoders are needed. ⑤ Now that the deletion read to bank0 is added into the read queue.

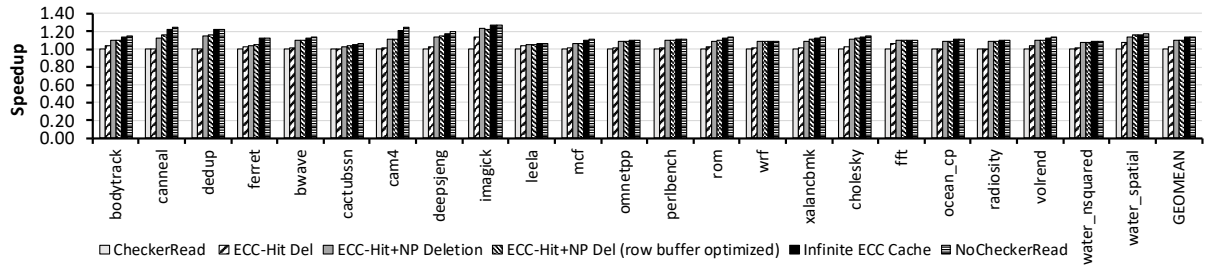

Figure 10: Normalized performance comparison.

Bank0 is no longer an NP bank until this deletion read is served. The EN_ID0 will be flipped back to 0 to block more NP bank deletions from bank 0 to the read queue. Because the demand reads to bank 0 arrive after a deletion read is added, the performance impact will be small.

5.4.3 No Pending Deletion with row buffer hit optimization. A deletion read to the NP bank can be served in parallel with demand reads. The total amount of deletion reads not only depends on the time window but also depends on the latency of each deletion read. The shorter the latency for each deletion read, the more ECC entries that can be deleted.

One optimization for the NP bank deletion is to find deletion reads that can hit in the row buffers. Accessing data stored in the row buffer is faster and costs less energy. A bank in NP state may still have a row open as long as it is not precharged. Sending deletion reads to the open row can further increase the deletion rate. The flowchart of an NP bank deletion with a row buffer hit optimization scheme is shown in Figure 9.

If there is any bank identified as an NP state and has row open, a deletion read going to the open row will have the highest priority to the read queue. Then, the address in the open row will be searched by using the same row, bank, rank, and channel ID. Similar as what is shown in Figure 8 ③, ECC entry deletion going to the open row will be read out by a priority decoder. At last, the entry will be deleted and a corresponding deletion read is added to the read queue for stuck-at faults detection.

6 EVALUATION

This section presents the evaluation of performance, energy cost, hardware overhead, and the reliability of the proposed ECC cache.

6.1 Performance

To understand the performance of the ECC cache, speedup contribution, insertion breakdown, deletion breakdown, and the lifetime of the ECC entries are discussed.

Speedup Contribution. Figure 10 shows the speedup of adding each of the following deletions features one at a time: 1) hit deletion, 2) NP bank deletion, and 3) row buffer hit optimization. All of the limited capacity ECC cache configurations use the same insertion policy described in Section 5.3.

The baseline is a design with verify-after-write. After adding a 64KB, 16 set way-associative ECC cache with hit deletion, the performance can be improved by an average of 2.8%. This is because hit deletion by itself is not sufficient to make space for future writes. Only a few demand accesses will hit in the ECC cache, hence the ECC cache is quickly filled up. As a result, most of the attempted insertions to the ECC cache fail due to set full. For the evaluated benchmarks, 98.8% of checker reads are added due to the ECC cache set full. This also proves that a proactive deletion is necessary to achieve a high deletion rate. After adding normal NP deletion, the performance on an average can be improved by 8.8%, which is much higher than hit deletion only. This is because the entries in the ECC cache can be deleted by sending a deletion read to NP banks, which utilizes the bank-level parallelism to minimize the negative performance impact on demand reads. After adding NP bank deletion, only 4.9% of the writes would send a checker read because of set full conditions, which means almost all of the writes can be successfully inserted into an ECC cache entry. Moreover, optimizing the latency for NP deletion with row buffer hit can have an average of 9.5% of speedup over the baseline and only 1.37% of writes will add checker reads due to set full (Figure 12).

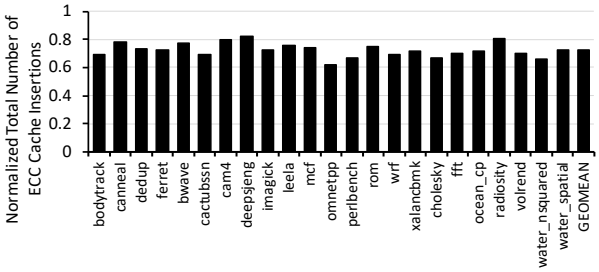

Figure 11: Insertion reduction after NP detection.

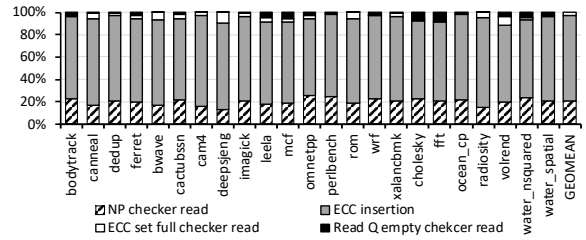

Figure 12: Insertion Breakdown.

Insertion Breakdown. As described in Section 5.3, there are four possibilities for each demand write: 1) read queue is empty and adds a checker read to read queue. 2) checker read will go to an NP bank and hence add a checker read (read queue not empty). 3) add a checker read because the ECC cache set is full, and 4) insert an ECC entry into the ECC cache and do not need a checker read but might require a deletion read in the future. For the read queue empty case, the checker read can only block demand reads if demand reads arrive after checker read is issued. The performance impact of this case is the lowest. However, for the selected write-intensive applications, less than 1.2% of the checker reads will be added into the read queue when it is empty. Issuing checker reads to NP banks helps to reduce the total number of insertions to the ECC cache without blocking demand reads. This reduces the total number of ECC cache entry insertions by an average of 27% (Figure 11). This optimization can reduce the ECC cache energy with nearly zero impact on system performance.

An average of 20.49% of the writes adds checker read to NP banks, which has a very low negative performance impact. ECC set full case

is a condition that could have the most negative performance impact because checker reads must be added regardless of whether they will block demand reads. After using NP Bank deletion with row buffer hit optimization, only 1.37% of the writes add checker reads due to the set full condition. Given the fact that ECC cache is small and set-associative (64KB, 16-way associativity), the low percentage of set full condition shows that the proposed deletion scheme is effective and can match with the insertion rate. Last but not least, more than 76% of the writes issue ECC cache insertions rather than checker reads, effectively preventing demand read blocks.

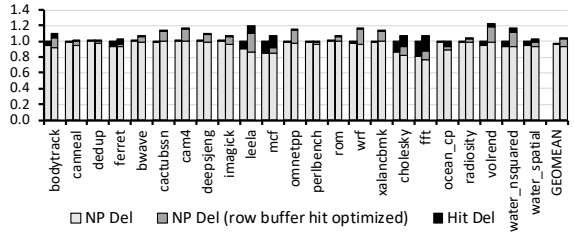

Figure 13: Deletion Breakdown (normalized). Left bars: without row buffer hit optimization; right bars: with row buffer hit optimization.

Deletion Breakdown. An ECC entry can be deleted from ECC cache in three ways: 1) when a demand read hit the ECC cache the demand read can serve as a deletion read; 2) NP bank deletion that detects NP bank state, deletes an entry from a full set and adds a deletion read; and 3) NP bank deletion that sends a deletion read to an open row. Deletion breakdowns for ECC cache with and without row buffer hit optimization are presented in Figure 13. For the tests without row buffer hit optimization, hit deletion takes an average of 1.03% of the total deletions. For applications like mcf, cholesky, and fft, nearly 15% of the deletions are from hit deletion, still not effective enough to make space for future insertions. More than 98% of the deletion are from NP bank deletion, which is why the performance can be significantly improved by adding NP bank deletion. For the tests with row buffer hit optimization, an average of 8.7% of the total deletions are row buffer hit deletions. The number of other NP bank deletions stays almost the same and the total number of deletions is increased by 7.2%. Given the fact that a bank does not stay in the NP state for too long, reducing the latency of deletion read through row buffer hit optimization can increase the total number of deletions.

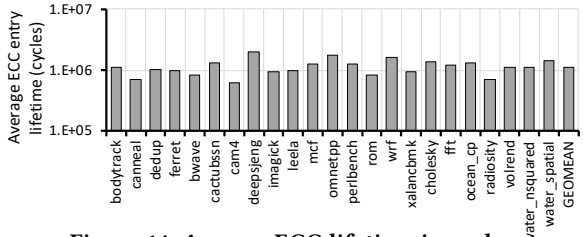

Figure 14: Average ECC lifetime in cycles.

ECC Entry Lifetime. A key difference between the proposed ECC cache-based stuck-at fault detection scheme and a conventional verify-after-write scheme is that writing ECC entries delays the detection. Therefore the system detects the stuck-at faults later than the conventional design. The lifetime of an ECC entry is the cycle count from its insertion to its deletion. As shown in Figure 14, most of the applications have an average of ECC entry lifetime around a million cycles, which is less than 1ms under a 2.6 GHz frequency, which does not influence reliability significantly (Section 6.5). Write requests are not on the critical path, and deferred write with write queue draining already delays write requests. In fact, the lifetime of the ECC entries shows the buffering effect for stuck-at fault detection to allow flexible scheduling. Note that using ECC cache to delay stuck-at fault detection is more storage efficient as compared to verify-after-write. This is because verify-after-write needs to buffer the entire block, whereas ECC bits are much smaller (typically 12.5%) than the block data.

6.2 Energy Consumption

The ECC cache is not performance-critical. Sequentially accessing tag and data array can help to reduce the energy consumption from tag lookup. Adding a dedicated ECC cache for stuck-at faults detection will lead to higher power consumption. We break down energy consumption into five parts: 1) cache lookup dynamic energy for every demand access, 2) NP deletion cache lookup dynamic energy, 3) ECC cache insertion dynamic energy, 4) insertion and deletion controller dynamic energy, 5) leakage energy of both the ECC cache and modified memory controller logic.

To analyze the additional energy overheads, an energy breakdown is shown in Figure 15. On average, nearly half of additional energy overhead is from leakage energy, which is expected for a deeply scaled technology node. Demand access lookup, NP deletion lookup, ECC cache insertion, and insertion/deletion controller consume an average of 19.9%, 10.73%, 19.46%, and 4.54% of the total additional energy respectively.

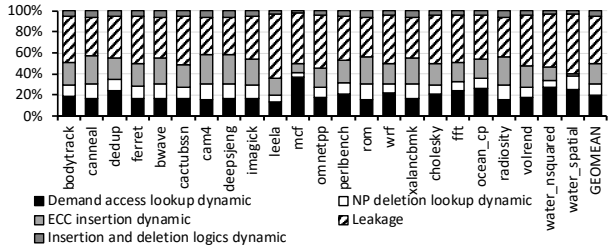

Figure 15: Energy overhead breakdown.

As compared to the total on-chip energy consumption, the ECC cache total energy overhead is negligible (ECC cache is at memory controller). This has two reasons: 1) the total amount of the cache lookup depends on the number of memory reads and writes, which is much lower than the number of on-chip cache accesses (less than 1% of L1 dcache accesses). Since the on-chip caches have a larger capacity, they can filter out most of the off-chip traffics, which also means relatively fewer accesses to the ECC cache. 2) The ECC cache is much smaller as compared to the LLC and therefore the leakage power is relatively small as well. The optimized insertion scheme can also effectively reduce ECC cache energy. As a result, the proposed ECC cache-based stuck-at fault detection scheme has low energy overhead.

Table 3: Area Breakdown of the ECC Cache Design.

ECC Logic	ECC Cache	Insertion and Deletion Controller	Total Area	
0.410	0.107	0.005	0.522	(mm^2)

6.3 Area Overhead

The proposed architecture adds the following hardware structures to the memory controller: 1) a 64KB ECC cache, 2) insertion and deletion controller to handle ECC entries in cache, and 3) a set of 6EC7ED BCH ECC encoder and decoder. The total area overhead is 0.522 mm^2, which occupies 1.13% of the processor die (46.19 mm^2).

Note that prior work like Free-p [42] also suggests using BCH code for error detection and correction, which requires similar overhead for BCH encoder and decoder.

6.4 ECC Cache Size

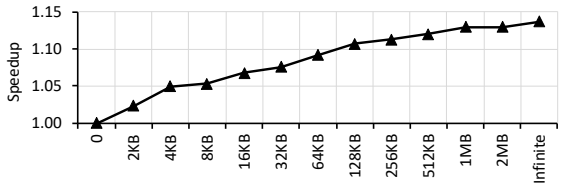

Figure 16: Speedup for different ECC cache sizes. Set-associativity is 16 for all of the limited size configurations.

Increasing the cache capacity can reduce the set full checker reads by providing more space for insertions. A study on different ECC cache sizes (2KB-2MB) is presented in Figure 16. An infinite ECC cache has a speedup of 13.4% over the baseline with verify-after-read. ECC cache capacity over 64KB can achieve better performance but has a diminishing return. A 4.8% speedup can still be achieved with a small 4KB 16-way-set associative ECC cache. Increasing ECC cache capacity can easily reduce set full cases that may block demand reads. However, adding a large ECC cache is expensive for limited space on the processor die. According to the sensitivity study the optimum size of proposed ECC cache would be 16 - 128 KB to achieve the best trading offs between performance and area overhead.

6.5 Reliability

Memory failure simulation is challenging because it is infeasible to simulate the real operations of memory over a full lifetime. For the stuck-at fault detection, soft errors can contribute to false positive, which means a soft error can be identified as a hard error because the detection method cannot distinguish among different types of errors. To reduce the false positive, writes can be performed again to confirm whether the detected error is due to stuck-at faults. The performance overhead is expected to be low if the failure rate is very low. The limited endurance is the root cause of stuck-at faults. If storage cell endurance variation follows a normal distribution and the writes are evenly distributed, the stuck-at fault induced bit error rate would increase as the device ages. As the device ages the reliability gets worse and stuck-at faults will dominate the bit error rate. Hence, this work builds a failure model focusing on stuck-at faults by first assuming the storage cell endurance follows a normal distribution with a mean of 10^8 write operations, and a 0.25 coefficient of variance (δ in equation. 1) is used similar to many prior work [3, 17, 33, 42]. With more than 50% of capacity decay after nine years [33, 42], the failure model only includes the first nine years result. The probability that a random cell has a stuck-at faults error at the t^{th} write is

$$P(t) = \int_{-oo}^{t} \frac{1}{\sqrt{2\pi}\delta} e^{\frac{(t/\text{lifetime}-1)^2}{2\delta^2}} \quad (1)$$

, which is a cumulative normal distribution function. To estimate the uncorrectable error rate, a binomial test is performed as shown in equation. 2

$$P(X = k) = \binom{n}{k} p^k (1-p)^{n-k} \quad (2)$$

, in which n is the length of the codeword, k is the number of errors, p is the error probability from equation 1. Therefore, the uncorrectable error probability for SECDED is when the received data have more than one errors (P(X>1)).

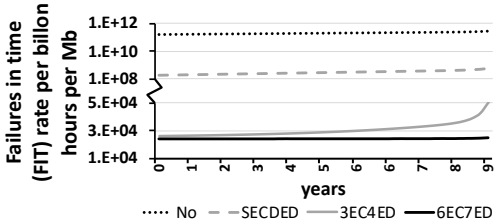

Figure 17: Uncorrectable FIT for different ECC codes in ECC cache. Endurance in number of writes is converted into time assuming a perfect wear-leveling and the write throughput is 20%[1] of the PCM peak bandwidth (40Mb/sec per chip).

The failure in time (FIT) is a standard metric to measure the reliability of a device. FIT refers to the total number of failures in one billion device hours. Recent studies on DRAM soft error rate show an average of 25,000 - 75,000 FIT per Mbit [35, 37]. Given the fact that PCM is robust against particle induced soft errors and negligible resistance drift errors, the soft error rate of SLC PCM is expected to be lower than the DRAM. In the proposed work, a small static soft error rate is added into the failure model, *i.e.,* 25,000 FIT per Mbit. Unlike ECC, the checker read can detect and correct any number of errors within a block. Therefore, the proposed ECC cache can have additional uncorrectable errors because ECC can detect only limited number of errors. The stronger ECC used by ECC cache, the higher reliability the system can achieve.

As shown in Figure 17, the reliability impact of SECDED hamming code, 3EC4ED BCH code, and 6EC7ED BCH code are evaluated. The FIT rate for SECDED starts from over 10^8. 3EC4ED maintains in a safe range within the first 3-4 years, but starts increase rapidly after 5 years. 6EC7ED can maintain a stable FIT rate for nine years. As compared to soft error rate (25,000), using 6EC7ED in the proposed ECC cache increase the FIT by 0.004%. Since most of the server machine retire in eight years or more, a 6EC7ED BCH code should detect the stuck-at faults with relatively high reliability.

7 CONCLUSION

In this paper, a novel lightweight PCM stuck-at fault detection scheme is proposed, which leverages temporary ECC to reduce the performance overhead of error detection without additional in-memory ECC storage. The proposed architecture combines verify-after-write with an ECC cache. Checker reads are used after writes when they are not blocking demand reads, whereas ECC bits are calculated and stored temporarily in a small ECC cache at the memory controller. To better utilize the limited ECC cache space, dynamic insertion and deletion schemes are proposed to minimize the negative performance impact of error detection. As a result, the proposed ECC cache can achieve similar performance with a system that does not issue verify-after-write while adding small area and energy overhead.

ACKNOWLEDGMENT

This material is based upon work supported by the National Science Foundation at Lehigh University under Grant CCF-1750826, CCF-1723624, and CCF-2006748. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

[1]This is close to what it is in the evaluated workloads.

REFERENCES

[1] Amro Awad, Pratyusa Manadhata, Stuart Haber, Yan Solihin, and William Horne. 2016. Silent Shredder: Zero-Cost Shredding for Secure Non-Volatile Main Memory Controllers. , 14 pages. https://doi.org/10.1145/2872362.2872377

[2] Manu Awasthi, Manjunath Shevgoor, Kshitij Sudan, Bipin Rajendran, Rajeev Balasubramonian, and Viii Srinivasan. 2012. Efficient scrub mechanisms for error-prone emerging memories. , 12 pages.

[3] Rodolfo Azevedo, John D Davis, Karin Strauss, Parikshit Gopalan, Mark Manasse, and Sergey Yekhanin. 2013. Zombie memory: Extending memory lifetime by reviving dead blocks. , 452–463 pages.

[4] Rajeev Balasubramonian, Andrew B Kahng, Naveen Muralimanohar, Ali Shafiee, and Vaishnav Srinivas. 2017. CACTI 7: New tools for interconnect exploration in innovative off-chip memories. *ACM Transactions on Architecture and Code Optimization (TACO)* 14, 2 (2017), 14.

[5] Christian Bienia. 2011. *Benchmarking Modern Multiprocessors*. Ph.D. Dissertation. Princeton University.

[6] B Butscher and H Weimer. 2013. libquantum. The C library for quantum computing and quantum simulation.

[7] Cadense. 2019. Cadence Genus Synthesis Solution. https://www.cadence.com/content/dam/cadence-www/global/en_US/documents/tools/digital-design-signoff/genus-synthesis-solution-ds.pdf.

[8] Siddhartha Chhabra and Yan Solihin. 2011. i-NVMM: a secure non-volatile main memory system with incremental encryption. , 177–188 pages.

[9] Youngdon Choi, Ickhyun Song, Mu-Hui Park, Hoeju Chung, Sanghoan Chang, Beakhyoung Cho, Jinyoung Kim, Younghoon Oh, Duckmin Kwon, Jung Sunwoo, et al. 2012. A 20nm 1.8 V 8Gb PRAM with 40MB/s program bandwidth. , 46–48 pages.

[10] SPEC CPU®. 2017. Standard Performance Evaluation Corporation. https://www.spec.org/cpu2017/.

[11] Yu Du, Miao Zhou, Bruce R Childers, Daniel Mossé, and Rami Melhem. 2013. Bit mapping for balanced PCM cell programming. *ACM SIGARCH Computer Architecture News* 41, 3 (2013), 428–439.

[12] Yu Du, Miao Zhou, Bruce R. Childers, Daniel Mossé, and Rami Melhem. 2013. Bit Mapping for Balanced PCM Cell Programming. , 12 pages. https://doi.org/10.1145/2485922.2485959

[13] Jie Fan, Song Jiang, Jiwu Shu, Youhui Zhang, and Weimin Zhen. 2013. Aegis: Partitioning data block for efficient recovery of stuck-at-faults in phase change memory. , 433–444 pages.

[14] Alexandre P Ferreira, Miao Zhou, Santiago Bock, Bruce Childers, Rami Melhem, and Daniel Mossé. 2010. Increasing PCM main memory lifetime. , 914–919 pages.

[15] Qing Guo, Xiaochen Guo, Ravi Patel, Engin Ipek, and Eby G. Friedman. 2013. AC-DIMM: Associative Computing with STT-MRAM. , 12 pages. https://doi.org/10.1145/2485922.2485939

[16] Xiaochen Guo, Mahdi Nazm Bojnordi, Qing Guo, and Engin Ipek. 2017. Sanitizer: Mitigating the impact of expensive ecc checks on stt-mram based main memories. *IEEE Trans. Comput.* 67, 6 (2017), 847–860.

[17] Engin Ipek, Jeremy Condit, Edmund B Nightingale, Doug Burger, and Thomas Moscibroda. 2010. Dynamically replicated memory: building reliable systems from nanoscale resistive memories. , 3–14 pages.

[18] Zhang Jun, Wang Zhi-Gong, Hu Qing-Sheng, and Xiao Jie. 2005. Optimized design for high-speed parallel BCH encoder. , 97–100 pages.

[19] Kinam Kim. 2005. Technology for sub-50nm DRAM and NAND flash manufacturing. , 323–326 pages.

[20] Junyoung Ko, Jisu Kim, Youngdon Choi, HK Park, and Seong-Ook Jung. 2015. Temperature-tracking sensing scheme with adaptive precharge and noise compensation scheme in PRAM. *IEEE Transactions on Circuits and Systems I: Regular Papers* 62, 8 (2015), 2091–2102.

[21] Kangho Lee and Seung H Kang. 2010. Development of embedded STT-MRAM for mobile system-on-chips. *IEEE Transactions on Magnetics* 47, 1 (2010), 131–136.

[22] Sheng Li, Jung Ho Ahn, Richard D Strong, Jay B Brockman, Dean M Tullsen, and Norman P Jouppi. 2009. McPAT: an integrated power, area, and timing modeling framework for multicore and manycore architectures. , 469–480 pages.

[23] Wei Liu and SUNG Won-Yong. 2012. Bose-Chaudhuri-Hocquenghem error correction method and circuit for checking error using error correction encoder. US Patent 8,122,328.

[24] Rami Melhem, Rakan Maddah, and Sangyeun Cho. 2012. RDIS: A recursively defined invertible set scheme to tolerate multiple stuck-at faults in resistive memory. , 12 pages.

[25] Sparsh Mittal. 2017. A survey of soft-error mitigation techniques for non-volatile memories. *Computers* 6, 1 (2017), 8.

[26] Mohammad Gh Mohammad. 2011. Fault model and test procedure for phase change memory. *IET computers & digital techniques* 5, 4 (2011), 263–270.

[27] Ravi H Motwani and Kiran Pangal. 2016. Use of error correction pointers to handle errors in memory. US Patent 9,250,990.

[28] Matthew Poremba, Tao Zhang, and Yuan Xie. 2015. Nvmain 2.0: A user-friendly memory simulator to model (non-) volatile memory systems. *IEEE Computer Architecture Letters* 14, 2 (2015), 140–143.

[29] Moinuddin K Qureshi. 2011. Pay-As-You-Go: low-overhead hard-error correction for phase change memories. , 318–328 pages.

[30] Moinuddin K Qureshi, Michele M Franceschini, and Luis A Lastras-Montano. 2010. Improving read performance of phase change memories via write cancellation and write pausing. , 11 pages.

[31] Moinuddin K Qureshi, John Karidis, Michele Franceschini, Vijayalakshmi Srinivasan, Luis Lastras, and Bulent Abali. 2009. Enhancing lifetime and security of PCM-based main memory with start-gap wear leveling. , 14–23 pages.

[32] Daniel Sanchez and Christos Kozyrakis. 2013. ZSim: Fast and accurate microarchitectural simulation of thousand-core systems. , 475–486 pages.

[33] Stuart Schechter, Gabriel H Loh, Karin Strauss, and Doug Burger. 2010. Use ECP, not ECC, for hard failures in resistive memories. , 141–152 pages.

[34] Stuart Schechter, Karin Strauss, Gabriel Loh, and Douglas C Burger. 2014. Error correcting pointers for non-volatile storage. US Patent 8,839,053.

[35] Bianca Schroeder, Eduardo Pinheiro, and Wolf-Dietrich Weber. 2009. DRAM errors in the wild: a large-scale field study. *ACM SIGMETRICS Performance Evaluation Review* 37, 1 (2009), 193–204.

[36] Nak Hee Seong, Dong Hyuk Woo, Vijayalakshmi Srinivasan, Jude A Rivers, and Hsien-Hsin S Lee. 2010. SAFER: Stuck-at-fault error recovery for memories. , 115–124 pages.

[37] Nak Hee Seong, Sungkap Yeo, and Hsien-Hsin S Lee. 2013. Tri-level-cell phase change memory: Toward an efficient and reliable memory system. , 440–451 pages.

[38] James E Stine, Ivan Castellanos, Michael Wood, Jeff Henson, Fred Love, W Rhett Davis, Paul D Franzon, Michael Bucher, Sunil Basavarajaiah, Julie Oh, et al. 2007. FreePDK: An open-source variation-aware design kit. , 173–174 pages.

[39] Dmitri Strukov. 2006. The area and latency tradeoffs of binary bit-parallel BCH decoders for prospective nanoelectronic memories. , 1183–1187 pages.

[40] Jue Wang, Xiangyu Dong, Yuan Xie, and Norman P Jouppi. 2013. i 2 WAP: Improving non-volatile cache lifetime by reducing inter-and intra-set write variations. , 234–245 pages.

[41] Steven Cameron Woo, Moriyoshi Ohara, Evan Torrie, Jaswinder Pal Singh, and Anoop Gupta. 1995. The SPLASH-2 programs: Characterization and methodological considerations. *ACM SIGARCH computer architecture news* 23, 2 (1995), 24–36.

[42] Doe Hyun Yoon, Naveen Muralimanohar, Jichuan Chang, Parthasarathy Ranganathan, Norman P Jouppi, and Mattan Erez. 2011. FREE-p: Protecting non-volatile memory against both hard and soft errors. , 466–477 pages.

[43] Jiangwei Zhang, Donald Kline Jr, Liang Fang, Rami Melhem, and Alex K Jones. 2017. Dynamic partitioning to mitigate stuck-at faults in emerging memories. , 651–658 pages.

[44] Zhe Zhang, Weijun Xiao, Nohhyun Park, and David J Lilja. 2012. Memory module-level testing and error behaviors for phase change memory. , 358–363 pages.

[45] Pengfei Zuo, Yu Hua, Ming Zhao, Wen Zhou, and Yuncheng Guo. 2018. Improving the performance and endurance of encrypted non-volatile main memory through deduplicating writes. , 442–454 pages.