2018 51st Annual IEEE/ACM International Symposium on Microarchitecture

Exploring and Optimizing Chipkill-correct for Persistent Memory Based on High-density NVRAMs

Da Zhang
Department of Computer Science
Virginia Tech
Blacksburg, USA
daz3@vt.edu

Vilas Sridharan
RAS Architecture
AMD
Boston, USA
vilas.sridharan@amd.com

Xun Jian
Department of Computer Science
Virginia Tech
Blacksburg, USA
xunj@vt.edu

***Abstract*—Emerging high-density non-volatile random access memories (NVRAMs) can significantly enhance server main memory by providing both higher memory density and fast persistent memory. An unique design requirement for server main memory is strong reliability because uncorrectable errors can cause a system crash or permanent data loss. Traditional dynamic random access memory (DRAM) subsystems have used chipkill-correct to provide this reliability, while storage systems provide similar protection using very long ECC words (VLEWs).**

This paper presents an efficient chipkill-correct scheme for persistent memory based on high-density NVRAMs. For efficiency, the scheme decouples error correction at boot time from error correction at runtime. At boot time, when bit error rates are higher, the scheme uses VLEWs to efficiently ensure reliable data survival for a week to a year without refresh by correcting a large number of bit errors at low storage cost. At runtime, when bit error rates are lower, it reuses each memory block's chip failure protection bits to opportunistically correct bit errors at high performance. The proposal incurs a total storage cost of 27%. Compared to a bit error correction scheme, the proposal adds chip failure protection at no additional storage cost and at 2% average performance overhead.

***Index Terms*—ECC, Microarchitecture, Persistent Memory Systems, Reliability**

I. Introduction

Emerging high-density non-volatile random access memories (NVRAMs), such as multi-level phase change memory (PCM) and resistive random access memory (ReRAM), provide both higher density than DRAM and fast persistent memory [1]–[13]. High memory density is attractive for server systems, whose memory needs have increased due to big data, in-memory computing, and server virtualization. Persistent memory can accelerate I/O-intensive server applications by providing fast access to non-volatile storage at memory block granularity instead of the page granularity provided by storage systems. As such, dense NVRAM-based memory may find wide adoption among future server memory systems.

A key design requirement for server memory systems is reliability because an uncorrectable error can cause a system crash resulting in costly downtime; large-scale surveys report service downtime costs of millions of dollars per hour [14], [15]. In the context of persistent memory, uncorrectable errors can also cause permanent data corruption because data stored in persistent memory may not be backed up in storage.

To ensure reliability, DRAM-based server memory systems implement chipkill-correct as a standard feature to protect against both bit errors and memory chip failures. Improving the efficiency of chipkill-correct is an active area of research [16]–[31]. Providing chipkill-correct for NVRAM-based memory systems is challenging, however, because dense NVRAMs have higher random raw bit error rates (RBER) than DRAM [32]–[38]. RBER is especially high after a long time without refresh (i.e., a week to a year), and the ability to reliably tolerate long intervals without refresh is essential for NVRAMs to serve as persistent memory. As demonstrated in Section III-B, using DRAM chipkill-correct to tolerate the high NVRAM RBER incurs expensive (e.g., 69%) storage costs.

Storage systems also need to tolerate both chip failures and high RBER. Storage systems incur low storage overheads by using very long ECC words (VLEWs) to correct errors. At a given error rate, longer ECC words provide equivalent reliability to shorter ECC words with less storage overhead [39]. Applying VLEWs to main memory is challenging, however, because the access granularity of main memory is much smaller than the size of VLEWs, as VLEWs protect storage systems, whose access granularity is much bigger than main memory. Therefore, protecting main memory with VLEWs incurs high read bandwidth overhead as each read request must over-fetch many memory blocks to check the ECC. Protecting main memory with VLEWs also incurs high write bandwidth overhead because updating code bits for a write request requires a read-modify-write operation when the codeword is bigger than the written block.

This paper explores efficient chipkill-correct for NVRAM-based persistent memory. To provide efficient chipkill-correct for dense NVRAM-based persistent memory, we decouple correction of NVRAM bit errors at boot time from correction of NVRAM bit errors at runtime. We use storage-optimized VLEWs where each ECC word spans tens of blocks to ensure, at minimum storage cost, reliable data survival between the

978-1-5386-6240-3/18/$31.00 ©2018 IEEE
DOI 10.1109/MICRO.2018.00063

last system outage and the next reboot; we use performance-optimized short ECC words, where each word spans a single block, to correct bit errors at runtime to minimize read bandwidth cost. Instead of using a dedicated short ECC to correct bit errors at runtime, our scheme reuses each block's chip failure protection bits to opportunistically correct bit errors at runtime to minimize storage cost for runtime. To reduce write bandwidth overhead for updating VLEW code bits, we observe that dirty persistent memory blocks occupy only a small fraction of on-chip cache capacity, on average, because applications that use persistent memory frequently clean dirty persistent memory blocks (i.e., memory blocks belonging to persistent memory regions) [1]–[6]. As such, we propose preserving old memory values of dirty persistent memory blocks in the last-level cache to reduce write bandwidth overhead at runtime; since the number of dirty persistent memory blocks in the cache is small, this incurs a small dynamic reduction in usable cache capacity. Compared to providing only bit error correction for persistent memory, the proposal adds chip failure protection while incurring no extra storage cost and only 2% average performance cost across many persistent memory applications.

II. Background

A. Memory organization and technologies

Server memory systems access a group or *rank* of memory chips in lockstep for each memory request. The amount of data transferred per request is called a *memory block*. Each block is typically 64B. Each memory chip typically contributes 8B[1] to the accessed block.

NVRAMs capable of DRAM-like latency, such as PCM, ReRAM, and STT-RAM, are emerging as viable memory technologies for future systems due to their higher storage density and non-volatility [41]–[44]. NVRAMs can provide higher density than DRAM for the same feature size because the material can often store multiple logical bits per cell and support a crossbar array architecture, which is $\sim 50\%$ density than 1T1C DRAM arrays [45] [44]. NVRAMs can provide non-volatility because the bit cells have much longer retention time than DRAM.

NVRAM subsystems will likely have similar chip structure and system organization as DRAM subsystems [41]–[44], [46]–[51]. Therefore, maximizing the reuse of existing infrastructure and standards facilitates NVRAMs' adoption. Many DRAM-like NVRAM chips and modules have been prototyped and manufactured [52]–[55]. 3DXPoint [56], a commercially available non-volatile memory for servers, also spreads each access across a group of chips like accesses to a rank in server memory [57]–[59].

B. Bit Errors in NVRAMs

Figure 1 shows the RBERs of 2-bit PCM, 3-bit PCM, and ReRAM reported in recent studies [34], [60]–[64]. For comparison, Figure 1 also includes the RBER of commercially available Flash devices and the cell fault rate of 28nm DRAM [29], [65], [66]. The RBERs of NVRAMs resembles Flash more than DRAM.

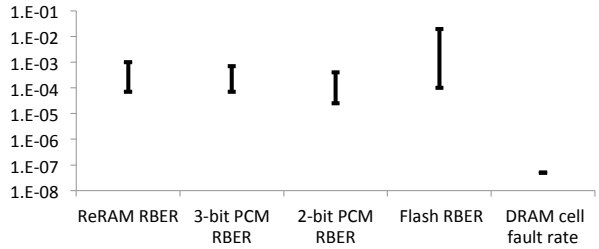

Fig. 1. RBERs of memory and storage.

The RBER of high-density NVRAMs increases with the amount of time since last write or refresh; this is the reason for the wide range of RBER in Figure 1. Similar behavior also exists in Flash; Cai et al. [66] report that the RBER of Flash cells three months after last write can be 100X higher than RBER one day after last write. In general, all memories (e.g., HDD, Flash, NVRAM, DRAM) forget data over time; the longer since last memory refresh, the more data memory loses. To ensure data survive across system outages, which can last a long time, persistent memory must tolerate the high RBER after a long time without refresh. We target a RBER of 10^{-3} for persistent memory; this corresponds to the RBER of ReRAM one year since last refresh [63] and the RBER of 3-bit PCMs one week after last refresh [60].

The raw bit errors in dense NVRAMs are predominantly stochastic, similar to DRAM soft errors [33], [34]. In particular, bit errors in multi-level PCMs are dominated by resistance drift and bit errors in ReRAM and STT-RAM are dominated by retention errors, both of which are random processes [34], [60], [61], [63], [64]. Wear errors in ReRAMs are also probabilistic; the probability that a given cell will be read erroneously rises gradually with the number of writes to that cell before eventually reaching 100% [64].

III. Problem

Due to the high RBER of dense NVRAMs, simply extending prior works on memory error correction to implement chipkill-correct for dense NVRAM-based persistent memory incurs prohibitive storage overheads. For the remainder of this paper, we assume a reliability target of less than one block with an uncorrectable error (UE) per 10^{15} blocks [60] and less than one block with silent data corruption (SDC) per 10^{17} blocks at any instant (e.g., boot time or runtime) during the memory system's lifetime. We use standard combinatorial error probability analysis throughout the paper, similar to [34]. For simplicity, our analytical model assumes that write requests do not provide any free error scrubbing effect, similar to [34].

A. Extending prior work on NVRAM bit error correction

Prior works have explored how to correct NVRAM random bit errors [33], [34], [67]. They protect each 64B memory

[1] Although current DDR4 X4 chips transmit only 4B data per access, DDR5 X4 chips will also transmit 8B data [40].

block with a multi-bit-correcting BCH code. For example, to tolerate PCM RBER up to 10 minutes after last refresh, Awashi et al. [33] propose protecting each block with a BCH code that can correct up to eight bits of errors (i.e., 8-bit-EC BCH); similarly, to tolerate the RBER of STT-RAMs five seconds after last refresh, Naeimi et al. [34] propose protecting each block with a 5-bit-EC BCH. One way to tolerate 10^{-3} RBER for NVRAM-based persistent memory is to protect each block with a stronger 14-bit-EC BCH. BCH requires $t(\lceil log_2(k) \rceil + 1)$ code bits to correct t bad bits when protecting k bits of data; protecting each 64B memory block with 14-bit-EC BCH incurs 28% storage overhead. 14-EC BCH provides adequate protection against 10^{-3} RBER.

Note that simply increasing the strength of BCH ECC to 14-bit ECC only provides bit error protection, but not chip failure protection. A single chip failure in the rank can cause up to 64 bits of errors in each block. Strengthening the short per-block BCH ECC to correct the 64 bits of errors due to chip failure requires increasing its strength to $64 + 14$ =78-bit error correction, which incurs a prohibitive 152% storage overhead.

Some prior works on addressing NVRAM RBER have proposed limiting memory density; NVRAM RBER is lower when memory density is lower. For example, Seong et al. [68] propose reducing PCM RBER by limiting the number of logical bits per cell to 1.5; this comes at a high cost of 100% capacity overhead compared to allowing 3 bits/cell. This approach is also not scalable. For example, commercially available QLC Flash cells has recently scaled up to four bits per cell; ideally, we want NVRAMs to benefit from similar scaling.

B. Extending prior works on DRAM chipkill-correct to persistent memory based on dense NVRAMs

The fault rate of DRAM cells may rise sharply to 10^{-4} in future generations of higher density DRAMs [28], [29], [31]. As such, several recent works have explored chipkill-correct for DRAMs with high cell fault rate [28], [29], [31]. Unlike bit errors in NVRAMs, which are random, bit errors in DRAMs are dominated by permanent cell faults, such as stuck-at-faults [28], [29], [31]. Weak ECCs can tolerate very high rates of permanent cell faults (e.g., even when 0.01% of all cells are permanently faulty) [28], [29], [31]; because permanently faulty cells can be identified at manufacturing time, memory chip or module manufactures can simply discard chips or even memory modules with patterns of permanent faulty cells that are uncorrectable by the weak ECCs at a small yield loss [28], [29], [31]. However, when the large set (e.g., millions) of erroneous bits keep changing over time, which is the case when bit errors are random, weak ECCs can no longer provide adequate protection. For example, DUO [31], the most recent work on chipkill-correct for DRAMs with high cell fault rates, only tolerates up to 10^{-10} random bit error rate[2].

[2]Random RBER of 10^{-10} is calculated from the worst-case random error modeling assumption in [31] where 10^{-5} of DRAM cells are intermittently faulty cells and each such cell has a 10^{-5} error activation probability.

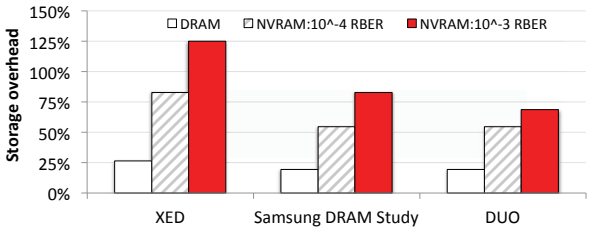

Fig. 2. Total storage cost when adapting DRAM chipkill-correct to implement chipkill-correct for dense NVRAM-based persistent memory.

A simple approach to extend prior works on DRAM chipkill-correct to protect dense NVRAM-based persistent memory is to add more code bits to each codeword to correct more bit errors. XED [28] and a Samsung study [29] protect every group of 8B and 16B of data within each chip, respectively, with a BCH ECC to correct bit errors in individual data chips and then use a parity chip to correct a faulty chip in the rank. One simple way to adapt these works to NVRAM-based persistent memory is to increase the number of code bits in each BCH ECC word. DUO [31] uses a Reed-Solomon (RS) code to protect each 64B block against both bit errors and chip failures. RS ECC corrects errors at the byte granularity, at the storage cost of two check bytes to correct each erroneous data byte; when the locations of the bad bytes are known, however, as is the case for errors due to a chip failure, RS ECC can correct each bad byte using just one check byte via erasure correction. A bad byte whose location is known is called an erasure. DUO uses one RS check byte to correct each chip-failure-induced erasure and uses two RS check bytes to correct each random bit error. DUO can be extended to NVRAMs again by simply increasing the strength of RS ECC to tolerate the higher RBER. However, the above simple extensions incur high storage overheads, as shown in Figure 2; the lowest storage cost for 10^{-3} RBER is 69%.

IV. Motivation and Challenges

We observe storage systems also need to tolerate both chip failures and high RBER. Storage systems commonly use strong ECC (e.g., 12 to 41 error correction [69], see Figure 3) to tolerate high RBER and use a parity disk/chip to correct chip/disk failures. Storage systems require very low redundancy, however; assuming eight data chips and one parity chip, the total storage overhead is only $13\%+1/8*(1+13\%)$=27% when protecting MLC Flash chips against bit errors using 41-bit-EC. Storage systems enjoy such low redundancy despite using strong (e.g., 41-bit-EC) codes by exploiting a well-known fact in coding theory that longer ECC words require less storage cost than shorter ECC words to ensure same reliability for the same RBER [39]. Storage systems have very large access granularity (i.e., 4KB) and, therefore, naturally benefit from VLEWs. Figure 3 shows BCH ECC words commonly used in commercial Flash chips; all ECC words in Figure 3 contain 512B of data.

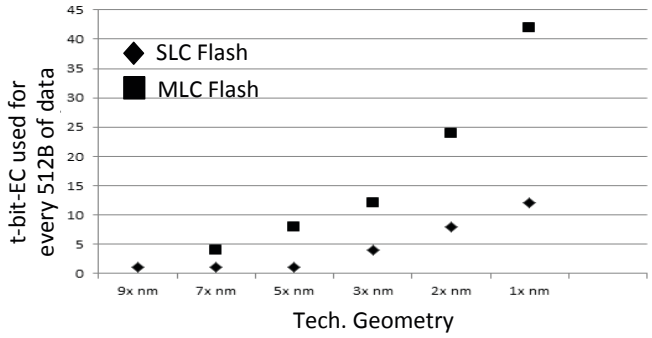

Fig. 3. Bit error correcting ECC in Flash [69]. Flash uses very strong ECCs (e.g., 41-bit-EC) and reduce storage cost of strong ECCs by using VLEWs each containing 512B of data.

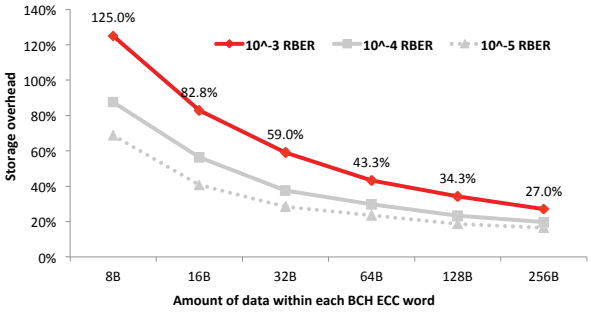

Fig. 4. Storage cost vs. codeword length.

Figure 4 shows the total storage cost when extending storage-inspired protection to NVRAMs by using VLEWs to correct bit errors and by using one parity chip for every data chips to correct a chip failure; Figure 4 shows BCH words of different lengths for comparison. When using VLEWs with 256B of data, total storage cost to tolerate both NVRAM bit errors and chip failure reduces to 27%, which matches the 28% storage cost of bit error protection alone (see Section III-A).

A. Read Challenges when Using VLEWs

While protecting persistent memory with VLEWs enables storage-optimized chipkill-correct, it comes at high memory bandwidth overhead. Because the access granularity of main memory is much smaller than that of storage systems, each VLEW spans many blocks when used in main memory. This is worsened by the fact that each VLEW only protects data within a single[3] chip; the 256B of data in a VLEW spans $256B/8B = 32$ memory blocks. For 10^{-3} RBER, each VLEW must correct up to 22 bad bits, which requires 33B of BCH code bits; as such, the code bits in each VLEW span $33B/8B \approx 4$ blocks. Using VLEWs to correct bit errors in one block requires fetching $32+4-1 = 35$ additional blocks; this translates to high bandwidth overheads, especially when RBER is high, which causes frequent error correction. While RBER is relatively lower at runtime when memory can be refreshed, it is still very high in absolute terms. For example, the RBER of ReRAM is $\sim 7 \cdot 10^{-5}$ [63] at runtime.

Under $7 \cdot 10^{-5}$ RBER, 4% of accesses still contain bit error(s); correcting bit errors for 4% of accesses incurs $4\% \cdot 35 = 140\%$ bandwidth overheads for read requests. For 3-bit PCM, RBER is also $7 \cdot 10^{-5}$ if refreshed once every second [60]. Unfortunately, refreshing NVRAMs requires correcting errors that have accumulated in NVRAMs; fetching all blocks to correct their errors once every second causes high (e.g., $\sim$1000%) memory bus bandwidth overhead even for small memory channels with small amount of NVRAMs (e.g., 160GB). For a more realistic refresh rate of once per hour, the RBER of 3-bit PCM increases to $2 \cdot 10^{-4}$ [60], which causes 10.3% of memory accesses to contain bit errors. Correcting bit errors for 10.3% of accesses incurs $10.3\% \cdot 35 = 360\%$ overall bandwidth overheads for reads.

One possible solution to mitigate the high bandwidth overheads is to perform VLEW error correction within NVRAM chips themselves. However, this is expensive because VLEWs are very long and strong (e.g., 22-bit-EC over 2048 bits of data). For example, Flash chips with embedded error correction suffer from either lower performance (e.g., 3X [70], [71]) or lower (e.g., 16X [72]) density compared to raw Flash chips, which rely on processor-side error correction. Flash chips with embedded correction logic also pay high (e.g., 66%) energy overheads [70] and increases cost per bit [73]. The high cost is because the memory manufacturing process is sub-optimal for implementing complex logic due to low transistor speed and low (e.g., three [74]) metal layer count while VLEW correction requires solving complex large systems of simultaneous equations [75].

B. Write Challenges when Using VLEWs

Protecting persistent memory with VLEWs also incurs high bandwidth overheads for writes because writing to memory requires updating the code bits protecting the modified data block. Because the amount of code bits in each VLEW is $33B/8B \approx 4X$ the access granularity of each chip, writing the new VLEW code bits of a written data block to memory requires four overhead write requests; this translates to 400% overheads for writes. The common approach of mitigating write bandwidth overhead via caching code bits can significantly complicate persistent memory design. If the system crashes between the write of persistent memory data or log and the write of their cached VLEW code bits, the written data and their stale VLEW code bits in persistent memory will be inconsistent with each other during system recovery and cause uncorrectable errors; these uncorrectable errors can cause irrecoverable persistent memory data corruption if they affect committed data that are beyond rollback. While co-designing persistent memory programming and VLEW code bits caching to ensure reliable survival after system crash may

[3]If each VLEW protects data across all chips in a rank, a chip failure can cause hundreds of bit errors in up to all VLEWs in the affected rank, rendering all VLEWs uncorrectable; correcting a faulty chip via the parity chip requires first correcting bit errors in working chips via the VLEWs.

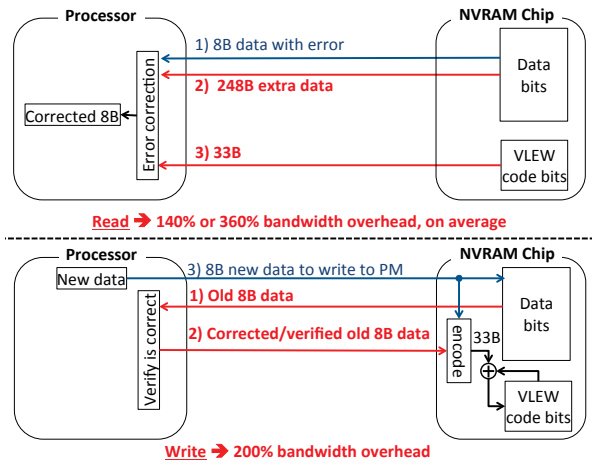

Fig. 5. Read (TOP) and write (BOTTOM) memory bandwidth overheads (shown in RED) when deploying VLEWs in persistent memory (PM).

be possible, it is complex because many memory blocks share the same VLEW code bits.

One possible solution to the write bandwidth overhead is to embed the encoder of VLEW code bits into NVRAM chips; when the new code bits are encoded in-memory, the processor no longer needs to perform any overhead write requests to VLEW code bits to update them after regular data write requests. Unlike error correction circuits, ECC encoding circuits incur very small overheads even for VLEWs. Because BCH ECC is a linear code, encoding simply calculates the right hand side values for a system of linear equations where all left hand side values are known [76]; this is far simpler than correction, which solves this very large system of equations with many unknown variables.

While the simple solution above eliminates overhead write requests to VLEW code bits, there still remains the high bandwidth overhead of accessing old data bits to compute the new VLEW code bits. Because a VLEW is bigger than a memory block, calculating the new VLEW code bits for a write request requires both old data bits and code bits as inputs. Specifically, $ECC_{new} = ECC_{old} \oplus ECC_{Update}$; $ECC_{Update} = f(x) \oplus f(x')$, where f is the ECC encoding function, x is the new data to be written, and x' is the old data to be overwritten, and $\oplus$ is bitwise XOR [76]. While x' and ECC_{old} reside in memory and thus can be fetched in memory without incurring traffic over the memory bus, x' and ECC_{old} can contain bit errors due to the high NVRAM RBER. Fortunately, bit errors in ECC_{old} are tolerable because they simply propagate one-to-one (i.e., without spreading) to ECC_{new} during bitwise XOR with ECC_{Update}; unfortunately, using a wrong x' to calculate $f(x')$ to obtain ECC_{new} can directly cause SDC. As such, the processor must fetch x' from memory for error detection/correction and send the corrected x' back[4]

[4]I/O transmission errors can still occur when writing the old data back to memory; however, modern memory chips use Write-CRC [77] to effectively detect these I/O errors and dynamically alert the processor to retransmit.

to NVRAM chips; this incurs an expensive 200% bandwidth overheads for write requests.

Figure 5 summarizes the error correction and write bandwidth overheads when protecting persistent memory with VLEWs. For clarity, Figure 5 shows only one NVRAM chip, since all chips in a rank are identical, and assumes the case where old data block happens to be error-free.

V. Efficient Chipkill-correct for Dense NVRAM-based Persistent Memory

NVRAM RBER is lower at runtime, when memory can be periodically refreshed, than at boot time when the NVRAM has potentially gone a long time without refresh. As such, a strong ECC is needed for boot time, but a weaker but faster per-block ECC suffices to correct bit errors that occur at runtime.

Therefore, to provide efficient chipkill-correct for dense NVRAM-based persistent memory, we decouple correction of NVRAM bit errors at boot time, when bit errors may have accumulated due to a long power outage, from correction of NVRAM bit errors at runtime. We propose using storage-optimized VLEWs to ensure reliable data survival at low storage cost and using performance-optimized short ECC words to correct bit errors at runtime. We also explore optimizations for both ECCs to address their respective drawbacks and achieve low storage and performance cost.

A. Data and ECC Layout

Figure 6 shows the layout of data bits and ECC bits under the proposed scheme. Within each chip, each group of 256B data in the same row is part of the same VLEW; because the access size of each chip is 8B, each VLEW spans $256/8 = 32$ blocks. Each VLEW's code bits are located in the same row as the VLEW's data bits by increasing the number of bits per row in each chip, similar to Flash chips [78], [79]. Each VLEW contains sufficient code bits to provide 22-bit error correction to ensure data can reliably survive the high RBER after a long time without refresh.

Each rank also contains a parity chip to tolerate a complete chip failure. Each 8B from the parity chip protects 64B of data from eight data chips. Data in the parity chip is encoded using a Reed-Solomon code instead of a parity code [76].

The total storage cost due to the VLEW code bits in each chip and the parity chip is $33/256+1/8*(1+33/256) = 27\%$.

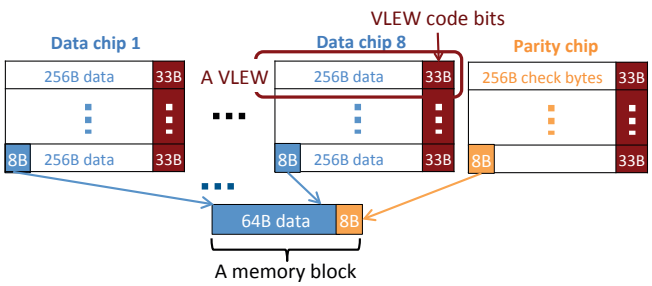

Fig. 6. Proposed rank-level data layout.

B. Error Correction at Boot Time

At boot, the memory controller fetches all VLEWs to scrub all bit errors that have accumulated in persistent memory. Assuming 3GHz memory bus frequency, scrubbing a large persistent memory system with a terabyte of memory per channel takes less than 1.5 minutes. If a VLEW in a data chip reports an uncorrectable error, a chip-level fault has occurred. The memory controller uses the parity chip to correct the faulty data chip. Through erasure correction, the eight Reed Solomon check bytes can correct up to eight bad bytes in the block to help correct a complete chip failure [76]. If the chip with an uncorrectable error is the parity chip, the memory controller recalculates the parity values in the parity chip using the contents of the data chips.

C. Error Correction at Runtime

During normal operation, the memory controller fetches 64B of data and the associated eight RS check bytes from the parity chip. The memory controller uses the RS check bytes to opportunistically correct bit errors in the data without needing to fetch the VLEW code bits from the parity chip.

The eight RS check bytes can correct up to four random bytes of errors. When a memory access contains five or more errors, the number of errors exceeds the ECC's correction capability and the ECC can miscorrect the errors, resulting in silent data corruption (SDC). $1.5 \cdot 10^{-7}$ of memory accesses contain five or more errors assuming an RBER of $2 \cdot 10^{-4}$ (see Figure 7); under $2 \cdot 10^{-4}$ RBER, using each block's RS ECC to correct all bit errors yields an SDC rate of $3.2 \cdot 10^{-11}$ (see Appendix), which is $3,000,000X$ higher than the SDC target rate of $1 \cdot 10^{-17}$. Assuming a lower RBER of $7 \cdot 10^{-5}$ still results in an SDC rate that is $18,000X$ higher than the SDC rate target.

To reliably correct random bit errors using per-block RS ECC, we observe that a miscorrection is more likely to appear as a large number of corrections (e.g., three or four bytes in error) than as a small number of corrections (e.g., one or two bytes in error). That is, a larger number of corrections indicates a higher probability of a miscorrection. As such, we set a threshold on the number of corrections. The threshold is less than the number of errors the per-block RS ECC can correct (i.e., four). We conservatively assume that accesses with a larger number of corrections than the threshold may

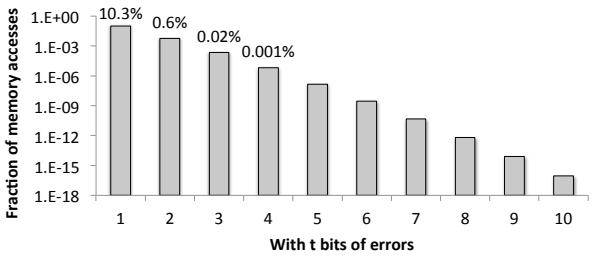

Fig. 7. Distribution of number of bit errors in 64B memory requests assuming $2 \cdot 10^{-4}$ RBER.

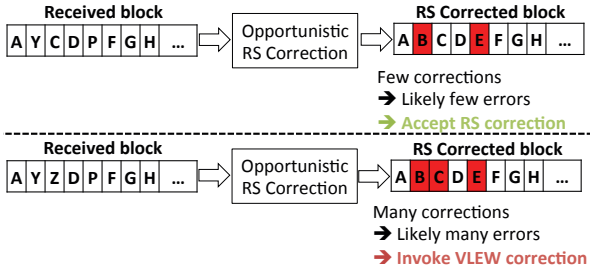

Fig. 8. When to accept (TOP) and when to reject (BOTTOM) the opportunistic bit error correction of per-block ECC.

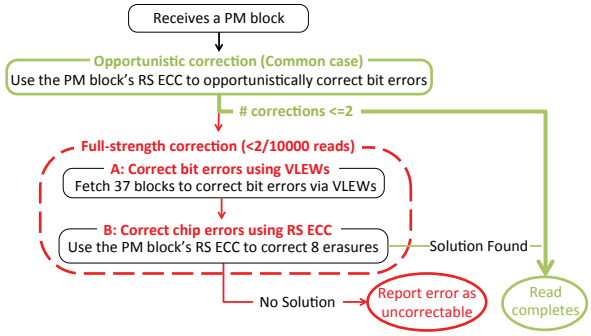

Fig. 9. Correction at runtime.

be miscorrected, and fall back on VLEW correction in those cases, as shown in Figure 8. Based on Figure 7, we set the threshold to two errors because $> 99.98\%$ of accesses have two or fewer errors. When using RS ECC to correct up to two bits of errors, the SDC rate is $3.3 \cdot 10^{-22}$ (see Appendix), which is several orders of magnitude lower than the target rate.

Figure 9 shows the complete error correction procedure at runtime. After receiving a memory block from off-chip persistent memory, the memory controller uses the block's RS ECC to correct errors; if RS ECC makes no more than two corrections, the memory controller accepts the RS correction results and sends the corrected memory block to the last level cache (LLC). If RS ECC either makes more than two corrections or recognizes the errors as uncorrectable, the memory controller fetches VLEWs to correct bit errors. Using VLEWs to correct bit errors frees up per-block RS ECC to correct errors from chip failures; as such, normal operation continues to benefit from chip failure protection.

On average, 0.018% of reads require fetching VLEWs to correct errors. This translates to a bandwidth overhead of $0.018\% \cdot 36 = 0.6\%$, which is much lower compared to 140% to 360% overhead when using VLEWs alone (Figure 5).

D. VLEW Updates at Runtime

VLEW code bits must be updated for every write request. Calculating new VLEW code bits for every write requires a read-modify-write operation, incurring a 200% bandwidth overhead (see bottom of Figure 5). However, persistent memory applications frequently write dirty persistent memory

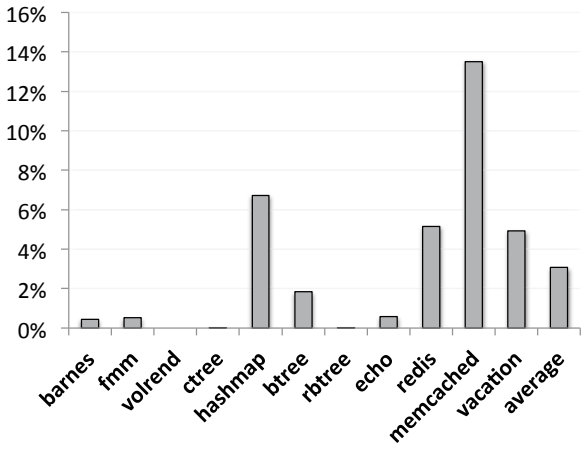

Fig. 10. Fraction of cachelines in the cache hierarchy (one 4MB LLC and four 64KB L1) that are occupied by dirty persistent memory blocks.

blocks to memory proactively through cacheline cleaning instructions such as *clwb* and *clflush* [1]–[6]. As such, dirty persistent memory blocks occupy only a small amount of on-chip cache capacity. Figure 10 shows that dirty persistent memory blocks occupy 4% of the total number of cachelines in the on-chip cache hierarchy, on average across our evaluated persistent memory applications (see Section VI). Based on the observation, we propose preserving the old memory values (OMVs) of dirty persistent memory blocks in the last level cache (LLC) at only a small dynamic cost to usable LLC capacity; preserving the old memory values (OMVs) of dirty persistent memory blocks helps to avoid the bandwidth overhead of fetching OMVs from off-chip before writing to persistent memory.

To preserve OMVs of dirty persistent memory blocks in LLC, we add two bits to each LLC cacheline's tag. One is the "SameAsMem" (*SAM*) bit, which records whether the cacheline currently has same value as off-chip persistent memory. The other is the OMV bit, which records whether the cacheline holds the OMV of a dirty persistent memory block. While both bits record whether a cacheline has same value as off-chip persistent memory, the difference is that cachelines with SAM bits set are visible to/accessible by memory instructions, but cachelines with their OMV bits set are not.

LLC cachelines update their SAM and OMV bits as follows. After being filled with data arriving from persistent memory or being cleaned by a cacheline cleaning instruction, an LLC cacheline sets its SAM bit; a cacheline resets its SAM bit after being filled with data from a dirty writeback from an upper-level cache. When receiving a dirty writeback to a cacheline with its SAM bit set, LLC preserves the cacheline's OMV by setting the cacheline's OMV bit and allocating a different way in the same cacheset to handle the dirty writeback.

The LLC uses the SAM and OMV bits as follows. Before writing back or cleaning a dirty block, LLC searches within the dirty block's set for a block with its OMV bit set and has the same address as the dirty block. When finding such a matching block, LLC computes the bitwise XOR of the block's value (which is an OMV) and the dirty block's value and sends the result to the memory controller to save the memory controller from explicitly fetching OMV from persistent memory for the memory write request; LLC also removes the matching block since its value will no longer equal off-chip persistent memory value after the memory write request. A cacheline cleaning instruction can also clean a dirty persistent memory block currently residing in an upper-level cache to off-chip memory; when such a dirty block passes through LLC, LLC looks for a matching LLC block with a set SAM or OMV bit, uses the LLC block's value to compute the bitwise XOR to send to the memory controller, and removes the LLC block if its OMV bit is set.

While fetching OMV from LLC eliminates the bandwidth overhead of reading old block, the processor still needs to send OMV to NVRAM chips to compute the ECC update (see Figure 5); this still incurs 100% bandwidth overheads for writes to persistent memory. To address this final challenge, we propose piggybacking the old block in the new block by modifying each write request to persistent memory to send the bitwise sum of the two blocks to memory, instead of just the new block as do conventional write requests. Upon receiving a bitwise sum from a write request, each NVRAM chips can internally recover the new data by simply bitwise subtracting the old data stored in the NVRAM chip from the received bitwise sum of old and new data. Each NVRAM chip can also use the received bitwise sum to directly encode the update to the VLEW code bits because the BCH code is linear [76] like most ECCs used in memory and storage systems. Recall from Section IV-B that ECC update is $f(x) \oplus f(x')$; because f is linear, $f(x) \oplus f(x') = f(x \oplus x')$. $x \oplus x'$ is the bitwise sum sent by a memory write.

Figure 11 shows the support needed within an NVRAM chip. On receiving a bitwise sum, each NVRAM chip fetches x' (old data) from the open row, bitwise subtracts x' from the received bitwise sum, and writes back the result (i.e., x or new data) to the open row; the ability to internally read-modify-write data is supported in current and emerging memory chips [29], [80]. To update the VLEW code bits, we

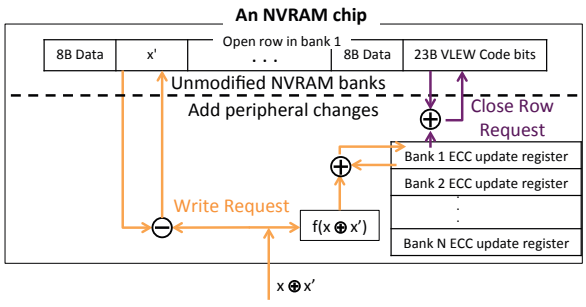

Fig. 11. Hardware support for NVRAM chips to internally update data and VLEW code bits from the bitwise sum received from the proposed method of writing to memory.

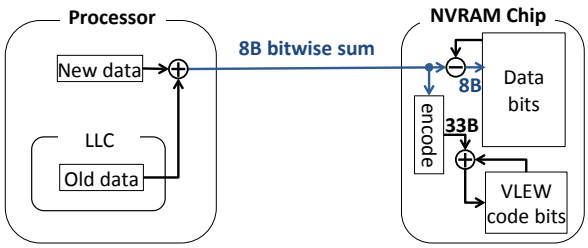

Fig. 12. Summary of how to efficiently update VLEWs at runtime when writing to persistent memory.

note that since each VLEW contains 256B of data, the same VLEW in an open row may be written many times due to row buffer locality; as such, an NVRAM chip may coalesce all updates for the same VLEW code bits into a single ECC update and only use the coalesced ECC update to modify the row's content when closing the row. The coalesced ECC updates can be temporarily stored in a small ECC Update Registerfile (EUR). Each EUR register stores the bitwise sum of all the ECC updates from writes to the same VLEW in an open row; as such, an EUR only requires $B \cdot R/256$ total registers, where B is the number of memory banks and R is row size in bytes. When receiving a row close request, an NVRAM chip must first drain the coalesced ECC updates to the row before closing the row; for each nonempty register in the EUR that corresponds to the closing row, the NVRAM chip internally fetches the corresponding VLEW code bits from the row, bitwise XORs them with the register value, and writes back the result to the row. The latency overhead to a row close request is deterministic for the processor; as such, the EUR is compatible with future NVRAM chips with deterministic latency interfaces.

Figure 12 summarizes how to write to persistent memory. It eliminates the 200% write bandwidth overhead of protecting persistent memory with VLEWs (see Figure 5).

E. Discussions: Handling Permanent Faults, Compatibility with Write Leveling, Area/Latency Overheads

A permanently faulty chip in a rank may cause repeated errors in many blocks and, therefore, frequent VLEW correction, which in turn incurs high performance overheads. One solution to mitigate this problem is to retire memory affected by permanent chip failure after correcting its data and migrating them elsewhere. After correcting a faulty chip, many systems today retire the affected memory to avoid uncorrectable errors occurring due to another chip failing in the same rank later [81]–[85]; memory retirement will likely be common place for persistent memory where uncorrectable errors can cause permanent data corruption. Another solution is to remap the contents of the faulty chip to an ECC chip in the faulty rank, at the cost of replacing the per-block RS ECC bits. To efficiently correct bit errors without per-block RS ECC bits, the memory controller dynamically re-encodes each VLEW in the faulty rank from 256B of data across *all* surviving chips in the rank; recall in healthy ranks without chip failures, the memory controller encodes each VLEW from 256B of data in a *single* chip (see Figure 6). Because each reconfigured VLEW contains $256B/64B = 4$ 64B blocks, each striped across the rank, using it to correct bit errors only requires fetching four data blocks via four regular requests. Reconfiguration maintains the same VLEW length and strength and, therefore, incurs no additional capacity overheads.

Permanent bit faults can develop in individual memory blocks due to the limited write endurance of NVRAMs; NVRAM-based memory systems may need to disable individual worn-out blocks [86]. While VLEWs protect data at much coarser granularity than individual blocks, they are compatible with disabling individual blocks. When disabling a block, the memory controller can simply update the VLEW code bits assuming the physical bits corresponding to the disabled block in the VLEW hold only zeros. Similarly, when fetching a VLEW for error correction, the values of the physical bits in the VLEW corresponding to the disabled block can be logically replaced by zeros prior to performing error correction. Note that any underlying block disabling mechanism must already pay the overhead of tracking which blocks have been disabled; as such, applying VLEW protection to a memory system with block disabling does not incur additional storage overheads. To identify worn-out blocks, prior works check whether errors remain in a block after error correction by re-reading the block right after writing it to memory [86]. This is also compatible with our proposal; after performing VLEW correction for a block, the memory controller may write the corrected block back to memory in the conventional manner (i.e., send raw data, instead of bitwise XOR, to directly overwrite data in memory) and re-read the block to identify worn-out blocks.

Prior works level wear between different data blocks by dynamically remapping blocks to different memory locations [87]. To support wear leveling, after remapping a block, the memory controller can update VLEW code bits assuming the physical bits that previously held the remapped block now contains only zeros, similar to handling block disabling above. Prior works level wear between ECC and data bits by periodically rotating the physical cells for storing ECC [88]; to support ECC leveling, while refreshing each row, the memory controller may instruct NVRAM chips to reserve a different group of bits in the row for storing VLEW code bits for the next refresh period.

Updating VLEW code bits for each write request requires increasing the number of code bits per write request and, therefore, may reduce NVRAM's write lifetime. Prior work report that increasing write latency can effectively improve NVRAM write lifetime; as such, we make up for loss in write lifetime by increasing write latency [89]. Our evaluation accounts for the write latency overheads (see Section VI).

The proposal requires embedding BCH encoders in NVRAM chips. BCH code bits can be computed in parallel via one XOR tree per code bit; this allows simple memory-array-like layout using only two metal layers, as shown in Figure 13. Using CACTI [90], we calculate area to be $0.1mm^2$ when assuming only semi-global metal wires and that each logic gate

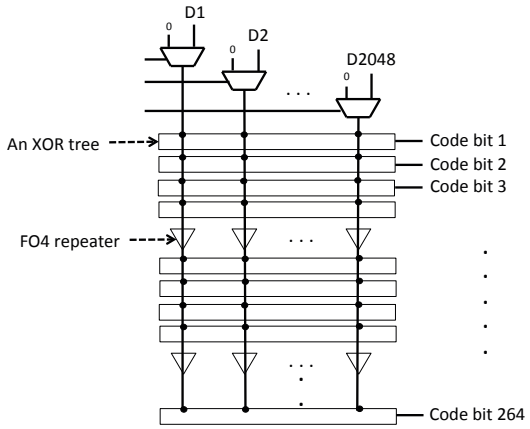

Fig. 13. Circuit diagram of an encoder for 22-bit-EC BCH with 256B data.

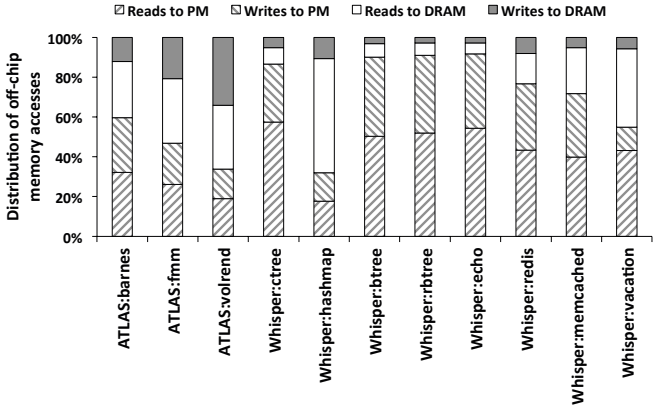

Fig. 14. Workload characterization: off-chip memory access breakdown.

equals two SRAM cells in size (similar to [91]). We estimate latency to be 1.6ns using LSTP bulk transistor latency [92].

Under $2 \cdot 10^{-4}$ BER, $1/200$ and $1.8/10000$ of reads need multi-error RS correction and BCH correction, respectively. We estimate latency of correcting a multi-byte error using RS ECC to be 45ns and the area requirement to be $0.002mm^2$ based on latency and area reported for an 8-byte-EC RS decoder in [93], after adjusting for process technology and codeword length. We estimate latency of 22-EC BCH ECC to be 200ns and area to be $0.05mm^2$ based on a 32-EC BCH decoder reported in [94], after adjusting for process technology and codeword length.

VI. Methodology

We evaluated WHISPER persistent memory benchmarks [2] and SPLASH3 [95] benchmarks running in ATLAS [1], a persistent memory library. To stress the memory system, we increased the problem size of each benchmark to the maximum supported by our available software and hardware. For example, we increased the MemCached capacity setting from the default of 64MB in WHISPER to 1GB. The total memory footprint of the workloads range from 2GB to 20GB. We evaluate the workloads in Gem5 [96] by executing them in an OS running in a simulated X86 processor; the OS is Ubuntu Server 16, a recent Linux distribution. The WHISPER workloads take a long time to initialize because they begin with an empty in-memory database or data structure and gradually fill it up transaction by transaction; as such, we warmup the workloads in native execution speed via Gem5 KVM CPU until the workloads' resident memory sizes, as reported by the simulated Ubuntu OS' TOP utility, have increased to steady levels. The native execution warmup times range from 2-10 minutes. After native execution warmup, we use Gem5's functional simulation to warmup the simulated processor's cache for 500ms of simulated time. Finally, we use Gem5's cycle-accurate simulation to measure the workload's performance during 20ms of simulated time.

Table I shows the microarchitectural parameters of the simulated processor. We simulated four cores per processor, similar to [2]. For each Whisper benchmark, we simulate multiple processes of the same benchmark, with a single-thread per process. We use IPC as the performance metric for these single-thread Whisper workloads. For each SPLASH3 benchmark, we simulate one process with four threads; we use FLOPS as their performance metric because they are parallel floating-point-heavy scientific workloads. All Whisper and ATLAS workloads utilize both persistent memory and volatile DRAM; as such, we modeled a hybrid memory channel with one rank of DRAM and one rank of persistent memory. We map the persistent memory address ranges, as specified in each WHISPER workload, to the NVRAM ranks, and map the remainder to DRAM ranks. For SPLASH3 running under ATLAS, we kept ATLAS' default setting of allocating all heap objects in persistent memory. Figure 14 shows the breakdown

TABLE I
Microarchitectural Parameters

Core	4 cores, 3GHz, 4-issue OOO 168 ROB entries, 64B cacheline
L1 d-cache, i-cache	2-way, 64KB, 1 cycle
Shared LLC	32-way, 4 MB, 14 cycles
Memory Controller	128 read buffer, 128 write buffer/channel closed page policy, FR-FCFS
Memory System	One 2400Mhz channel, with 1 DRAM rank and 1 persistent memory rank; 16 banks/rank

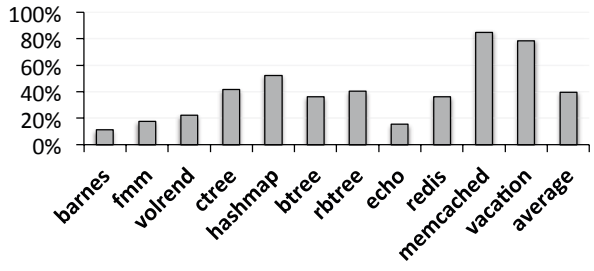

Fig. 15. Workload characterization: the ratio between the number of updates to VLEWs and the number of off-chip write requests to persistent memory.

of off-chip memory accesses; all benchmarks significantly exercise persistent memory.

We incorporate Ramulator [97] in Gem5 to simulate memory performance. We use Ramulator's default 2400Mhz DDR4 parameters, default FR-FCFS command scheduling policy, and default row buffer policy, which closes a page after 50ns of inactivity. Similar to [42], we model NVRAMs by modifying DRAM timing parameters because datasheets for dense NVRAMs chips are not yet available. We set the NVRAM rank's $tRCD$ and tWR parameters to the NVRAM read and write latencies, respectively. For ReRAM, we model 120ns read and 300ns write latencies, similar to [89]. For PCM, we model 250ns read latency by taking the 250ns eM-metric reported in [60]. We model 600 NVRAM write latency, which is in the middle of the 100ns - 1000ns write latency range described in [60].

To model the proposal, we modify LLC to cache OMVs. We model the read bandwidth overhead due to fetching VLEWs to correct bit errors for 0.02% of read requests (see Section V-C) by randomly force-prefetching 37 blocks at 0.02% probability. Recall from Section V-E we increase write latency to make up for loss in write lifetime due to updating VLEW code bits, which increases the number of physical bits written per write request. The number of physical bits written per write request increases by $33B/8B \cdot C$, where C is the ratio between the number of writes to VLEW code bits and the number of write requests to persistent memory; recall from Section V-D that a dirty row's VLEW code bits are written only once when the memory controller closes the dirty row. When evaluating a workload, we pessimistically assume the worst-case (i.e., linear) relationship between write endurance and write latency [89] and, therefore, increase tWR by $33B/8B{\cdot}C$; Figure 15 shows the C factor we measured for each workload and, therefore, used to calculated the increased tWR when evaluating the proposal. We note that C depends on a workload's spatial locality; to stress the proposed memory system design, we maximize C by setting data item size to a small 64B for all Whisper benchmarks with adjustable data item sizes (i.e., $echo$, $memcached$, $hashmap$, $btree$, and $rbtree$). Finally, to account for the 1.6ns BCH encoder latency (see Section V-E) and internal read of old data (see Figure 11), we pessimistically increase tWR by yet another 20ns.

VII. Experimental Results

Figure 16 and Figure 17 show the proposal's performance normalized to the bit-error correction baseline under ReRAM latencies and under PCM latencies, respectively. The proposal incurs a slightly higher average performance overhead (i.e., 2.3% vs. 1.4%) under PCM latencies than for ReRAM latencies. This is because we modeled a much longer baseline write latency (i.e., 600ns) for PCM than for ReRAM (i.e., 300ns), which increases the impact of the proposal's write latency overhead on overall performance. On average across both sets of evaluations, the proposal incurs a performance overhead of 2%. We believe this is a small cost for providing chip failure protection for persistent memory. In a large-scale field study, Vilas et al. [98] report that chipkill-correct provides 40X reliability improvement; we expect similar reliability improvement compared to protecting persistent memory with only bit error correction.

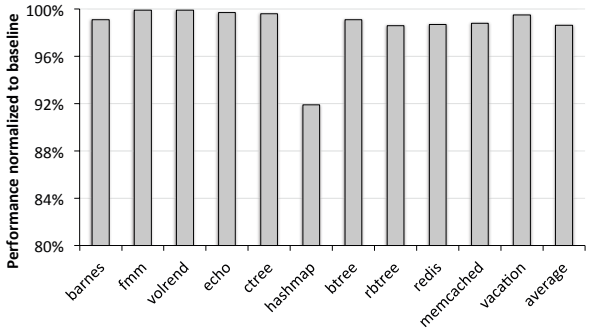

Fig. 16. Performance normalized to baseline for ReRAM latencies (i.e., baseline has 120ns tRCD and 300ns tWR).

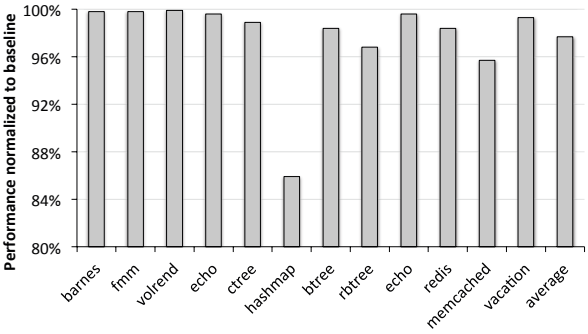

Fig. 17. Performance normalized to baseline for PCM latencies (i.e., baseline has 250ns tRCD and 600ns tWR.)

In the worst case (i.e,. $hashmap$), the proposal incurs 14% performance overhead. According to Whisper description [2], $hashmap$ performs only write queries, such as item deletion and modification; as such, $hashmap$ represents the worst-case workload for the proposal, which requires increasing write latency to provide iso-write-endurance as the baseline. We note that $ctree$, $btree$, and $rbtree$ also perform only write queries [2]; however, the proposal's performance for these workloads is $>= 96.8\%$ normalized to the baseline. We believe these workloads' lower sensitivity to write latency is due to the pointer-chasing memory access pattern of tree data structures, which reads from few banks (e.g., one) at a time and, therefore, reduces the probability of reading from a bank with on-going write.

While other Whisper workloads such as $memcached$, $echo$, $redis$, and $vacation$ also have a high ratio of write to read memory requests (see Figure 14), they are also not performance-sensitive to the increased write latency (see Figures 16, 17). Unlike the previously discussed workloads, this group of workloads process a network request for each query, which takes up a significant portion of the query's execution time and, therefore, reduces the overall performance impact of

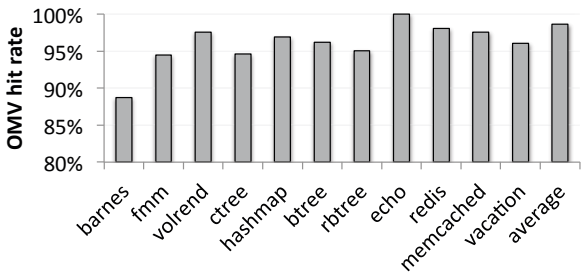

Fig. 18. Fraction of writes to persistent memory whose OMV is served from LLC instead of off-chip memory.

increased write latency.

Figure 18 shows the fraction of writes to persistent memory whose OMV is served from LLC instead of off-chip memory. On average across all the workloads, the hit rate is 98.6%. As such, on average only 1.4% of writes to persistent memory incurs the overhead of fetching from OMV from off-chip memory. Surprisingly, $barnes$ has the highest OMV miss rate - 11% -, despite only occupying 0.5% of cache capacity with dirty persistent memory blocks (see Figure 10). We believe this is because Gem5 does not enforce cache inclusivity and, therefore, reduces the probability of finding a matching LLC block when L1 cleans a dirty persistent memory block to memory or writes back to LLC.

VIII. Conclusion

This paper explores the problems and challenges of protecting dense NVRAM-based persistent memory with chipkill-correct (i.e., protection against both bit errors and chip failures). Chipkill-correct is a standard feature in today's volatile server main memory. Because errors in persistent memory can cause permanent data loss, which is more severe than the loss of data in volatile memory, persistent memory requires at least the same level of protection as volatile server memory (i.e., chipkill-correct). Chipkill-correct for dense NVRAM-based persistent memory is challenging due to high RBER after a long time (i.e., a week to a year) without refresh. The ability to tolerate high RBER enables the reliable survival of data in NVRAMs after system crashes/power outages, which is essential for NVRAMs to serve as persistent memory. Simply extending DRAM chipkill-correct techniques to dense NVRAM-based persistent memory requires prohibitive storage overhead of $>= 69\%$.

To efficiently protect dense NVRAM-based persistent memory with chipkill-correct, we decouple correction of errors at boot time from correction of errors at runtime to simultaneously achieve low storage cost and low performance cost. We use very long ECC words (VLEWs) to ensure reliable data survival for a week to a year without refresh by correcting the maximum number of bit errors at minimum storage cost; at runtime when RBER is lower, we use each memory block's chip failure protection bits to opportunistically correct bit errors at high performance. The proposal incurs only a total storage cost of 27%. Compared to protecting persistent memory with only bit error correction, the proposed chipkill-correct adds chip failure protection at no additional storage cost and only 2% average performance overhead.

Acknowledgment

We thank Rakesh Kumar from the University of Illinois at Urbana-Champaign for his insightful comments. We thank Changhee Jung and Qing Rui from Virginia Tech for their help showing us how to run applications under ATLAS. We also thank Advanced Research Computing at Virginia Tech for providing computational resources and technical support for generating the experimental results in the paper.

Appendix: Miscorrection Probability Calculation

We calculate miscorrection probability as the product of two terms. Term A is the probability of having a noncodeword containing at least the threshold number of errors required to cause miscorrection (i.e., n_{th}); Term B is the probability RS ECC will decode such a noncodeword (i.e., an invalid word under a given code) into a codeword (i.e., a valid word under a given code). Term A can be obtained using standard combinatorial probability analysis by taking as inputs n_{th}, RBER, and the total number of data bytes (i.e., k) and check bytes (i.e., r) each word contains. To obtain Term B, we denote as t the maximum number of errors one decides to correct in a codeword. Because RS ECC decodes any noncodeword within t Hamming distance[5] from a codeword into the codeword [76], Term B is the probability that an uncorrectable noncodeword is $<= t$ distance from an unintended codeword; on average, this probability equals the total number of noncodewords that are $<= t$ distance away from each codeword (i.e. ${}^{(k+r)}C_t \cdot 2^{8 \cdot t}$) multiplied by the total number of possible codewords (i.e., $2^{8 \cdot k}$), and divided by the total number of possible words (i.e., $2^{8 \cdot (k+r)}$) [76].

For our per-block RS codewords, $k = 64$, $r = 8$, and every pair of codeword has a minimum distance of $r + 1 = 9$; when using the RS ECC to correct $t = 4$ errors, miscorrection may occur for a noncodeword with $n_{th} = 5$ errors. Using these values and RBER$= 2 \cdot 10^{-4}$, Term A and B are $1.3 \cdot 10^{-7}$ and $2.4 \cdot 10^{-4}$ respectively; this translate to an SDC rate of $3.2 \cdot 10^{-11}$. When $t = 2$, $n_{th} = 9 - 2 = 7$ because only noncodewords with 7 errors from their intended codeword can be within two Hamming distance from an unintended codeword and thus be mis-corrected into the unintended codeword; increasing n_{th} from 5 to 7 reduces Term A to $3.6 \cdot 10^{-11}$. A smaller $t = 2$ value also reduces Term B to $9.1 \cdot 10^{-12}$. This translates to an overall SDC rate of $3.3 \cdot 10^{-22}$ when $t = 2$.

[5]For RS ECC, Hamming distance between two words is how many bytes the two words differ from one another.

REFERENCES

[1] D. R. Chakrabarti, H.-J. Boehm, and K. Bhandari, "Atlas: Leveraging locks for non-volatile memory consistency," *SIGPLAN Not.*, vol. 49, no. 10, pp. 433–452, Oct. 2014. [Online]. Available: http://doi.acm.org/10.1145/2714064.2660224

[2] S. Nalli, S. Haria, M. D. Hill, M. M. Swift, H. Volos, and K. Keeton, "An analysis of persistent memory use with whisper," *SIGOPS Oper. Syst. Rev.*, vol. 51, no. 2, pp. 135–148, Apr. 2017. [Online]. Available: http://doi.acm.org/10.1145/3093315.3037730

[3] S. Pelley, P. M. Chen, and T. F. Wenisch, "Memory persistency," *SIGARCH Comput. Archit. News*, vol. 42, no. 3, pp. 265–276, Jun. 2014. [Online]. Available: http://doi.acm.org/10.1145/2678373.2665712

[4] J. Zhao, S. Li, D. H. Yoon, Y. Xie, and N. P. Jouppi, "Kiln: Closing the performance gap between systems with and without persistence support," in *2013 46th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, Dec 2013, pp. 421–432.

[5] A. Kolli, S. Pelley, A. Saidi, P. M. Chen, and T. F. Wenisch, "High-performance transactions for persistent memories," *SIGPLAN Not.*, vol. 51, no. 4, pp. 399–411, Mar. 2016. [Online]. Available: http://doi.acm.org/10.1145/2954679.2872381

[6] W.-H. Kim, J. Kim, W. Baek, B. Nam, and Y. Won, "Nvwal: Exploiting nvram in write-ahead logging," *SIGPLAN Not.*, vol. 51, no. 4, pp. 385–398, Mar. 2016. [Online]. Available: http://doi.acm.org/10.1145/2954679.2872392

[7] K. Doshi, E. Giles, and P. Varman, "Atomic persistence for scm with a non-intrusive backend controller," in *2016 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, March 2016, pp. 77–89.

[8] A. Joshi, V. Nagarajan, S. Viglas, and M. Cintra, "Atom: Atomic durability in non-volatile memory through hardware logging," in *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, Feb 2017, pp. 361–372.

[9] T. C.-H. Hsu, H. Brügner, I. Roy, K. Keeton, and P. Eugster, "Nvthreads: Practical persistence for multi-threaded applications," in *Proceedings of the Twelfth European Conference on Computer Systems*, ser. EuroSys '17. New York, NY, USA: ACM, 2017, pp. 468–482. [Online]. Available: http://doi.acm.org/10.1145/3064176.3064204

[10] L. Marmol, J. Guerra, and M. K. Aguilera, "Non-volatile memory through customized key-value stores," in *8th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage 16)*. Denver, CO: USENIX Association, 2016. [Online]. Available: https://www.usenix.org/conference/hotstorage16/workshop-program/presentation/marmol

[11] S. Shin, S. K. Tirukkovalluri, J. Tuck, and Y. Solihin, "Proteus: A flexible and fast software supported hardware logging approach for nvm," in *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO-50 '17. New York, NY, USA: ACM, 2017, pp. 178–190. [Online]. Available: http://doi.acm.org/10.1145/3123939.3124539

[12] N. Cohen, M. Friedman, and J. R. Larus, "Efficient logging in non-volatile memory by exploiting coherency protocols," *Proc. ACM Program. Lang.*, vol. 1, no. OOPSLA, pp. 67:1–67:24, Oct. 2017. [Online]. Available: http://doi.acm.org/10.1145/3133891

[13] M. Wu, Z. Zhao, H. Li, H. Li, H. Chen, B. Zang, and H. Guan, "Espresso: Brewing java for more non-volatility with non-volatile memory," in *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '18. New York, NY, USA: ACM, 2018, pp. 70–83. [Online]. Available: http://doi.acm.org/10.1145/3173162.3173201

[14] ITIC, "Itic 2015 - 2016 global server hardware, server os reliability report," 2015, http://www.lenovo.com/images/products/system-x/pdfs/white-papers/itic_2015_reliability_wp.pdf.

[15] ——, "Intel xeon processor e7 family reaches reliability parity with risc/unix, delivers 99.999% reliability, availability and serviceability," June 2013, http://www.intel.com/content/dam/www/public/us/en/documents/white-papers/xeon-e7-ras-itic-paper.pdf.

[16] H. P. Enterprise, "Hp advanced memory error detection technology," [online]. Available:http://h20565.www2.hpe.com/hpsc/doc/public/display?docId=emr_na-c02878598&lang=en-us&cc=us.

[17] AMD, "AMD, BIOS and Kernel Developer's Guide for AMD NPT Family 0Fh Processors," 2009, [Online]. Available:http://developer.amd.com/wordpress/media/2012/10/325591.pdf.

[18] Intel, "Intel E7500 Chipset MCH Intel x4 SSDC," 2002, http://www.intel.com/content/www/us/en/chipsets/e7500-chipset-mch-x4-single-device-data-correction-note.html.

[19] DELL, "Memory for Dell PowerEdge 12th Generation Servers," 2012, [Online]. Available:http://www.dell.com/downloads/global/products/pedge/poweredge_12th_generation_server_memory.pdf.

[20] T. J. Dell, "A White Paper on the Benefits of Chipkill Correct ECC for PC Server Main Memory," 1997, IBM Microelectronics Division.

[21] D. H. Yoon and M. Erez, "Virtualized and flexible ecc for main memory," in *Proceedings of the Fifteenth Edition of ASPLOS on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS XV. New York, NY, USA: ACM, 2010, pp. 397–408.

[22] A. N. Udipi, N. Muralimanohar, R. Balsubramonian, A. Davis, and N. P. Jouppi, "LOT-ECC: LOcalized and Tiered Reliability Mechanisms for Commodity Memory Systems," *ISCA*, pp. 285 – 296, 2012.

[23] X. Jian and R. Kumar, "Adaptive Reliability Chipkill Correct (ARCC)," in *High Performance Computer Architecture (HPCA2013), 2013 IEEE 19th International Symposium on*, 2013, pp. 270–281.

[24] X. Jian, H. Duwe, J. Sartori, V. Sridharan, and R. Kumar, "Low-power, low-storage-overhead chipkill correct via multi-line error correction," in *Proceedings of SC13: International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '13. New York, NY, USA: ACM, 2013, pp. 24:1–24:12. [Online]. Available: http://doi.acm.org/10.1145/2503210.2503243

[25] X. Jian and R. Kumar, "ECC Parity: A technique for efficient memory error resilience for multi-channel memory systems," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '14. Piscataway, NJ, USA: IEEE Press, 2014, pp. 1035–1046. [Online]. Available: http://dx.doi.org/10.1109/SC.2014.89

[26] X. Jian, V. Sridharan, and R. Kumar, "Parity helix: Efficient protection for single-dimensional faults in multi-dimensional memory systems," in *2016 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, March 2016, pp. 555–567.

[27] J. Kim, M. Sullivan, and M. Erez, "Bamboo ecc: Strong, safe, and flexible codes for reliable computer memory," in *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*, Feb 2015, pp. 101–112.

[28] P. J. Nair, V. Sridharan, and M. K. Qureshi, "Xed: Exposing on-die error detection information for strong memory reliability," in *Proceedings of the 43rd Annual International Symposium on Computer Architecture*, ser. ISCA '16, 2016.

[29] S. Cha, S. O, H. Shin, S. Hwang, K. Park, S. J. Jang, J. S. Choi, G. Y. Jin, Y. H. Son, H. Cho, J. H. Ahn, and N. S. Kim, "Defect analysis and cost-effective resilience architecture for future dram devices," in *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, Feb 2017, pp. 61–72.

[30] S. L. Gong, M. Rhu, J. Kim, J. Chung, and M. Erez, "Clean-ecc: High reliability ecc for adaptive granularity memory system," in *2015 48th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, Dec 2015, pp. 611–622.

[31] S. L. Gong, J. Kim, S. Lym, M. Sullivan, H. David, and M. Erez, "Duo: Exposing on-chip redundancy to rank-level ecc for high reliability," in *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, February 2018.

[32] A. Athmanathan, "Multilevel-cell phase-change memory," Ph.D. dissertation, STI, Lausanne, 2016.

[33] M. Awasthi, M. Shevgoor, K. Sudan, B. Rajendran, R. Balasubramonian, and V. Srinivasan, "Efficient scrub mechanisms for error-prone emerging memories," in *Proceedings of the 2012 IEEE 18th International Symposium on High-Performance Computer Architecture*, ser. HPCA '12. Washington, DC, USA: IEEE Computer Society, 2012, pp. 1–12. [Online]. Available: http://dx.doi.org/10.1109/HPCA.2012.6168941

[34] H. Naeimi, C. Augustine, A. Raychowdhury, L. Shih-Lien, and J. Tschanz, "Sttram scaling and retention failure." *Intel Technology Journal*, vol. 17, no. 1, pp. 54 – 75, 2013.

[35] B. Gao, H. Zhang, B. Chen, L. Liu, X. Liu, R. Han, J. Kang, Z. Fang, H. Yu, B. Yu, and D. L. Kwong, "Modeling of retention failure behavior in bipolar oxide-based resistive switching memory," *IEEE Electron Device Letters*, vol. 32, no. 3, pp. 276–278, March 2011.

[36] J. Frascaroli, F. G. Volpe, S. Brivio, and S. Spiga, "Effect of al doping on the retention behavior of hfo2 resistive switching memories,"

Microelectron. Eng., vol. 147, no. C, pp. 104–107, Nov. 2015. [Online]. Available: https://doi.org/10.1016/j.mee.2015.04.043

[37] S. Yu, Y. Yin Chen, X. Guan, H.-S. Philip Wong, and J. A. Kittl, "A Monte Carlo study of the low resistance state retention of HfO_x based resistive switching memory," *Applied Physics Letters*, vol. 100, no. 4, p. 043507, Jan. 2012.

[38] D. Acharyya, A. Hazra, and P. Bhattacharyya, "A journey towards reliability improvement of tio2 based resistive random access memory: A review," *Microelectronics Reliability*, vol. 54, no. 3, pp. 541 – 560, 2014. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0026271413004344

[39] S. Dolinar, D. Divsalar, and F. Pollara, "Code performance as a function of block size," in *The Telecommunications and Mission Operations Progress Report*, vol. 42-133, 1998, pp. 1–23.

[40] B. Aichinger, "DDR5: The new JEDEC standard for Computer Main Memory," 2017, [Online]. Available: https://www.futureplus.com/ddr5-the-new-jedec-standard-for-computer-main-memory/.

[41] L. E. Ramos, E. Gorbatov, and R. Bianchini, "Page placement in hybrid memory systems," in *Proceedings of the International Conference on Supercomputing*, ser. ICS '11. New York, NY, USA: ACM, 2011, pp. 85–95. [Online]. Available: http://doi.acm.org/10.1145/1995896.1995911

[42] B. C. Lee, E. Ipek, O. Mutlu, and D. Burger, "Architecting phase change memory as a scalable dram alternative," *SIGARCH Comput. Archit. News*, vol. 37, no. 3, pp. 2–13, Jun. 2009. [Online]. Available: http://doi.acm.org/10.1145/1555815.1555758

[43] E. Kultursay, M. Kandemir, A. Sivasubramaniam, and O. Mutlu, "Evaluating stt-ram as an energy-efficient main memory alternative," *2013 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pp. 256–267, 2013.

[44] C. Xu, D. Niu, N. Muralimanohar, R. Balasubramonian, T. Zhang, S. Yu, and Y. Xie, "Overcoming the challenges of crossbar resistive memory architectures," in *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*, Feb 2015, pp. 476–488.

[45] M. K. Qureshi, M. M. Franceschini, L. A. Lastras-Montaño, and J. P. Karidis, "Morphable memory system: A robust architecture for exploiting multi-level phase change memories," *SIGARCH Comput. Archit. News*, vol. 38, no. 3, pp. 153–162, Jun. 2010. [Online]. Available: http://doi.acm.org/10.1145/1816038.1815981

[46] A. Prodromakis, N. Papandreou, E. Bougioukou, U. Egger, N. Toulgaridis, T. Antonakopoulos, H. Pozidis, and E. Eleftheriou, "Controller architecture for low-latency access to phase-change memory in openpower systems," in *2016 26th International Conference on Field Programmable Logic and Applications (FPL)*, Aug 2016, pp. 1–4.

[47] L. Jiang, Y. Zhang, B. R. Childers, and J. Yang, "Fpb: Fine-grained power budgeting to improve write throughput of multi-level cell phase change memory," in *Proceedings of the 2012 45th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO-45. Washington, DC, USA: IEEE Computer Society, 2012, pp. 1–12. [Online]. Available: http://dx.doi.org/10.1109/MICRO.2012.10

[48] S. Schechter, G. H. Loh, K. Strauss, and D. Burger, "Use ecp, not ecc, for hard failures in resistive memories," in *Proceedings of the 37th Annual International Symposium on Computer Architecture*, ser. ISCA '10. New York, NY, USA: ACM, 2010, pp. 141–152. [Online]. Available: http://doi.acm.org/10.1145/1815961.1815980

[49] R. Ramanujan, G. Hinton, and D. Zimmerman, "Dynamic partial power down of memory-side cache in a 2-level memory hierarchy," Oct. 9 2014, uS Patent App. 13/994,726. [Online]. Available: https://www.google.com/patents/US20140304475

[50] R. Ramanujan, D. Ziakas, D. Zimmerman, M. Kumar, M. Swaminathan, and B. Coury, "Apparatus and method for implementing a multi-level memory hierarchy over common memory channels," Apr. 19 2016, uS Patent 9,317,429. [Online]. Available: https://www.google.com/patents/US9317429

[51] S. Qawami and J. Hulbert, "Phase change memory in a dual inline memory module," Jan. 7 2014, uS Patent 8,626,997. [Online]. Available: https://www.google.com/patents/US8626997

[52] A. Akel, A. M. Caulfield, T. I. Mollov, R. K. Gupta, and S. Swanson, "Onyx: A protoype phase change memory storage array," in *Proceedings of the 3rd USENIX Conference on Hot Topics in Storage and File Systems*, ser. HotStorage'11. Berkeley, CA, USA: USENIX Association, 2011, pp. 2–2. [Online]. Available: http://dl.acm.org/citation.cfm?id=2002218.2002220

[53] R. Fackenthal, M. Kitagawa, W. Otsuka, K. Prall, D. Mills, K. Tsutsui, J. Javanifard, K. Tedrow, T. Tsushima, Y. Shibahara, and G. Hush, "19.7 a 16gb reram with 200mb/s write and 1gb/s read in 27nm technology," in *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, Feb 2014, pp. 338–339.

[54] L. Mearian, "Everspin ships first ST-MRAM memory with 500X performance of flash," 2012, [Online]. Available: http://www.computerworld.com/article/2493603/data-center/everspin-ships-first-st-mram-memory-with-500x-performance-of-flash.html.

[55] Everspin, "DDR3 DRAM Compatible MRAM - Spin Torque Technology," [Online]. Available: https://www.everspin.com/ddr3-dram-compatible-mram-spin-torque-technology.

[56] Intel, "Solve the most demanding storage and memory challenges with the Intel Optane SSD DC P4800X Series," 2017, [Online]. Available: https://www.intel.com/content/dam/www/public/us/en/documents/product-briefs/optane-ssd-dc-p4800x-brief.pdf.

[57] B. Tallis, "Intel Introduces Optane SSD DC P4800X With 3DXPoint Memory," 2017, [Online]. Available: https://www.anandtech.com/show/11208/intel-introduces-optane-ssd-dc-p4800x-with-3d-xpoint-memory.

[58] M. Chiappetta, "Intel Optane SSD DC P4800X With 3D Xpoint Memory Debuts Ultra-Low Latency Storage, New Memory Tier," 2017, [Online]. Available: https://hothardware.com/reviews/intel-optane-ssd-dc-p4800x-enterprise-storage-featuring-3d-xpoint-memory-technology-debuts.

[59] P. Stone, "Intel Optane DC P4800X SSD & Consumer 3D Crosspoint," 2017, [Online]. Available: https://www.gamersnexus.net/news-pc/2845-intel-optane-dc-p4800x-ssd.

[60] A. Athmanathan, M. Stanisavljevic, N. Papandreou, H. Pozidis, and E. Eleftheriou, "Multilevel-cell phase-change memory: A viable technology," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 6, no. 1, pp. 87–100, March 2016.

[61] W. S. Khwa, M. F. Chang, J. Y. Wu, M. H. Lee, T. H. Su, K. H. Yang, T. F. Chen, T. Y. Wang, H. P. Li, M. BrightSky, S. Kim, H. L. Lung, and C. Lam, "7.3 a resistance-drift compensation scheme to reduce mlc pcm raw ber by over 100x for storage-class memory applications," in *2016 IEEE International Solid-State Circuits Conference (ISSCC)*, Jan 2016, pp. 134–135.

[62] A. Calderoni, S. Sills, and N. Ramaswamy, "Performance comparison of o-based and cu-based reram for high-density applications," in *2014 IEEE 6th International Memory Workshop (IMW)*, May 2014, pp. 1–4.

[63] S. Sills, S. Yasuda, A. Calderoni, C. Cardon, J. Strand, K. Aratani, and N. Ramaswamy, "Challenges for high-density 16gb reram with 27nm technology," in *2015 Symposium on VLSI Circuits (VLSI Circuits)*, June 2015, pp. T106–T107.

[64] S. Sills, S. Yasuda, J. Strand, A. Calderoni, K. Aratani, A. Johnson, and N. Ramaswamy, "A copper reram cell for storage class memory applications," in *2014 Symposium on VLSI Technology (VLSI-Technology): Digest of Technical Papers*, June 2014, pp. 1–2.

[65] T. Parnell, "Nand flash basics & error characteristics; why do we need smart controllers?" *Flash Memory Summit*, 2017.

[66] Y. Cai, G. Yalcin, O. Mutlu, E. F. Haratsch, A. Cristal, and O. S. Unsal, "Error analysis and retention-aware error management for nand flash memory," *Intel Technology Journal*, vol. 17, 2013.

[67] M. Mao, P. Y. Chen, S. Yu, and C. Chakrabarti, "A multilayer approach to designing energy-efficient and reliable reram cross-point array system," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 25, no. 5, pp. 1611–1621, May 2017.

[68] N. H. Seong, S. Yeo, and H.-H. S. Lee, "Tri-level-cell phase change memory: Toward an efficient and reliable memory system," in *Proceedings of the 40th Annual International Symposium on Computer Architecture*, ser. ISCA '13. New York, NY, USA: ACM, 2013, pp. 440–451. [Online]. Available: http://doi.acm.org/10.1145/2485922.2485960

[69] CYPRESS, "SLC Versus MLC NAND Flash Memory," 2015, [Online]. Available: http://www.cypress.com/file/209181/download.

[70] R. Metzger, "Advantages of ecc-free nand in high performance applications," *Flash Memory Summit*, 2012.

[71] J. Cooke and M. Kim, "Making Informed Memory Choices," 2014, [Online]. Available: cache.freescale.com/files/training/doc/ftf/2014/FTF-IND-F0378.pdf.

[72] Toshiba, "SLC NAND & BENAND Reliability and Performance," August 2016, [Online]. Available: https://toshiba.semicon-storage.com/content/dam/toshiba-ss/ncsa/en_us/taec/components/FAB/SLC-NAND-BENAND-FAB.pdf.

[73] MICRON, "NOR NAND FLASH Guide," 2013, [Online]. Available: https://www.micron.com/~/media/documents/products/product-flyer/flyer_nor_nand_flash_guide.pdf.

[74] N. Chatterjee, M. OÕConnor, D. Lee, D. R. Johnson, S. W. Keckler, M. Rhu, and W. J. Dally, "Architecting an energy-efficient dram system for gpus," in *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, Feb 2017, pp. 73–84.

[75] C. Wilkerson, A. R. Alameldeen, Z. Chishti, W. Wu, D. Somasekhar, and S.-l. Lu, "Reducing cache power with low-cost, multi-bit error-correcting codes," in *Proceedings of the 37th Annual International Symposium on Computer Architecture*, ser. ISCA '10. New York, NY, USA: ACM, 2010, pp. 83–93. [Online]. Available: http://doi.acm.org/10.1145/1815961.1815973

[76] S. Lin and D. J. Costello, *Error Control Coding, Second Edition*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 2004.

[77] MICRON, "4Gb:x4, x8, x16 DDR4 SDRAM," https://www.micron.com/~/media/documents/products/data-sheet/dram/ddr4/4gb_ddr4_sdram.pdf.

[78] Sandisk, "Flash management: A detailed overview of flash management techniques," 2013.

[79] MICRON, "Error correction code (ecc) in micron® single-level cell (slc) nand," 2011, https://www.micron.com/~/media/documents/products/technical-note/nand-flash/tn2963_ecc_in_slc_nand.pdf.

[80] T. Y. Oh, H. Chung, J. Y. Park, K. W. Lee, S. Oh, S. Y. Doo, H. J. Kim, C. Lee, H. R. Kim, J. H. Lee, J. I. Lee, K. S. Ha, Y. Choi, Y. C. Cho, Y. C. Bae, T. Jang, C. Park, K. Park, S. Jang, and J. S. Choi, "A 3.2 gbps/pin 8 gbit 1.0 v lpddr4 sdram with integrated ecc engine for sub-1 v dram core operation," *IEEE Journal of Solid-State Circuits*, vol. 50, no. 1, pp. 178–190, Jan 2015.

[81] A. A. Hwang, I. A. Stefanovici, and B. Schroeder, "Cosmic Rays Don't Strike Twice: Understanding the Nature of DRAM Errors and the Implications for System Design," *SIGARCH Comput. Archit. News*, pp. 111–122, 2012.

[82] J. Abrams and D. Iler, "Pre-failure alerts provided by Dell PowerEdge server systems management," 2015, [Online]. Available: http://en.community.dell.com/techcenter/extras/m/white_papers/20441294/download.

[83] M. T. Chapman, "Introducing IBM X6 Technology," 2014, [Online]. Available: http://www.lenovo.com/images/products/system-x/pdfs/white-papers/XSW03145USEN.PDF.

[84] Microsoft, "Predictive Failure Analysis (PFA)," 2017, [Online]. Available: http://tinyurl.com/n34z657.

[85] Oracle, "Oracle Solaris and Oracle SPARC ServersÑ Integrated and Optimized for Mission Critical Computing," 2010, [Online]. Available: http://www.oracle.com/technetwork/server-storage/solaris/documentation/solaris-sparc-rf-final-175353.pdf.

[86] R. Azevedo, J. D. Davis, K. Strauss, P. Gopalan, M. Manasse, and S. Yekhanin, "Zombie memory: Extending memory lifetime by reviving dead blocks," *SIGARCH Comput. Archit. News*, vol. 41, no. 3, pp. 452–463, Jun. 2013. [Online]. Available: http://doi.acm.org/10.1145/2508148.2485961

[87] M. K. Qureshi, J. Karidis, M. Franceschini, V. Srinivasan, L. Lastras, and B. Abali, "Enhancing lifetime and security of pcm-based main memory with start-gap wear leveling," in *Proceedings of the 42Nd Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO 42. New York, NY, USA: ACM, 2009, pp. 14–23. [Online]. Available: http://doi.acm.org/10.1145/1669112.1669117

[88] H. Farbeh, H. Kim, S. G. Miremadi, and S. Kim, "Floating-ecc: Dynamic repositioning of error correcting code bits for extending the lifetime of stt-ram caches," *IEEE Transactions on Computers*, vol. 65, no. 12, pp. 3661–3675, Dec 2016.

[89] L. Zhang, B. Neely, D. Franklin, D. Strukov, Y. Xie, and F. T. Chong, "Mellow writes: Extending lifetime in resistive memories through selective slow write backs," in *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, June 2016, pp. 519–531.

[90] HPLabs, "Cacti 6.5," [online]. Available: http://www.hpl.hp.com/research/cacti/cacti65.tgz.

[91] A. R. Alameldeen, I. Wagner, Z. Chishti, W. Wu, C. Wilkerson, and S.-L. Lu, "Energy-efficient cache design using variable-strength error-correcting codes," in *Proceedings of the 38th Annual International Symposium on Computer Architecture*, ser. ISCA '11. New York, NY, USA: ACM, 2011, pp. 461–472. [Online]. Available: http://doi.acm.org/10.1145/2000064.2000118

[92] "The International Technology Roadmap for Semiconductors (ITRS), System Drivers," 2012, [Online]. Available: http://www.itrs2.net/2012-itrs.html.

[93] C. Yang, Y. Emre, and C. Chakrabarti, "Product code schemes for error correction in mlc nand flash memories," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 20, no. 12, pp. 2302–2314, Dec 2012.

[94] Y. Lee, H. Yoo, I. Yoo, and I. C. Park, "6.4gb/s multi-threaded bch encoder and decoder for multi-channel ssd controllers," in *2012 IEEE International Solid-State Circuits Conference*, Feb 2012, pp. 426–428.

[95] C. Sakalis, C. Leonardsson, S. Kaxiras, and A. Ros, "Splash-3: A properly synchronized benchmark suite for contemporary research," in *2016 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, April 2016, pp. 101–111.

[96] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, and D. A. Wood, "The gem5 simulator," *SIGARCH Comput. Archit. News*, vol. 39, no. 2, pp. 1–7, Aug. 2011. [Online]. Available: http://doi.acm.org/10.1145/2024716.2024718

[97] Y. Kim, W. Yang, and O. Mutlu, "Ramulator: A fast and extensible dram simulator," *IEEE Computer Architecture Letters*, vol. 15, no. 1, pp. 45–49, Jan 2016.

[98] V. Sridharan and D. Liberty, "A study of dram failures in the field," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, ser. SC '12. Los Alamitos, CA, USA: IEEE Computer Society Press, 2012, pp. 76:1–76:11. [Online]. Available: http://dl.acm.org/citation.cfm?id=2388996.2389100