# Binary Star: Coordinated Reliability in Heterogeneous Memory Systems for High Performance and Scalability

Xiao Liu[1] David Roberts Rachata Ausavarungnirun[2] Onur Mutlu[3] Jishen Zhao[1]
[1]UC San Diego [2]King Mongkut's University of Technology North Bangkok [3]ETH Zürich

## ABSTRACT

As memory capacity scales, traditional cache and memory hierarchy designs are facing increasingly difficult challenges in ensuring high reliability with low storage and performance cost. Recent developments in 3D die-stacked DRAM caches and nonvolatile memories (NVRAMs) introduce promising opportunities in tackling the reliability, performance, and capacity challenges, due to the diverse reliability characteristics of the technologies. However, simply replacing DRAM with NVRAM does not solve the reliability issues of the memory system, as conventional memory system designs maintain separate reliability schemes across caches and main memory. Our goal in this paper is to enable a reliable and high-performance memory hierarchy design, as memory capacity scales. To this end, we propose Binary Star, which coordinates the reliability schemes and consistent cache writeback between 3D-stacked DRAM last-level cache and NVRAM main memory to maintain the reliability of the cache and the memory hierarchy. Binary Star significantly reduces the performance and storage overhead of consistent cache writeback by coordinating it with NVRAM wear leveling. As a result, Binary Star is much more reliable and offers better performance than state-of-the-art memory systems with error correction. On a set of memory-intensive workloads, we show that Binary Star reduces memory failures in time (FIT) by 92.9% compared to state-of-the-art error correction schemes, while retaining 99% of the performance of a conventional DRAM design that provides no error correction.

## CCS CONCEPTS

• **Computer systems organization → Reliability**; **Processors and memory architectures**; • **Hardware → Emerging technologies**.

## KEYWORDS

reliability, nonvolatile memory, hybrid memory, memory systems

**ACM Reference Format:**
Liu, et al.. 2019. Binary Star: Coordinated Reliability in Heterogeneous Memory Systems for High Performance and Scalability. In *Proceedings of The 52nd Annual IEEE/ACM International Symposium on Microarchitecture, Columbus, OH, USA, October 12–16, 2019 (MICRO-52),* 14 pages.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
*MICRO-52, October 12–16, 2019, Columbus, OH, USA*
© 2019 Association for Computing Machinery.
ACM ISBN 978-1-4503-6938-1/19/10…$15.00
https://doi.org/10.1145/3352460.3358262

## 1 INTRODUCTION

Modern cloud servers adopt increasingly larger main memory and last-level caches (LLC) to accommodate the large working set of various in-memory computing [83, 85], big-data analytics [4, 73], deep learning [82], and server virtualization [4] applications. The memory capacity demand requires further process technology scaling of traditional DRAMs, e.g., to sub-20nm technology nodes [8, 53]. Yet, such aggressive technology scaling imposes substantial challenges in maintaining reliable and high-performance memory system operation due to two main reasons. First, resilience schemes to maintain memory reliability can impose high-performance overhead in future memory systems. Future sub-20nm DRAMs require stronger resilience techniques, such as in-DRAM error correction codes (IECC) [8, 34, 58, 62, 64, 67] and rank-level error correction codes (RECC) [46, 48, 75], compared to current commodity DRAMs. Figure 1 shows that such error correction code (ECC) schemes impose significant performance overhead to the memory hierarchy that employs sub-20nm DRAM as main memory. Second, even with strong resilience schemes, future memory systems do not appear to be as reliable as the state-of-the-art. In fact, due to increased bit error rates (BERs), even when we combine RECC and IECC, sub-20nm-DRAM-based memory systems with 3D-stacked DRAM LLC can have lower reliability than a 28nm-DRAM-based memory system that employs only RECC [8], as seen in Figure 1 (when comparing the triangle and the circle).

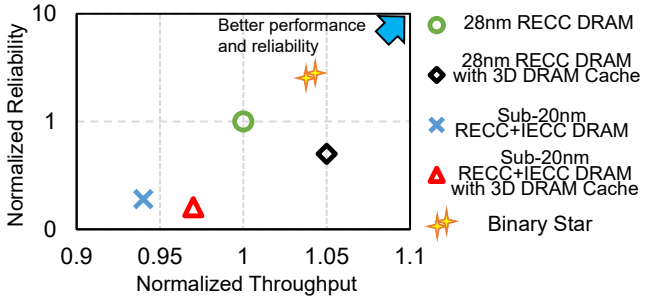



**Figure 1: Reliability and throughput of various LLC and main memory hierarchy configurations normalized to 28nm DRAM with RECC. We define "reliability" as the reciprocal of device failures in time (FIT) [8, 79], i.e., 1/FIT. Reliability data is based on a recent study [8]. Throughput is measured using the benchmarks and system configurations described in Section 5. SRAM caches, which are less critical to scalability are kept constant across systems.**

To address the DRAM technology scaling challenges, next-generation servers will adopt various new memory technologies. For example, Intel's next-generation Xeon servers and Lenovo's ThinkSystem SD650 servers support Optane DC PM [27, 29], which

demonstrates the practical use of large-capacity nonvolatile memories (NVRAMs) in servers. Intel's Knights Landing Xeon Phi server processors integrate up to 16GB multi-channel DRAM (MCDRAM) [26], which is a type of 3D-stacked DRAM,[1] used as the LLC. 3D DRAM and NVRAM promise much larger capacity than commodity SRAM-based caches and DRAM-based main memory, respectively.

However, reliability and performance issues remain. For example, compared with a 28nm DRAM (represented by the circle in Figure 1), a sub-20nm DRAM (the cross in the figure) has both worse performance and reliability. When 3D DRAM LLC is used, the sub-20nm 3D-DRAM-cached DRAM (represented by the triangle in Figure 1) outperforms the sub-20nm DRAM (the cross in the figure), but suffers from degraded reliability. We make similar observations when we compare the 28nm 3D-DRAM-cached DRAM (the diamond in Figure 1) and the 28nm DRAM (the circle in the figure).

Most NVRAM technologies are less vulnerable to soft errors than DRAM [2, 76]. However, many of these technologies have much lower endurance compared to DRAM [42, 74]. As a result, most NVRAM-based main memory needs to adopt hard error protection techniques, imposing both performance and storage overheads compared to DRAM-based main memory [3, 74, 81]. NVRAM also enables persistent memory techniques [30, 66], which allows data structures stored in NVRAM to be recovered after system failures. However, the performance of persistent memory systems is degraded by the high performance and storage overheads of multi-versioning and write-order control mechanisms (e.g., cache flushes and memory barriers) [30, 59, 84].

Our goal in this paper is to achieve both high reliability and high performance, when we scale the capacity of LLC and main memory using new 3D DRAM and NVRAM technologies. To achieve our goal, we propose Binary Star,[2] a coordinated memory hierarchy reliability scheme, which consists of 1) coordinated reliability mechanisms between 3D DRAM LLC and NVRAM (Section 3.1), and 2) NVRAM wear leveling with consistent cache writeback (Section 3.2). The key insight of our design is that traditional memory hierarchy designs typically strive to *separately* optimize the reliability of LLCs and main memory with *decoupled* reliability schemes, yet coordinating reliability schemes across the LLC and main memory enhances the reliability of the *overall* memory hierarchy, while at the same time reducing unnecessary ECC, multiversioning, and ordering control overheads that degrade performance. As illustrated in Figure 1, Binary Star (represented by a double star) achieves much better reliability than state-of-the-art resilience schemes with various LLC and main memory configurations. Even with a slower PCM as the main memory, Binary Star's performance is comparable to the combination of 28nm DRAM main memory and 3D DRAM LLC. We show that Binary Star provides benefits across various types of workloads, including in-memory databases, in-memory key-value stores, online transaction processing, data mining, scientific computation, image processing, and video encoding (Section 6).

[1] We use *3D DRAM* and *3D-stacked DRAM* interchangeably in this paper.

[2] A "binary star" is a star system consisting of two stars orbiting around their common barycenter. Our coordinated memory hierarchy reliability design appears like a star system that consists of two "stars" – a 3D DRAM LLC and an NVRAM main memory.

This paper makes the following key contributions:

- We identify the inefficiency of traditional reliability schemes that are decoupled and uncoordinated across the memory hierarchy (i.e., between the LLC and main memory).
- We propose "Binary Star", the first coordinated reliability scheme across 3D DRAM LLC and NVRAM main memory. Our design leverages consistent cache writeback in NVRAM to recover errors in 3D DRAM LLC. The LLC requires only error detection to perform consistent cache writeback. This significantly reduces the performance and storage overhead of consistent cache writeback.
- We develop a new technique that coordinates our consistent cache writeback technique with NVRAM wear leveling to reduce the performance, storage, and hardware implementation overheads of memory reliability schemes.
- We develop a set of new software-hardware cooperative mechanisms to enable our design. Binary Star is transparent to the applications and thus requires no application code modification.

## 2 BACKGROUND AND MOTIVATION

Existing memory hierarchy designs typically strive to optimize the reliability of individual components (either caches or main memory) in the memory hierarchy [8, 53, 62]. These designs lead to decoupled and uncoordinated reliability across different memory hierarchy levels. As a result, existing designs suffer from high performance overhead and high hardware implementation cost. In this section, we motivate our coordinated reliability mechanism by first discussing the inefficiency of existing reliability schemes for DRAM in Section 2.1, 3D DRAM LLC in Section 2.2, and NVRAM in Section 2.3. Then, we illustrate issues with decoupled, uncoordinated reliability schemes in Section 2.4.

### 2.1 DRAM Main Memory

DRAM has been used as the major technology for implementing main memory. However, DRAM scaling to sub-20nm technology nodes introduces substantial reliability challenges [8, 37, 53, 54].

**DRAM errors.** DRAM errors can be roughly classified into transient errors and hard errors. Transient errors are caused by alpha particles and cosmic rays [24], as well as retention time fluctuations [34, 44, 65, 67, 86]. These transient errors are less likely to repeat and can be avoided after rewriting the corresponding erroneous bits. Hard errors are caused by physical defects or failures [48, 55, 79]. These hard errors repeatedly occur in the same memory cells due to permanent faults.

**Challenges with process scaling.** Process technology scaling can compromise DRAM reliability. Randomly distributed single-cell failure (SCF) is the primary source of device failures [79]. The frequency of SCF increases as DRAM cell size reduces, because as smaller transistors and capacitors are more vulnerable to process variation and manufacturing imperfections [8, 44]. Redundant rows and columns can be used to fix SCFs and keep DRAM device yield high [8]. Recent studies show that in-DRAM ECC (IECC), which integrates ECC engines inside the DRAM device, is a promising method to address DRAM reliability issues [8, 33, 64]. However, IECC leads to significant storage overhead in DRAM chips [8, 58, 62, 64]. Furthermore, IECC needs to be combined with RECC [8],

to achieve reliability close to commodity DRAM [79], imposing substantial performance overhead (See Section 6).

## 2.2 3D DRAM Last Level Cache

Using large 3D DRAM as an LLC can accommodate the increasing working set size of server applications and reduce the performance overhead of capacity scaling. However, the Through Silicon Vias (TSVs) connecting the layers of 3D DRAM can increase the BER by introducing additional defects [32]. Previous studies adopt two-level error correction/detection schemes to improve 3D DRAM reliability [9, 47, 57, 77]. However, these usually require heavy hardware modifications and significant storage overhead.

## 2.3 NVRAM Main Memory

To tackle the scalability issues with DRAM technologies, NVRAM (e.g., PCM [42, 78], STT-RAM [41, 93], and RRAM [6]) introduces a new tier in the memory hierarchy, due to its promising scalable density and cost potential [16, 51, 80]. Many data center server software and hardware suppliers are adopting NVRAMs in their next-generation designs. Examples include Intel's Optane DC PM [29], Microsoft's storage class memory support in Windows OS and in-memory databases [11, 18], Red Hat's persistent memory support in the Linux kernel [52], and Mellanox's persistent memory support over the network fabric [15].

**NVRAM errors.** Similar to DRAM, NVRAM is also susceptible to transient and hard errors. However, the sources of these errors are different from DRAM. Alpha particles and cosmic rays are less of an issue with NVRAM. Instead, NVRAM transient errors are largely caused by resistance drift [76]. In fact, resistance drift can dominate bit errors in MLC NVRAMs (e.g., MLC PCM [2]). Resistance drift is a less critical issue with SLC NVRAMs, as it can be addressed by infrequent scrubbing (e.g., every several seconds) [2, 76]. NVRAM hard errors are often caused by limited endurance. For example, PCM cells can wear out after $10^7$-$10^9$ write cycles [42, 91]. Certain NVRAM technologies, such as certain types of PCM, may have much lower transient BER compared to DRAM before the cells wear out [10, 91].

To mitigate transient and hard errors, previous works propose techniques to reduce MLC NVRAM BER at low performance and storage overheads [2, 76, 90]. SLC NVRAM main memory can be effectively protected using existing ECC mechanisms, as done in state-of-the-art DRAM [10, 76]. In addition to ECC, scrubbing can be used to address NVRAM transient errors (resistance drift) [2, 76]. To mitigate hard errors caused by endurance issues, previous studies adopt wear leveling [42, 69, 94], which scatters NVRAM accesses across the whole device to ensure that all bits in the device wear out in a balanced manner. Hard errors (bits that are already worn out) can be remapped [3, 81], which disables the faulty bits, redirecting accesses to alternative physical locations in the memory.

## 2.4 Issues with Decoupled Reliability

A modern system that employs a large 3D DRAM as LLC and a high-capacity NVRAM as main memory typically utilizes two *separate* uncoordinated reliability schemes for DRAM and NVRAM [3, 21, 77, 81]. The key issue with such decoupled reliability across the LLC and main memory is redundant protection across the memory hierarchy: the same piece of data can be repeatedly (and unnecessarily) protected multiple times. Figure 2 (a) shows an example. A clean cache line-$A$ is protected by ECCs in both 3D DRAM LLC and main memory (represented by $A_c$ and $A_m$, respectively). In fact, as shown in Figure 2 (b) (our methodology, system, and workload configurations are described in Section 5), a large portion of the 3D DRAM LLC lines are clean throughout execution time (after warm-up) with an *identical* copy sitting in main memory, and both copies are protected with ECCs. In this case, carefully enforcing the reliability of only one of the data copies is sufficient. For instance, without ECC in LLC, we can simply invalidate a clean LLC line $A_c$ in case it has a transient error; the error will be naturally recovered by servicing a cache miss to $A$ with an ECC-protected copy $A_m$ from main memory. As such, the LLC only needs to detect whether or not $A_c$ has an error. However, it is challenging to fully exploit the coordination between cache and main memory. For example, what if cache line $A_c$ is dirty? Once $A_c$ has errors, we lose the new value of that cache line because the copy in memory ($A_m$) does not have the up-to-date value. Even if $A_c$ is *not* dirty, $A_c$ with $A_m$ can lead to inconsistent data structures in the program: $A_c$ may belong to a data structure comprising multiple cache lines. If we load the old value $A_m$ to recover the LLC line, the data structure will become inconsistent: the old value of $A_m$ will be inconsistent with new values of other dirty LLC lines that belong to the same data structure. Section 3 discusses our solution to the reliability and consistency problems of both clean and dirty LLC lines.

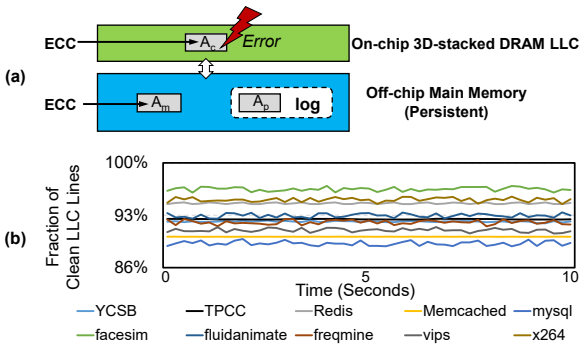



**Figure 2: (a) Multiple reliability mechanisms for a cache line across LLC and main memory. (b) Fraction of clean LLC lines during application execution. We collected data for ten seconds after the benchmark finishes the warm up stage.**

While there are several prior works that handle some SRAM cache and DRAM main memory reliability issues together [36, 87, 88], these are not desirable for the combination of 3D DRAM and NVRAM. On the one hand, these works are hard to scale to the 3D DRAM size. On the other hand, because these designs require frequently clearing dirty cache lines [36, 87] or off-loading SRAM cache ECC to the main memory [87, 88], these mechanisms further consume NVRAM's short endurance and low bandwidth.

Instead of coordinating the reliability schemes of caches and main memory, persistent memory designs enhance the reliability of the main memory when a system completely loses data in caches due to a system crash or power failure [38, 92]. Persistent memory allows data in NVRAM to be accessed via load/store instructions like traditional main memory, yet recoverable across system reboots [12,

49]. As shown in Figure 2 (a), persistent memory systems typically maintain an additional copy of data (e.g., $A_p$ that can be a log entry or a checkpoint), which is used to recover original data ($A_m$) in NVRAM main memory if needed after a system crash. This process can be done through controlled data versioning (e.g., by logging [84] or shadow paging [12]), or via write-ordering (e.g., by cache flushes and memory barriers [12]). However, relying on persistent memory mechanisms for error recovery purposes has at least two major disadvantages: first, all these techniques require code modifications, which require significant software engineering effort [70]; second, applications with no persistence requirements suffer from substantial performance and storage overheads in main memory access [30, 70, 84].

In summary, we need to address the overhead and scalability issues with (i) the decoupled reliability schemes across caches and main memory in state-of-the-art memory systems that combine 3D DRAM and NVRAM, and (ii) the challenges of exploiting persistent memory techniques in order to efficiently maintain the reliability of memory hierarchy. **Our goal** in this paper is to design a comprehensive coordinated memory hierarchy reliability scheme that is efficient and effective for memory hierarchies that combine multiple technologies.

## 3 BINARY STAR DESIGN

To address the aforementioned reliability challenges, we propose Binary Star. Binary Star 1) coordinates reliability schemes across 3D DRAM LLC and NVRAM main memory, and 2) uses an efficient hardware-based 3D DRAM LLC error correction and a software-based error recovery mechanism.

3D DRAM LLC and NVRAM technologies allow the memory hierarchy to continue to scale in a cost-efficient manner. Figure 3 shows our basic architectural configuration. We assume that higher-level caches are SRAM. The LLC in the processor is a 3D DRAM. We use SLC PCM, which is a well-understood NVRAM technology, as a representative for NVRAM main memory to make our design and evaluation concrete. Our design principles can be applied to various NVRAM technologies. In this section, we describe our design principles and mechanisms, using Figure 4 to illustrate them. We leave the description of the implementation of our mechanisms to Section 4.

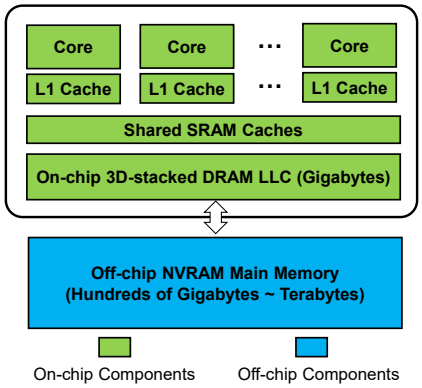



**Figure 3: Basic architecture configuration.**

### 3.1 Coordinated Reliability Scheme

Our coordinated reliability scheme strives to optimize the reliability of the 3D DRAM LLC and NVRAM hierarchy as a whole, instead of separately optimizing the reliability of individual components at different hierarchy levels, at low performance and storage cost. Our scheme is enabled by the following key observation:

***Observation 1:*** *Errors in the LLC can be corrected by consistent data copies in NVRAM main memory.*

The data stored in the LLC is a subset of the data in main memory. Therefore, as long as the main memory maintains a consistent data copy at a known point in time, errors in the corresponding LLC line can be corrected (or ignored) using the consistent data copy in memory. This observation makes error correction codes in the LLC unnecessary. Instead, Binary Star coordinates 3D DRAM LLC and NVRAM reliability in the following manner: the LLC adopts only cyclic redundancy check (CRC) codes [39] to ensure errors can be detected; NVRAM maintains consistent data copies that can be used to correct (or avoid) the detected errors in the LLC. While it is straightforward to implement CRC in the LLC, maintaining consistent data in NVRAM for error recovery purposes is complicated for two reasons. First, storing and updating multiple versions of data can incur significant storage and performance overhead [70, 92]. Second, LLC writebacks can corrupt consistent versions of data in NVRAM: For example, a data structure update that spans multiple cache lines can result in multiple dirty cache lines; yet only one or several of the dirty cache lines may be written back to NVRAM main memory at a given time, leaving the state of the data structure in main memory inconsistent.

We need to avoid such inconsistency of data in NVRAM main memory, in order to use the main memory contents to correct errors that happen in LLC lines. To this end, we propose *periodic forced writeback* and *consistent cache writeback* (depicted in Figure 4 (a) and (b), respectively).

**Periodic forced writeback.** During the execution of an application, portions of updated data might remain in the dirty cache lines, causing inconsistent data to be present in the NVRAM. Because dirty cache lines contain all the most recent updates, Binary Star periodically forces the writeback of dirty cache lines from all cache levels into the NVRAM main memory (e.g., cache lines *a* and *b* in Figure 4 (a)). These written-back data along with previously written-back dirty cache lines (e.g, cache line *c* in the figure.) generate a set of consistent data, i.e., a *checkpoint of data*, in the NVRAM by ensuring that all updates to data are propagated to the NVRAM. When taking the checkpoint, Binary Star marks all updated NVRAM blocks as consistent (i.e., the striped squares in Figure 4 (a) turn into solid squares after the checkpoint is taken). To recover to this consistent checkpointed state in the presence of LLC errors, Binary Star maintains a single checkpoint and saves the state of an affected process to ensure that the process is able to recover to the point *immediately after* the periodic forced writeback. The checkpointed consistent data in NVRAM stays intact in NVRAM until the end of the next forced writeback period. This checkpointed consistent data in NVRAM can be used to recover from LLC errors detected by the CRC between two consecutive forced writeback periods. In our evaluation, we perform detailed sensitivity studies (Section 6.3) and

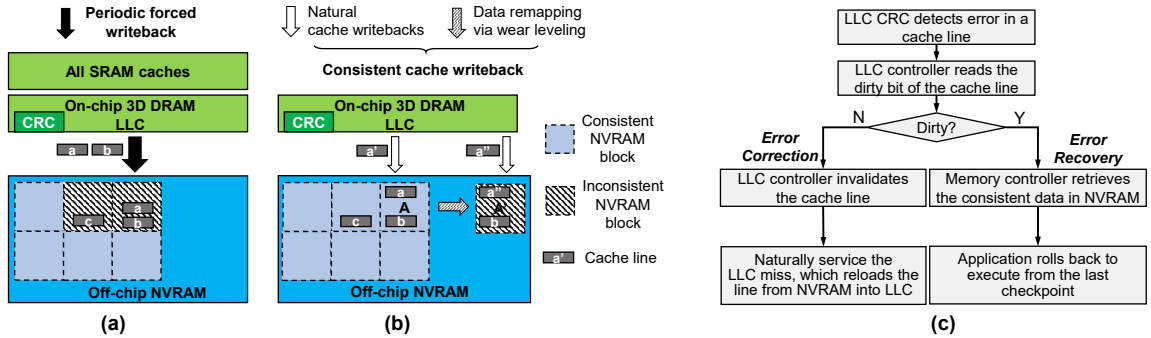



**Figure 4: Binary Star design overview. (a) Periodic forced writeback and (b) Consistent cache writeback of our Coordinated Reliability scheme. (c) Binary Star error correction and error recovery mechanisms. A solid filled square represents a checkpointed consistent data block, while a striped filled square represents a remapped inconsistent data block.**

demonstrate that 30 minutes is the best periodic forced writeback interval for the Binary Star system with our benchmarks.

**Consistent cache writeback.** To maintain consistency of the checkpointed data *between two forced writeback periods*, we propose a consistent cache writeback mechanism as shown in Figure 4 (b). This mechanism redirects natural LLC writebacks to NVRAM locations that are different from the locations of the checkpointed data, so that checkpointed data stays consistent.

**Application Transparency.** Maintaining data consistency through persistent memory typically requires support for (i) a programming interface (e.g., transactional interface), (ii) multi-versioning (e.g., logging or copy-on-write), and (iii) write-order control (e.g., cache flushes and memory barriers). These can impose non-trivial performance and implementation overheads on top of a traditional memory hierarchy [59, 92]. Furthermore, persistent memory systems need to redirect data upon each persistent data commit [59, 66, 84]. Our design does not impose large overheads due to three major reasons. First, the granularity of our consistent cache writeback mechanism is only one cache line. Therefore, we do not require a transactional interface to the application to determine the commit granularity. Second, our consistent cache writeback mechanism is used to update the application's working data, rather than the checkpointed consistent data. Therefore, we do not need to enforce write-order control of consistent cache writeback. Finally, our design only needs to ensure that natural LLC writebacks do not overwrite the original, i.e., checkpointed copy of consistent data, which is reserved to be used to correct errors in LLC. As such, we do *not* redirect each natural LLC writeback to a new NVRAM location. Rather, in our technique, the LLC writebacks to the same data block can repeatedly *overwrite* the same NVRAM block[3] that is outside of the checkpointed consistent data. As shown in Figure 4 (b), the up-to-date version (e.g., cache lines $a''$ and $b$) in NVRAM is visible to applications, while the checkpointed consistent version (e.g., the original data $a$) is hidden from applications.

[3]A block is the minimal granularity of remapping and data consistency ensured in the NVRAM in our design. The block size is larger than a cache line and depends on the granularity of wear leveling mechanism used.

## 3.2 Coordinating Wear Leveling and Consistent Cache Writeback

Our consistent cache writeback mechanism requires redirecting a natural LLC writeback to an NVRAM location that is different from the location of the checkpointed data. However, doing so with a traditional memory system design would introduce extra effort to (i) maintain address remapping with metadata and free data block management and (ii) manage alternative memory regions to store redirected natural LLC writebacks. These can impose substantial performance and NVRAM storage overheads. Instead of explicitly maintaining data consistency for each cache writeback, Binary Star leverages the remapping techniques that are already employed by existing NVRAM wear leveling mechanisms. Our method redirects the natural LLC writebacks to memory locations different from the checkpointed data locations during consistent cache writeback. We make the following key observation:

***Observation 2:*** *NVRAM wear leveling naturally redirects and maintains the remapping of data updates to alternative memory locations.*

Note that it is difficult to leverage NVRAM wear leveling with regular persistent data commits on a system with persistent memory. To provide crash consistency, persistent memory systems typically require each data commit to be redirected to a new location, e.g., a new log entry [84], a newly allocated shadow copy [12], a new checkpoint [70], or a new version managed by hardware [70, 92]. However, NVRAM wear leveling redirects data updates *only when* the original physical memory location is repeatedly overwritten for a given number of times [94]. Therefore, the remapping of data typically happens infrequently. If we would like to exploit wear leveling to provide consistency in persistent memory, wear leveling needs to be synchronized with each persistent data commit. This likely imposes a substantial performance overhead because it leads to prohibitively frequent data remapping and metadata updates.

In contrast, Binary Star's consistent cache writeback mechanism can effectively take advantage of existing NVRAM wear leveling mechanisms. During the interval between two consecutive occurrences of periodic forced writeback, Binary Star's consistent cache writeback mechanism needs to redirect NVRAM updates of each data block only once. After the first update to the consistent data block is redirected due to a natural LLC writeback (e.g., the update of cache line $a'$, which is overwritten by the later update $a''$, as shown

in Figure 4 (b)), subsequent natural LLC writebacks of the same data block can repeatedly overwrite the same redirected data location (e.g., the updated version $a''$, as shown in Figure 4 (b)). Therefore, Binary Star does *not* need to synchronize wear leveling with each consistent cache writeback. Instead, Binary Star must trigger one single extra *data remapping* via the wear leveling mechanism at only the *first* update to a consistent data block; wear leveling can operate as is during the rest of the execution.

By coordinating wear leveling with consistent data writeback, Binary Star enables better utilization of wear leveling. Traditional NVRAM wear leveling remaps data of frequently accessed NVRAM blocks to infrequently accessed blocks [94], without being explicitly aware of the consistency of data. Our design exploits this *remapping* capability by exposing data consistency to the wear leveling, such that we can guarantee data consistency by using the data remapping capability of wear leveling (as illustrated in Figure 4 (b)). In particular, we coordinate NVRAM wear leveling with our consistent cache writeback mechanism in the following manner:

- *Binary Star-triggered wear leveling*: When the memory controller receives an LLC writeback (either a natural LLC writeback or a periodic forced writeback) to a checkpointed consistent data block, Binary Star actively triggers wear leveling to remap the LLC writeback to another NVRAM block. Such remapped updates, along with the original blocks in the current checkpointed consistent data, form the next checkpoint of consistent data. As a result, Binary Star ensures that the *current* version of the checkpointed consistent data block remains intact.
- *Natural wear leveling*: During the rest of the time, wear leveling can operate as is. Note that Binary Star does *not* prevent natural wear leveling from remapping the checkpointed consistent data blocks. In case a checkpointed consistent data block is remapped, Binary Star simply updates the metadata to reflect the new location of the block (Section 4).
- *NVRAM updates without remapping*: NVRAM wear leveling is performed periodically [94]. During the interval when wear leveling is not performed, natural LLC writebacks to the blocks outside of the checkpointed consistent data blocks can directly overwrite the same location.

## 3.3 3D DRAM LLC Error Correction and Error Recovery

Our 3D DRAM LLC uses CRC codes that can only detect errors. Binary Star recovers from them using the copies of cache lines that are in the NVRAM main memory. Based on our coordinated reliability scheme, there can be two types of data copies in NVRAM for a given block: 1) the checkpointed consistent copy, which is generated by periodic forced writeback and which contains the consistent data, and 2) the latest updated copy that is managed by Binary Star-triggered wear leveling. Once CRC detects an error in a given 3D DRAM LLC cache line during an access to the 3D DRAM LLC, Binary Star performs error correction and recovery via the following steps that we also illustrate in Figure 4 (c): First, the LLC controller reads the dirty bit of the erroneous cache line.

If the LLC line is clean, Binary Star performs error correction in hardware as follows: 1) the LLC controller invalidates the cache line, 2) the corresponding LLC access turns into an LLC miss, which is serviced as a natural LLC miss without any hardware modifications. The LLC miss generates an access to the corresponding most recently updated data block in the NVRAM. The corresponding data block could reside in either the checkpointed consistent data block or the remapped data block.

If the LLC line is dirty, Binary Star determines that it is an uncorrectable error. If such an error is detected, Binary Star triggers error recovery through the system software by 1) reverting the memory state to the checkpointed consistent version of application data, and 2) rolling back the application to execute from the checkpoint.

## 3.4 Putting it All Together: An Example

With these design components, we demonstrate a simple example to show the detailed operation of Binary Star when running an unmodified application. Figure 5 depicts an example of four cases (ⓐ to ⓓ) that a running application may encounter when our proposed mechanisms are employed. These four cases are software and hardware operations that conduct consistent cache writeback (ⓐ) and periodic forced writeback (ⓑ) during normal execution, as well as handling of correctable errors (ⓒ) and handling of uncorrectable errors (ⓓ), when an error is detected.

- Binary Star hardware conducts consistent cache writeback during normal application execution (ⓐ). Consistent cache writeback is transparent and asynchronous to the application. When the LLC has to evict a cache line, the memory controller identifies if the cache line belongs to a consistent block (❶). The cache line is directly written into NVRAM if it belongs to an inconsistent block, otherwise the wear leveling remaps the consistent block to a new block and writes the cache line into it (❷). Section 4.2 provides the details.
- **Binary Star daemon** conducts periodic forced writeback during normal execution (ⓑ). The daemon stalls the application (❶), saves the process state, and executes the **Binary Star cache line writeback (*bclwb*)** instruction, which enables the memory controller to write back all dirty cache lines to main memory without invalidation and mark the corresponding NVRAM data blocks as consistent (❷). This procedure creates a new checkpoint and invalidates the previous one. The daemon resumes application execution once the periodic forced writeback finishes (❸).
- A correctable error (ⓒ), i.e., an error in a clean LLC line, is transparent to the software. Hardware detects and corrects these errors: **memory controller** detects the error using CRC (❶), reads the dirty bit and identifies that this is a correctable error (❷), invalidates the LLC line (❸), and issues a cache miss to retrieve the correct copy of the cache line from NVRAM (❹).
- An uncorrectable error, i.e., an error in a dirty LLC line, requires software and hardware operations to recover the LLC line (ⓓ). **LLC controller** detects the error using CRC (❶), reads the dirty bit and identifies that this is an uncorrectable error (❷), and sends a signal to the Binary Star daemon (❸). The daemon stalls the application and triggers the rollback procedure (❹). The daemon uses the **Binary Star data reset (*drst*)** instruction to reclaim the previously checkpointed consistent data blocks (i.e., the checkpoint) in NVRAM (❺). The whole procedure rolls the memory hierarchy back to the consistent checkpointed state and resumes normal application execution (❻).

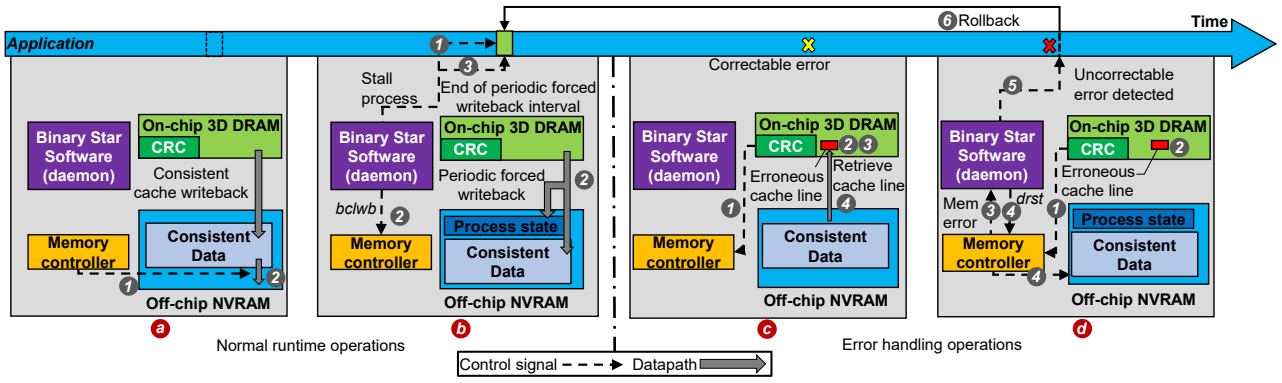



**Figure 5: An example of running an application on Binary Star.**

## 4 IMPLEMENTATION

This section discusses our Binary Star implementation, including modifications we make to the 3D DRAM LLC, the memory controller and the system software.

### 4.1 3D DRAM LLC Modification

We replace the ECC in 3D DRAM LLC with a 32-bit CRC (CRC-32) [39] for each 64B LLC line. This design allows Binary Star to detect up to eight-bit errors as well as a portion of nine or more bit errors, and all burst errors up to 32 bits long [39]. Binary Star also modifies the finite state machine in the LLC controller to support the steps required to handle LLC error detection, correction, and rollback as listed in Section 3.3.

### 4.2 Memory Controller Modifications

Binary Star modifies the state-of-the-art area- and performance-efficient Segment Swap design [94] for NVRAM wear leveling. Segment Swap counts the number of writes to each segment, and periodically swaps a hot segment with a cold segment.[4]

**Modifications to the wear leveling mechanism.** To coordinate the NVRAM wear leveling mechanism with our consistent cache writeback mechanism, we modify Segment Swap [94] design as shown in Figure 6. In the original design, the mapping table maps a "virtual" segment to a "physical" segment. In our design, we maintain a set of metadata with each segment that includes 1) a *consistent bit* that indicates whether the segment belongs to the consistent checkpoint, 2) a *free bit* that indicates whether the segment is free, and 3) a *segment number* that identifies the segment that stores the corresponding checkpointed consistent data. The metadata is stored in a reserved NVRAM space similar to previous works [69, 94]. During periodic forced writeback, the memory controller uses the segment numbers to release each segment belonging to the previous checkpointed consistent data (e.g., segment *A* ❶ in Figure 6) by resetting the *free bit* and adding them to the free list. After all dirty cache lines are written back, the memory controller also marks the current inconsistent segment as consistent (segment *A'* ❷) and deletes the stored segment number. During consistent cache writeback, NVRAM wear leveling performs remapping by selecting a free segment, copying data from the consistent segment

[4]Segment is the granularity of wear leveling. In this paper, we set the segment size to be one Mbyte, similarly to the original design [94].

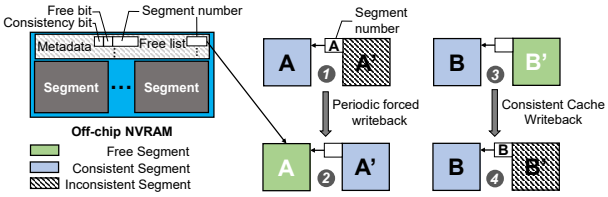



**Figure 6: Modifications to NVRAM wear leveling.**

(segment *B* ❸) to it, and marking it as an inconsistent segment (segment *B'* ❹).

**Modification to NVRAM write control.** We modify the memory controller's wear leveling control logic to enable our consistent cache writeback mechanism (Section 3.2). When the memory controller performs a cache writeback, the wear leveling control logic reads the *consistent bit* in the metadata space of NVRAM to determine the consistency state of a segment that the cache line belongs to. If the cache line belongs to an inconsistent segment, the controller directly writes data into NVRAM. If the cache line belongs to a consistent segment, the segment is remapped via the wear leveling mechanism. In this case, the controller selects a free segment from the free list. The free list keeps track of all the free segments' numbers and is stored in the memory controller. The writeback data, along with the original data in the old segment, is moved to the new free segment. The memory controller also changes the entry in the mapping table to the new segment [94].

**New instructions.** To coordinate between the memory controller and the system software, we add two instructions to 1) support periodic forced writeback, and 2) retrieve and roll back to checkpointed consistent data.

The first instruction is called the *Binary Star cache line writeback instruction* (*bclwb*). Binary Star leverages the *bclwb* instruction to write back all dirty cache lines. Similar to an existing cache line writeback instruction (*clwb*) [38], *bclwb* allows Binary Star to write back each dirty cache line to NVRAM without invalidating the cache line. The *bclwb* instruction not only performs the same cache line writeback functionality as *clwb*, but also triggers the memory controller to modify wear leveling metadata to guarantee consistency. During the execution of a *bclwb* instruction, after data writeback, the memory controller marks the current segment as consistent. The memory controller also tracks the old segment with

the segment number, frees it by setting the free bit, and adding it to the free list.

The second instruction is called the *Binary Star data reset instruction* (*drst*) and is used to retrieve checkpointed consistent data. *drst* has only one operand: memory address. *drst* is used for handling error-triggered rollback. When the memory controller receives a *drst* command, the memory controller checks the *consistent bit* of the associated segment's memory address to see whether or not the segment is consistent. The memory controller does nothing if the segment is consistent. Otherwise, the memory controller finds the corresponding consistent segment using the stored consistent segment number. The memory controller then invalidates the current segment by setting the free bit, and adding the segment number to the free list.

**Reducing latency by controlling NVRAM writebacks.** Because the latency of Binary Star-triggered wear leveling can degrade the performance of a latency-critical application, Binary Star provides two optional optimizations. First, when the NVRAM bus is idle, Binary Star allows the memory controller to issue preemptive consistent cache writeback to remap a consistent segment in advance. Second, when read requests are delayed by the writebacks, Binary Star allows the memory controller to postpone LLC writebacks, which has been shown to provide performance benefits in a previous study [68].

**NVRAM over-provisioning.** NVRAM preserves a certain amount of physical space specifically for wear leveling purposes [28]. Over-provisioned space guarantees that wear leveling can always find a free segment when the logical space is full. To ensure that Binary Star-triggered wear leveling always finds a free segment, this over-provisioned space has to be large enough to provide a sufficient number of free segments. Our experiments (Section 3.2) show that 10% extra space for over-provisioning is sufficient to accommodate a 30-minute periodic forced writeback interval with all of our benchmarks.

## 4.3 System Software Modifications

We implement software support as part of the operating system to leverage the hardware support.

**Binary Star daemon.** We develop a Binary Star daemon, which is in charge of periodic forced writeback and rollback. Because the periodic forced writeback and rollback are infrequent operations, we assume a single daemon can manage all processes on a Binary Star enabled machine. Figure 7 shows the state machine of the daemon. Binary Star daemon can be implemented as a kernel loadable module.

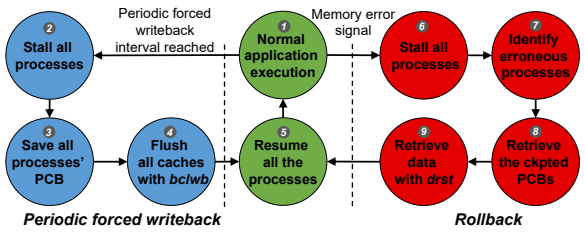



**Figure 7: State machine of the Binary Star software daemon.**

The Binary Star daemon operates in normal mode until either 1) the periodic forced writeback interval is reached, or 2) a memory error signal is received. To determine if the periodic forced writeback interval is reached, we use a simple timer, which consumes minimal CPU resources (❶). When the timer reaches a pre-defined time interval, the daemon sends interrupt signals to stall all the running processes and starts the periodic forced writeback procedure.

The periodic forced writeback procedure consists of five steps. First, the daemon sends an interrupt to each of the executing processes and stalls them (❷). Second, a copy of each process control block (PCB) is flushed into the memory, so that the process can be restored (❸). Each PCB contains process-related state, such as process ID, stack, and page table. Instead of introducing extra implementation cost, Binary Star leverages the PCB state to roll-back a process to the previously written-back checkpoint in the same way as existing designs [60]. Third, the daemon flushes all the dirty caches lines (including SRAM and 3D DRAM caches) to the memory, by using the *bclwb* instruction (❹). Fourth, when all cache line writebacks are completed, the daemon resumes all the stalled processes (❺).

The second key task of the Binary Star daemon is to manage error-triggered rollback. When the daemon receives an unrecoverable memory error signal from the memory controller, it starts the rollback procedure. First, the daemon sends an interrupt to stall all executing processes (❻). Second, the daemon identifies the error-affected processes, inspects page tables of each process and identifies the process that triggered the error (❼). Third, the daemon retrieves all the PCB data, such that the affected processes' PCBs are rolled back to the checkpointed PCBs (❽). Fourth, the daemon retrieves the data of all error-affected processes (❾). The daemon rolls back memory data by retrieving the old page table from the PCB and using the *drst* instruction to fetch previously checkpointed consistent data. The daemon also employs *drst* to free all the remapped inconsistent segments to avoid memory leakage. Finally, the daemon resumes the execution of all the processes (❺).

## 4.4 Binary Star Overheads

**Overhead in 3D DRAM Cache.** The CRC-32 code requires 32 bits per 64-byte cache line, i.e., 6.25% storage overhead, which is 50% lower than the storage overhead of traditional ECC. The area and performance overheads of CRC encoding and decoding engines are similar to the existing ECC encoding and decoding engines.

**Overhead in NVRAM.** Our modification to NVRAM wear leveling requires three bytes per segment. With a 512GB NVRAM and a 1MB segment size, this introduces 0.0003% storage overhead. The free segment list takes 1.1875MB additional space at most. Because NVRAM typically over-provisions its capacity by roughly 10% to support wear leveling [28], our design reuses the over-provisioning space and, as such, avoids additional storage overhead.

**Software overhead.** For software support, Binary Star only requires a loadable module to the operating system. An application requires no modification to utilize Binary Star.

**Overhead of periodic forced writeback and uncorrectable error-triggered rollback.** We test periodic forced writeback and rollback using several workloads on an emulated NVRAM. Typically, the periodic forced writeback takes 50$\mu$s to 3.6 seconds. Because Binary Star rollback reuses data hidden by wear leveling and

avoids large amounts of data movement, the rollback takes only dozens of microseconds. The Binary Star daemon has negligible performance impact on the operating system and applications because of the low frequency of periodic forced writeback and the very small likelihood of error-triggered rollback (See Section 6.3).

## 5 EXPERIMENTAL METHODOLOGY

We evaluate both the performance and reliability of Binary Star by comparing it with four baseline memory configurations: 1) DRAM memory with 3D DRAM LLC (**3D DRAM+DRAM**), 2) DRAM memory without 3D DRAM LLC (**DRAM**), 3) PCM as main memory with 3D DRAM LLC (**3D DRAM+PCM**), and 4) PCM as main memory without 3D DRAM LLC (**PCM**).

For all four memory configurations, we use various state-of-the-art ECC mechanisms, which we describe below. With DRAM, we evaluate both 28nm and sub-20nm process technologies. We use Failures In Time (FIT) as our main reliability metric.

**Performance simulation.** We evaluate the performance of Binary Star modifications using McSimA+ [1], which is a Pin-based [45] cycle-level multi-core simulator. Our infrastructure models the row buffer as well as row buffer hits and row buffer misses. We use the FR-FCFS memory scheduling policy [71, 95] for Binary Star and all our baselines. We configure the simulator to model a multi-core out-of-order processor with 3D DRAM LLC and NVRAM DIMM (for Binary Star) as described in Table 1. The higher-level L1 and L2 caches are SRAM-based. We model the 3D DRAM LLC based on the HMC 2.1 specification [50]. We use PCM timing parameters [42] in our basic configuration for the NVRAM DIMM. We adopt the state-of-the-art NVRAM error protection design [81] in our evaluations.
**Reliability simulation and modeling.** We evaluate the reliability of our design using a combination of reliability simulation and theoretical calculation. We employ FaultSim [56], a configurable memory resilience simulator, to compare the reliability and storage overhead of various memory technologies and reliability schemes.

To evaluate the error correction and recovery ability of Binary Star, we calculate the device failure rates using a method proposed in previous work [8]. A device failure occurs when one or more erroneous bits within any 64-bit line of a ×8 device are uncorrectable after the corresponding ECC or correction/recovery mechanisms are applied. We calculate the device failure rate based on the error correction/recovery capability (the number of bits that can be corrected) of each mechanism given the number and distribution of single cell errors [8]. We assume that single cell errors are randomly distributed. We omit column, row, and connection faults for the same reasons discussed in previous studies [8]. We only consider errors that are found after device shipment time [8]. We evaluate various reliability schemes, including no-ECC, RECC only [75], combined RECC (rank-level ECC) and IECC (in-DRAM ECC) [8], combined Chipkill and IECC [8, 58], and Binary Star (which combines CRC-32 and our error correction and recovery techniques described in Section 4). For both reliability simulation and theoretical analysis, we adopt PCM and sub-20nm DRAM device reliability parameters published in previous works [8, 10, 56, 79]. We implement 3D DRAM Chipkill with multiple DRAM cubes. For a fair comparison, we use multiple DRAM cubes across all the reliability experiments. However, we note that Binary Star is applicable to both single-cube and multiple-cube 3D DRAM. Because 3D DRAM does not have ranks, RECC on 3D DRAM is the same as ECC on a single-cube as in various previous works [31, 57].

**Table 1: Evaluated processor and memory configurations.**

| | |
|---|---|
| **Processor** | |
| Cores | 4 cores, 3.2 GHz, 2 threads/core, 22nm |
| IL1 Cache | 32KB per core, 8-way, 64B cache lines, 1.6ns latency |
| DL1 Cache | 32KB per core, 8-way, 64B cache lines, 1.6ns latency |
| L2 Cache | 32MB shared, 16-way, 64B cache lines, 4.4ns latency |
| LLC | 3D DRAM, 16 GB, CRC-32 (for Binary Star), tCAS-tRCD-tRP: 8-8-8, tRAS-tRC: 26-34 [50, 77] |
| **Main Memory** | |
| DRAM | 512 GB DDR4-2133 timing, 2 128-bit channels, ***ECC schemes***: non-ECC, RECC [75], combined RECC and IECC [8], combined Chipkill and IECC [8, 58] |
| PCM | 512 GB and 52GB over-provisioning space, 36ns row-buffer hit, 100/300ns read/write row-buffer conflict [42], 2 channels, 128 bits per channel, Single error correction, double error detection (SECDED) + NVRAM remap [81] |

**Benchmarks.** We evaluate our design with various data-intensive workloads. Redis [7] and Memcached [19] are in-memory key-value store systems widely used in web applications. Our Redis benchmark has one million keys in total and simulates the case when Redis is used as an LRU cache; Our Memcached benchmark has 100K ops (read/write), with 5% write/update operations. TPC-C [14] is a popular on-line transaction processing (OLTP) benchmark; we run 400K transactions, mix of reads and updates (40%). YCSB [13] is an open-source specification and program suite for evaluating retrieval and maintenance capabilities of computer programs; we run eight million transactions, randomly performing inserts, updates, reads, and scans. MySQL [63] is one of the most widely used relational database management systems (RDBMSs), which is typically used in online transaction processing; we use a table of 10 million tuples and conduct complex OLTP operations (using sysbench [40]), including point, range, update, delete, and insert queries. We exercise these server workloads using four clients and scale the memory footprint. We also evaluate Binary Star with several memory-intensive benchmarks from the PARSEC 3.0 benchmark suite [5], including *facesim*, *freqmine*, *fluidanimate*, *vips*, and *x264*. We configure the dataset to be the same as the capacity of our main memory configuration (512GB) on all the benchmarks.

We evaluate application performance using instruction throughput (instructions per cycle) for PARSEC 3.0 [5] benchmarks and operational throughput (the number of executed application-level operations, e.g., inserting or deleting data structure nodes, per second, not including logging) for all other benchmarks.

## 6 RESULTS

This section presents the results of our Binary Star evaluation. Section 6.1 shows Binary Star improves reliability over various error correction baselines. Section 6.2 shows the performance of Binary Star compared to multiple memory configurations. Section 6.3 presents a study of the impact of Binary Star's periodic forced writeback interval on various metrics.

## 6.1 Reliability

Figure 8 shows the projected 3D DRAM device failure rates of different resilience schemes as the single cell bit error rate varies from $10^{-14}$ to $10^{-4}$. The x-axis represents the bit error rate (BER) of a single cell and the y-axis is the corresponding device failure rate. We make three major observations. First, for sub-20nm DRAM with $10^{-5}$ single cell bit error rate [8], Binary Star reduces the device failure rate by $10^{12}$ times compared to IECC with Chipkill [8] and by $10^{16}$ times compared to IECC with RECC [8]. Second, even when the sub-20nm single cell error rate is lower (i.e., $10^{-6}$), Binary Star still provides better reliability than other schemes: Binary Star reduces the device failure rate by $10^{16}$ times compared to IECC with Chipkill, and by $10^{22}$ times compared to IECC with RECC. These results indicate that Binary Star is effective for technologies that exhibit lower error rates than those projected in [8]. Third, compared to the commonly-employed RECC DRAM, Binary Star greatly lowers the device failure rate on both 28nm and sub-20nm technology nodes (i.e., at $10^{-10}$ and $10^{-15}$ single cell BERs). We conclude that Binary Star provides strong error protection on 3D DRAM LLCs, regardless of technology node and single cell error rate.

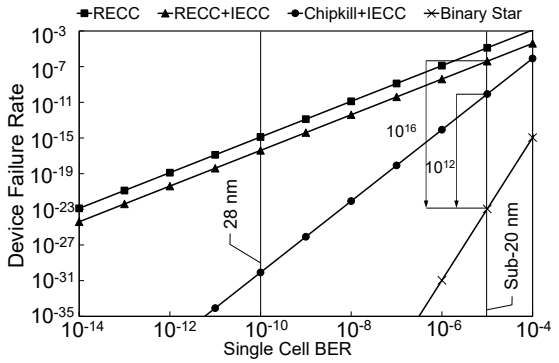



**Figure 8: Device failure rates of 3D DRAM for traditional RECC [81], RECC with IECC [8], Binary Star, and Chipkill with IECC [8, 58] vs. single cell bit error rate. Vertical lines represent 3D DRAM technology nodes (28nm and sub-20nm).**[5]

Table 2 summarizes the FIT and storage cost of five state-of-the-art baselines and Binary Star. We show results when the device single cell BER is $10^{-6}$ (left) and $10^{-5}$ (right). We make three observations. First, Binary Star provides a clear advantage against all state-of-the-art baselines. Compared to IECC+Chipkill, which is the best IECC based scheme, Binary Star reduces FIT by 14.4×. Compared to the IECC+RECC protected sub-20nm DRAM system, Binary Star reduces FIT by 29.7×. Second, compared to the NVRAM based main memory system that uses RECC 3D DRAM cache and PCM, Binary Star experiences 58.5% fewer failures. Third, Binary Star incurs lower space overhead compared to all other state-of-the-art designs, with only 6.25% additional space in 3D DRAM and only 12.79% additional space in PCM, respectively. We conclude that Binary Star is the most efficient reliability mechanism for hybrid emerging technologies of 3D DRAM cache and NVRAM main memory.

[5]The sub-20nm single bit error rate is based on [8]. The single cell error rates of future sub-20nm products can vary and depend on the specific technologies.

**Table 2: Comparison of the evaluated resilience schemes.**

| System | FIT | Storage cost | |
|---|---|---|---|
| | | DRAM LLC | Main memory |
| No-ECC [79] 28nm DRAM | 44032-66150 | N/A | 0% |
| RECC [72] 28nm DRAM | 8806-13230 | N/A | 12.5% |
| IECC+RECC [8] sub-20nm 3D DRAM + DRAM | 78211-117912 | 18.75% | 18.75% |
| IECC+Chipkill [8, 58] sub-20nm 3D DRAM + DRAM | 37949-59518 | 18.75% | 18.75% |
| RECC [72, 81] sub-20nm 3D DRAM + PCM | 6352-9963 | 12.5% | 12.79% |
| **Binary Star** sub-20nm 3D DRAM + PCM | 2637-3968 | 6.25% | 12.79% |

## 6.2 Performance

Figure 9 shows the performance of three baseline memory configurations, all of which have *no ECC*, and Binary Star normalized to the baseline with 3D DRAM cache and DRAM main memory.[6] We make two observations based on this data. First, Binary Star's performance is very close to the baseline with 3D DRAM. Binary Star performs within 0.2% of a baseline with 3D DRAM and PCM Binary Star also performs within 1% of the performance of the baseline with 3D DRAM cache and DRAM main memory. Second, we observe that using 3D DRAM as the LLC to PCM main memory improves performance by 10% over the PCM-only baseline. We observe that the performance improvements are significant on workloads that exhibit high LLC locality (e.g., memcached, facesim, vips and x264). In summary, we find that Binary Star provides comparable performance to the baseline with 3D DRAM cache and DRAM main memory but at significantly higher reliability.

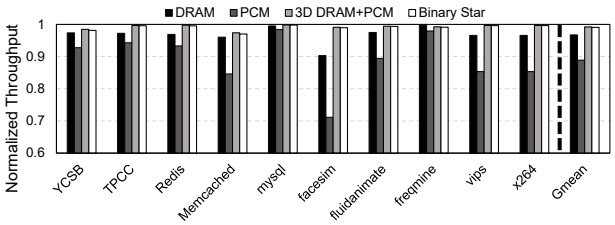



**Figure 9: System performance of different memory configurations with no ECC and Binary Star. We normalize the throughput of each configuration to the throughput of 3D DRAM LLC with DRAM.**

**Effect of NVRAM wear-out.** To understand the effect of NVRAM wear-out-induced latency on performance, Figure 10 provides the performance of each workload with different NVRAM access latencies. In this experiment, we vary the latency of PCM to 0.5×, 1×, 2× and 4× of the default PCM latency provided in Table 1. All throughput values are normalized to the highest performance memory configuration (3D DRAM+DRAM). As Figure 10 shows, Binary Star retains 99% of the performance of the baseline with a 0.5×

[6]We include the CRC encoding and decoding performance overheads based on [8]. We normalize the throughput of each configuration to the throughput of 3D DRAM LLC with DRAM. We find that the error detection and recovery overheads are negligible compared to the CRC encoding and decoding overheads, because errors happen rarely (See Section 6.1).

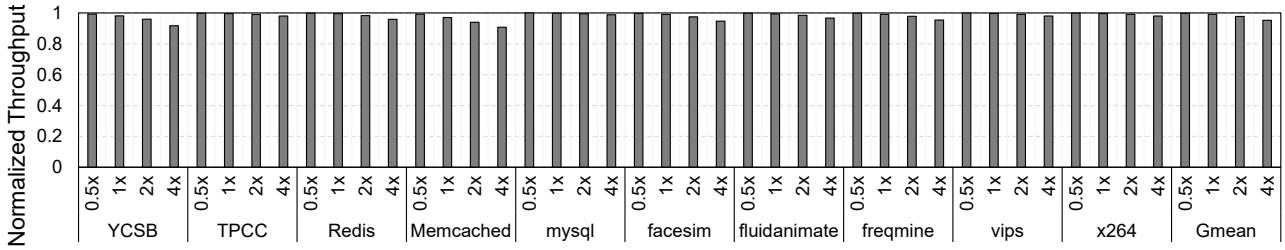



**Figure 10: Performance of Binary Star normalized to the performance of the 3D DRAM+DRAM configuration, as PCM latency is varied as a multiple of its value in Table 1.**

PCM latency and its average performance degradation is only 3% with a 2× PCM access latency. In the unrealistic case where the PCM access latency is quadrupled (4×), the average performance degradation of Binary Star is only 5% lower than the baseline.

## 6.3 Impact of the Periodic Forced Writeback Interval

Binary Star's periodic forced writeback interval is critical to throughput, number of rollbacks triggered by uncorrectable errors, NVRAM lifetime, and capacity.

**Throughput.** Figure 11 (a) shows the normalized system throughput of different workloads for six different periodic forced writeback intervals, ranging from one second to one hour. We normalize the throughput of each workload to the throughput of the same workload under the 3D DRAM LLC with DRAM configuration. We make two observations. First, throughput decreases when the interval decreases. As the periodic forced writeback interval decreases, both periodic forced writeback and consistent cache writeback happen more often and occupy more NVRAM bandwidth, leading to reduction in throughput. Second, the periodic forced writeback interval impacts the efficiency of consistent cache writeback. As the interval decreases, the locality of NVRAM access reduces such that Binary Star-triggered wear leveling happens more frequently. Therefore, Binary Star triggered wear leveling overlaps much less with endurance-triggered wear leveling, leading to throughput loss.

**Rollback Triggered by Uncorrectable Errors.** Figure 11 (b) shows the rollback rate for six different intervals. Rollback is triggered by uncorrectable errors that occur when accessing dirty cache lines in 3D DRAM. As such, the results also reflect the error rate on dirty cache lines. As the forced writeback interval decreases, dirty cache lines remain in LLC for less time. Therefore, the possibility of uncorrectable-error-triggered rollback reduces. When the interval decreases to one second, the likelihood of error-triggered rollback is $< 10^{-11}$. However, the throughput loss is too much in this case, as shown in in Figure 11 (a). At a 1800-second (30-minute) periodic forced writeback interval, the rollback rate is $10^{-9}$, which is close to the error rate of current DRAM without ECC.

**Endurance.** Figure 11 (c) shows the normalized number of NVRAM writes for six different periodic forced writeback intervals, normalized to the NVRAM writes in the 3D DRAM+PCM baseline for each benchmark. We make three observations. First, the number of writes increases drastically to 17.2× of the PCM-only baseline when the interval is only one second. Such a short interval leads to a low

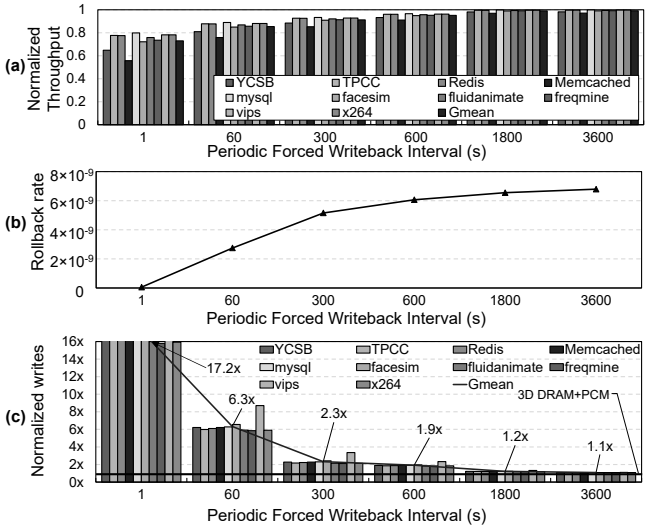



**Figure 11: Effect of varying the periodic forced writeback interval on (a) throughput, (b) probability of rollbacks (i.e., error rate on a dirty line), and (c) NVRAM writes.**

likelihood of rollback, but can cause up to 94% faster NVRAM wear-out due to the larger number of writes. Second, when the interval gets larger, the number of NVRAM writes gradually becomes close to that in the 3D DRAM+PCM baseline. Third, the increase in the interval length has diminishing benefit. We find that increasing the interval from 30 minutes to one hour only leads to a 10% reduction in the number of NVRAM writes, but forces applications to roll back consistent data from earlier in the past (up to one hour instead of up to 30 minutes) when an uncorrectable error happens, leading to high re-execution overheads.

**Lifetime impact of coordinated wear leveling.** We use the same methodology as previous work [90] to evaluate the NVRAM lifetime impact of our design. We determine the end of NVRAM lifetime as the time when NVRAM capacity drops below 90% of the original capacity (i.e., when the over-provisioned space is saturated). We make two observations when the periodic forced writeback interval is 30 minutes (1800 seconds). First, Binary Star triggers 20% additional NVRAM writes, compared to the 3D DRAM+PCM configuration (Figure 11 (c)). This leads to an 8.4% NVRAM lifetime reduction compared with the baseline wear leveling mechanism [94]. As the baseline wear leveling mechanism already offers 10× NVRAM lifetime improvement on native NVRAM without wear leveling [94], our design provides 9.2× better lifetime than native NVRAM, and our coordinated wear leveling mechanism has minimal impact on

NVRAM lifetime. Second, using 3D DRAM as LLC significantly prolongs the lifetime, because it reduces writes to NVRAM by 90.1% in the NVRAM baseline and 87.8% in Binary Star, respectively.

**Capacity.** One disadvantage of consistent cache writeback on NVRAM is that Binary Star gradually consumes more space over time because the consistent cache writeback triggered block remapping consumes free blocks and holds consistent blocks. To free up space for incoming remapping operations, periodic forced writeback releases space for previous checkpointed consistent data (as discussed in Section 4.2). For the above experiments, we guarantee that all the workloads have sufficient memory space while keeping 10% of the NVRAM space for wear leveling.

Overall, the periodic forced writeback interval length causes a trade off between performance, reliability, NVRAM endurance and the effective NVRAM capacity. In this work, we choose 30 minutes (1800s) as the interval, such that periodic forced writeback only incurs less than 1% average throughput loss, with only a $10^{-9}$ chance of rollback and 5 to 7.5 FIT per GB of NVRAM.

## 7 RELATED WORK

To our knowledge, Binary Star is the first memory hierarchy reliability scheme that coordinates the reliability mechanisms across the last-level cache and nonvolatile memory. In this section, we discuss related works that provide reliability to different components in the memory hierarchy.

**Optimizing DRAM main memory reliability.** Redundancy and ECC are commonly used to tackle DRAM errors. Redundancy is used to address hard errors by incorporating redundant rows and columns. Rows and columns with hard errors are remapped using the address decoder and/or the operating system page offlining policy [48, 55]. ECC can be used to recover both transient and hard errors using the parity data generated from the original data. Bose-Chaudhuri-Hocquenghem (BCH) codes can detect and correct errors [43]. CRC codes can only detect errors but provide no correction capability [39]. Single error correction, double error detection (SECDED) is one of the most popular ECC codes used in ECC DRAM [75], which adds an additional DRAM chip in a DIMM to store parity bits [23]. Traditional ECC engines in ECC DRAMs are deployed in a memory controller – we refer to such reliability mechanisms as rank-level ECC, e.g., rank-level SECDED [8]. Server memory systems can also adopt Chipkill to protect against both memory bit errors and chip errors [17, 22, 25]. One of the commercial Chipkill methods is Single Device Data Correction (SDDC) [22, 25], which remedies a single DRAM device failure using a combination of CRC and parity check codes. Several previous works address the storage overhead on traditional ECC [35, 55, 89].

Recent works employ in-DRAM ECC [8, 20, 33, 58], which integrates the ECC engine inside the DRAM chip. Existing LPDDR4 chips have in-DRAM ECC in them [64]. In-DRAM ECC requires changes inside a DRAM chip, introducing extra area, logic, and manufacturing costs.

**SRAM cache reliability.** Kim [36] proposes a design that uses error detection codes for clean cache lines and error correction codes for dirty cache lines. However, this requires flushing the entire cache every 4M cycles, which is unacceptable for a large 3D DRAM. Memory Mapped ECC (MME) [88] uses ECC codes to protect the dirty cache lines without periodic cache flush. However, MME stores error correction codes in main memory, which generates additional writes to NVRAM upon cache writeback. This is not desirable for NVRAM as writes reduce both NVRAM performance and endurance. ECC FIFO [87] stores error correction codes in a FIFO in main memory. However, ECC FIFO still requires cache flushes as frequently as [36].

**Coordinating reliability in heterogeneous architectures.** Gupta et al. [21] investigate the coordination of reliability in a system with heterogeneous main memory that consists of a 3D DRAM and a DIMM-based DRAM. The study focuses on the placement of hot data to improve system performance and reduce the energy consumption of reliability mechanisms in main memory. In contrast, Binary Star focuses on *providing reliability* by coordinating multiple reliability mechanisms across the memory hierarchy.

**Persistent memory.** Persistent memory combines a fast load/store based memory interface with data recovery support [12, 30, 38, 49, 59, 66, 70, 84, 92]. Though not explored in previous works, persistent memory may also be leveraged to recover data from memory errors via multiversioning (e.g., logging and copy-on-write) and write-order control (e.g., memory barriers and forced cache writebacks) [59, 61, 66]. However, these operations are costly. Instead, Binary Star maintains memory system reliability by borrowing only the notion of persistent data commits from persistent memory techniques. As such, Binary Star is orthogonal to previous proposals on persistent memory and can be combined with them to further improve the performance and reliability of persistent memory.

## 8 CONCLUSION

We propose Binary Star, the first coordinated memory hierarchy reliability scheme that achieves both high reliability and high performance with a 3D DRAM last-level cache and nonvolatile main memory. Binary Star coordinates 1) the reliability mechanisms between the last-level cache and main memory, leveraging consistent cache writeback and periodic forced writeback to the main memory to allow the elimination of ECC in the last-level cache, and 2) the wear leveling scheme in main memory with consistent cache writeback to significantly reduce the performance and storage overhead of consistent cache writeback. As a result, we can eliminate the error correction codes in the last-level cache, while achieving higher reliability than using costly sophisticated ECC mechanisms. Our work rethinks the reliability support in the memory hierarchy in light of the next-generation computing systems that may adopt two key new memory technologies, i.e., 3D DRAM caches and byte-addressable nonvolatile memories. We hopefully provide a critical step in reaping the full reliability advantages of these emerging technologies beyond simply replacing the existing memory technologies.

## ACKNOWLEDGMENTS

We thank the anonymous reviewers for their valuable feedback. This paper is supported in part by NSF grants 1829524, 1829525, 1817077, the SRC System Level Design program, and SRC/DARPA Center for Research on Intelligent Storage and Processing-in-memory.

## REFERENCES


[1] J. H. Ahn, S. Li, S. O, and N. P. Jouppi, "McSimA+: A manycore simulator with application-level+ simulation and detailed microarchitecture modeling," in *ISPASS*, 2013.

[2] M. Awasthi, M. Shevgoor, K. Sudan, B. Rajendran, R. Balasubramonian, and V. Srinivasan, "Efficient scrub mechanisms for error-prone emerging memories," in *HPCA*, 2012.

[3] R. Azevedo, J. D. Davis, K. Strauss, P. Gopalan, M. Manasse, and S. Yekhanin, "Zombie memory: Extending memory lifetime by reviving dead blocks," in *ISCA*, 2013.

[4] J. Barr, "EC2 in-memory processing update: Instances with 4 to 16 TB of memory and scale-out SAP HANA to 34 TB," 2017.

[5] C. Bienia, S. Kumar, J. P. Singh, and K. Li, "The PARSEC benchmark suite: Characterization and architectural implications," in *PACT*, 2008.

[6] C. Cagli, "Characterization and modelling of electrode impact in HfO2-based RRAM," in *Proceedings of the Memory Workshop*, 2012.

[7] J. L. Carlson, *Redis in action*. Manning Publications Co., 2013.

[8] 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 *HPCA*, 2017.

[9] H.-M. Chen, C.-J. Wu, T. Mudge, and C. Chakrabarti, "RATT-ECC: Rate Adaptive Two-Tiered Error Correction Codes for Reliable 3D Die-Stacked Memory," *ACM TACO*, 2016.

[10] W. C. Chien, H. Y. Cheng, M. BrightSky, A. Ray, C. W. Yeh, W. Kim, R. Bruce, Y. Zhu, H. Y. Ho, H. L. Lung, and C. Lam, "Reliability study of a 128Mb phase change memory chip implemented with doped Ga-Sb-Ge with extraordinary thermal stability," in *IEDM*, 2016.

[11] N. Christiansen, "Storage class memory support in the Windows OS," in *SNIA NVM Summit*, 2016.

[12] J. Condit, E. B. Nightingale, C. Frost, E. Ipek, B. Lee, D. Burger, and D. Coetzee, "Better I/O Through Byte-addressable, Persistent Memory," in *SOSP*, 2009.

[13] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears, "Benchmarking cloud serving systems with YCSB," in *SoCC*, 2010.

[14] T. Council, "tpc-c benchmark, revision 5.11," 2010.

[15] K. Deierling, "Persistent memory over fabric," in *SNIA NVM Summit*, 2016.

[16] K. Deierling, "Persistent memory over fabric: let it out of the box!" in *Open Server Summit*, 2016.

[17] T. J. Dell, "A white paper on the benefits of chipkill-correct ECC for PC server main memory," *IBM Microelectronics Division*, 1997.

[18] C. Diaconu, "Microsoft SQL Hekaton – Towards Large Scale Use of PM for In-memory Databases," in *SNIA NVM Summit*, 2016.

[19] B. Fitzpatrick, "Distributed caching with memcached," *Linux journal*, 2004.

[20] S. 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 *HPCA*, 2018.

[21] M. Gupta, V. Sridharan, D. Roberts, A. Prodromou, A. Venkat, D. Tullsen, and R. Gupta, "Reliability-aware data placement for heterogeneous memory architecture," in *HPCA*, 2018.

[22] HPE, "Memory RAS technologies in HPE ProLiant/Synergy/Blade servers," 2019, https://psnow.ext.hpe.com/doc?id=4aa4-3490enw.pdf.

[23] M.-Y. Hsiao, "A class of optimal minimum odd-weight-column SEC-DED codes," *IBM Journal of Research and Development*, 1970.

[24] 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," in *ASPLOS*, 2012.

[25] Intel, "Intel Xeon Processor E7 Family: Reliability, Availability, and Serviceability," 2011, https://www.intel.com/content/dam/www/public/us/en/documents/white-papers/xeon-e7-family-ras-server-paper.pdf.

[26] Intel, "An intro to MCDRAM (high bandwidth memory) on Knights Landing," 2016, https://software.intel.com/en-us/blogs/2016/01/20/an-intro-to-mcdram-high-bandwidth-memory-on-knights-landing.

[27] Intel, "Intel launches Optane DIMMs up to 512GB: Apache Pass is here!" 2018, https://www.anandtech.com/show/12828/intel-launches-optane-dimms-up-to-512gb-apache-pass-is-here.

[28] Intel, "Over-provisioning nand-based intel SSDs for better endurance," 2018, https://www.intel.com/content/dam/www/public/us/en/documents/white-papers/over-provisioning-nand-based-ssds-better-endurance-whitepaper.pdf.

[29] "Data-centric innovation spotlight series: Big memory breakthrough for your biggest data challenges," 2019, https://www.intel.com/content/www/us/en/architecture-and-technology/optane-dc-persistent-memory.html.

[30] Intel, "PMDK: Persistent Memory Development Kit," 2019, https://github.com/pmem/pmdk/.

[31] H. Jeon, G. H. Loh, and M. Annavaram, "Efficient RAS support for die-stacked DRAM," in *ITC*, 2014.

[32] U. Kang, H. Chung, S. Heo, D. Park, H. Lee, J. H. Kim, S. Ahn, S. Cha, J. Ahn, D. Kwon, J. Lee, H. Joo, W. Kim, D. H. Jang, N. S. Kim, J. Choi, T. Chung, J. Yoo, J. S. Choi, C. Kim, and Y. Jun, "8 Gb 3-D DDR3 DRAM Using Through-Silicon-Via Technology," *IEEE Journal of Solid-State Circuits*, 2010.

[33] U. Kang, H.-s. Yu, C. Park, H. Zheng, J. Halbert, K. Bains, S. Jang, and J. S. Choi, "Co-architecting controllers and DRAM to enhance DRAM process scaling," in *The Memory Forum*, 2014.

[34] S. Khan, D. Lee, Y. Kim, A. R. Alameldeen, C. Wilkerson, and O. Mutlu, "The Efficacy of Error Mitigation Techniques for DRAM Retention Failures: A Comparative Experimental Study," in *SIGMETRICS*, 2014.

[35] J. Kim, M. Sullivan, and M. Erez, "Bamboo ECC: Strong, safe, and flexible codes for reliable computer memory," in *HPCA*, 2015.

[36] S. Kim, "Area-efficient error protection for caches," in *DATE*, 2006.

[37] Y. Kim, R. Daly, J. Kim, C. Fallin, J. H. Lee, D. Lee, C. Wilkerson, K. Lai, and O. Mutlu, "Flipping Bits in Memory Without Accessing Them: An Experimental Study of DRAM Disturbance Errors," in *ISCA*, 2014.

[38] A. Kolli, J. Rosen, S. Diestelhorst, A. Saidi, S. Pelley, S. Liu, P. M. Chen, and T. F. Wenisch, "Delegated Persist Ordering," in *MICRO*, 2016.

[39] P. Koopman and T. Chakravarty, "Cyclic redundancy code (CRC) polynomial selection for embedded networks," in *DSN*, 2004.

[40] A. Kopytov, "Sysbench manual," *MySQL AB*, 2012.

[41] E. Kültürsay, M. Kandemir, A. Sivasubramaniam, and O. Mutlu, "Evaluating STT-RAM as an energy-efficient main memory alternative," in *ISPASS*, 2013.

[42] B. C. Lee, E. Ipek, O. Mutlu, and D. Burger, "Architecting phase change memory as a scalable DRAM alternative," in *ISCA*, 2009.

[43] S. Lin and D. J. Costello, *Error control coding*. Pearson Education India, 2004.

[44] J. Liu, B. Jaiyen, Y. Kim, C. Wilkerson, and O. Mutlu, "An Experimental Study of Data Retention Behavior in Modern DRAM Devices: Implications for Retention Time Profiling Mechanisms," in *ISCA*, 2013.

[45] C.-K. Luk, R. Cohn, R. Muth, H. Patil, A. Klauser, G. Lowney, S. Wallace, V. J. Reddi, and K. Hazelwood, "Pin: Building customized program analysis tools with dynamic instrumentation," in *PLDI*, 2005.

[46] Y. Luo, S. Govindan, B. Sharma, M. Santaniello, J. Meza, A. Kansal, J. Liu, B. Khessib, K. Vaid, and O. Mutlu, "Characterizing application memory error vulnerability to optimize datacenter cost via heterogeneous-reliability memory," in *DSN*, 2014.

[47] G. Mappouras, A. Vahid, R. Calderbank, D. R. Hower, and D. J. Sorin, "Jenga: Efficient Fault Tolerance for Stacked DRAM," in *ICCD*, 2017.

[48] J. Meza, Q. Wu, S. Kumar, and O. Mutlu, "Revisiting memory errors in large-scale production data centers: Analysis and modeling of new trends from the field," in *DSN*, 2015.

[49] J. Meza, Y. Luo, S. Khan, J. Zhao, Y. Xie, and O. Mutlu, "A case for efficient hardware/software cooperative management of storage and memory," in *WEED*, 2013.

[50] Micron, "Hybrid memory cube specification 2.1," 2014, http://http://www.hybridmemorycube.org/.

[51] Micron, "3D XPoint Technology: Breakthrough Nonvolatile Memory Technology," 2016, https://www.micron.com/products/advanced-solutions/3d-xpoint-technology.

[52] J. Moyer, "Persistent memory in Linux," in *SNIA NVM Summit*, 2016.

[53] O. Mutlu, "Memory scaling: A systems architecture perspective," in *IMW*, 2013.

[54] O. Mutlu and J. S. Kim, "Rowhammer: A retrospective," *IEEE TCAD*, 2019.

[55] P. J. Nair, D.-H. Kim, and M. K. Qureshi, "Archshield: Architectural framework for assisting DRAM scaling by tolerating high error rates," in *ISCA*, 2013.

[56] P. J. Nair, D. A. Roberts, and M. K. Qureshi, "FaultSim: A fast, configurable memory-reliability simulator for conventional and 3D-Stacked systems," *ACM TACO*, 2015.

[57] P. J. Nair, D. A. Roberts, and M. K. Qureshi, "Citadel: Efficiently protecting stacked memory from TSV and large granularity failures," *ACM TACO*, 2016.

[58] P. J. Nair, V. Sridharan, and M. K. Qureshi, "XED: Exposing on-die error detection information for strong memory reliability," in *ISCA*, 2016.

[59] S. Nalli, S. Haria, M. D. Hill, M. M. Swift, H. Volos, and K. Keeton, "An analysis of persistent memory use with WHISPER," in *ASPLOS*, 2017.

[60] D. Narayanan and O. Hodson, "Whole-system persistence," in *ASPLOS*, 2012.

[61] M. A. Ogleari, E. L. Miller, and J. Zhao, "Steal but no force: Efficient Hardware-based Undo+Redo Logging for Persistent Memory Systems," in *HPCA*, 2018.

[62] T. Oh, H. Chung, J. Park, K. Lee, S. Oh, S. Doo, H. Kim, C. Lee, H. Kim, J. Lee, J. Lee, K. Ha, Y. Choi, Y. Cho, Y. 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*, 2015.

[63] Oracle, "MySQL," 2019, https://www.mysql.com/.

[64] M. Patel, J. S. Kim, H. Hassan, and O. Mutlu, "Understanding and Modeling On-Die Error Correction in Modern DRAM: An Experimental Study Using Real Devices," in *DSN*, 2019.

[65] M. Patel, J. S. Kim, and O. Mutlu, "The Reach Profiler (REAPER): Enabling the Mitigation of DRAM Retention Failures via Profiling at Aggressive Conditions," in *ISCA*, 2017.

[66] S. Pelley, P. M. Chen, and T. F. Wenisch, "Memory persistency," in *ISCA*, 2014.

[67] M. K. Qureshi, D. Kim, S. Khan, P. J. Nair, and O. Mutlu, "AVATAR: A Variable-Retention-Time (VRT) Aware Refresh for DRAM Systems," in *DSN*, 2015.

[68] M. K. Qureshi, M. M. Franceschini, and L. A. Lastras-Montano, "Improving read performance of phase change memories via write cancellation and write pausing,"

in *HPCA*, 2010.
[69] 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 *MICRO*, 2009.
[70] J. Ren, J. Zhao, S. Khan, J. Choi, Y. Wu, and O. Mutlu, "ThyNVM: Enabling software-transparent crash consistency in persistent memory systems," in *MICRO*, 2015.
[71] S. Rixner, W. J. Dally, U. J. Kapasi, P. Mattson, and J. D. Owens, "Memory access scheduling," in *ISCA*, 2000.
[72] D. Roberts and P. Nair, "FAULTSIM: A fast, configurable memory-resilience simulator," in *The Memory Forum: In conjunction with ISCA*, 2014.
[73] SAP, "SAP HANA: an in-memory, column-oriented, relational database management system," 2014, http://www.saphana.com/.
[74] S. Schechter, G. H. Loh, K. Strauss, and D. Burger, "Use ECP, not ECC, for hard failures in resistive memories," in *ISCA*, 2010.
[75] B. Schroeder, E. Pinheiro, and W.-D. Weber, "DRAM errors in the wild: a large-scale field study," in *SIGMETRICS*, 2009.
[76] N. H. Seong, S. Yeo, and H.-H. S. Lee, "Tri-level-cell phase change memory: Toward an efficient and reliable memory system," *ISCA*, 2013.
[77] J. Sim, G. H. Loh, V. Sridharan, and M. O'Connor, "Resilient die-stacked DRAM caches," in *ISCA*, 2013.
[78] V. Sousa, "Phase change materials engineering for RESET current reduction," in *IMW*, 2012.
[79] V. Sridharan, N. DeBardeleben, S. Blanchard, K. B. Ferreira, J. Stearley, J. Shalf, and S. Gurumurthi, "Memory errors in modern systems: The good, the bad, and the ugly," in *ASPLOS*, 2015.
[80] K. Suzuki and S. Swanson, "The Non-Volatile Memory Technology Database (NVMDB)," 2015, http://nvmdb.ucsd.edu.
[81] S. Swami, P. M. Palangappa, and K. Mohanram, "ECS: Error-correcting strings for lifetime improvements in nonvolatile memories," *ACM TACO*, 2017.
[82] C. Szegedy, Wei Liu, Yangqing Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *CVPR*, 2015.
[83] The Apache Software Foundation, "Spark," 2014, http://spark.incubator.apache.org/.
[84] H. Volos, A. J. Tack, and M. M. Swift, "Mnemosyne: Lightweight persistent memory," in *ASPLOS*, 2011.
[85] VoltDB, "VoltDB: Smart data fast," 2014, http://voltdb.com/.
[86] D. S. Yaney, C. Y. Lu, R. A. Kohler, M. J. Kelly, and J. T. Nelson, "A meta-stable leakage phenomenon in DRAM charge storage - Variable hold time," in *IEDM*, 1987.
[87] D. H. Yoon and M. Erez, "Flexible cache error protection using an ECC FIFO," in *SC*, 2009.
[88] D. H. Yoon and M. Erez, "Memory mapped ECC: low-cost error protection for last level caches," in *ISCA*, 2009.
[89] D. H. Yoon and M. Erez, "Virtualized ECC: Flexible reliability in main memory," *IEEE micro*, 2011.
[90] D. H. Yoon, N. Muralimanohar, J. Chang, P. Ranganathan, N. P. Jouppi, and M. Erez, "FREE-p: Protecting non-volatile memory against both hard and soft errors," in *HPCA*, 2011.
[91] Z. Zhang, W. Xiao, N. Park, and D. J. Lilja, "Memory module-level testing and error behaviors for phase change memory," in *ICCD*, 2012.
[92] 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 *MICRO*, 2013.
[93] W. Zhao, E. Belhaire, Q. Mistral, C. Chappert, V. Javerliac, B. Dieny, and E. Nicolle, "Macro-model of spin-transfer torque based magnetic tunnel junction device for hybrid magnetic-CMOS design," in *IEEE International Behavioral Modeling and Simulation Workshop*, 2006.
[94] P. Zhou, B. Zhao, J. Yang, and Y. Zhang, "A durable and energy efficient main memory using phase change memory technology," in *ISCA*, 2009.
[95] W. K. Zuravleff and T. Robinson, "Controller for a Synchronous DRAM That Maximizes Throughput by Allowing Memory Requests and Commands to Be Issued Out of Order," 1997.