Anchoring Whole-System Persistence and Resilience in CXL

Yuchen Zhou
Purdue University
West Lafayette, IN, USA
zhou1166@purdue.edu

Jianping Zeng
Arizona State University
Tempe, AZ, USA
jpzeng@asu.edu

Changhee Jung
Purdue University
West Lafayette, IN, USA
chjung@purdue.edu

Abstract

This paper presents ANCHOR, a novel CXL-based memory hierarchy that achieves both persistence and resilience without recompilation or core modifications. For performant whole-system persistence (WSP), ANCHOR introduces a dual-path store architecture in which every committed store is sent to both the L1 cache and a CXL-attached memory-semantic SSD. To reduce the SSD write traffic, ANCHOR employs an STT-RAM write-combining buffer that coalesces stores before they reach the SSD. In particular, the SSD-resident committed store data serves as a ground-truth *anchor* for error repair, where ANCHOR repurposes cache and DRAM ECCs as detection codes and repairs detected errors using the corresponding SSD-resident clean copy. This approach effectively upgrades cache-side end-to-end reliability from intrinsic 1-bit correction to 3-bit correction, while also extending DRAM repair coverage beyond the limits of standard ChipKill—all at no additional ECC cost. Evaluation across 42 benchmarks shows that ANCHOR incurs only a 4.36% average run-time overhead while ensuring both WSP and system-wide resilience.

CCS Concepts

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

Keywords

CXL, whole-system persistence, resilience

ACM Reference Format:
Yuchen Zhou, Jianping Zeng, and Changhee Jung. 2026. Anchoring Whole-System Persistence and Resilience in CXL. In *2026 International Conference on Supercomputing (ICS '26), July 06–09, 2026, Belfast, United Kingdom.* ACM, New York, NY, USA, 17 pages. https://doi.org/10.1145/3797905.3800523

1 Introduction

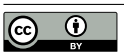
Emerging non-volatile memory (NVM) technologies—such as PCM [89, 136, 144, 154], ReRAM [4, 25], and STT-RAM [27, 69, 83, 93, 127]—offer data persistence, high cell density, DRAM-comparable latency, and byte-addressability. When deployed on the memory bus as *storage-class memory (SCM)* or *persistent memory*, these technologies enable *high-performance persistence* via regular stores, instead of going through the slow, software-heavy block-based storage stack (*e.g.,* file systems and I/O flushing on HDDs or SSDs). Products such as Intel Optane DC Persistent Memory [73] exemplify this class of SCM, effectively bridging the gap between volatile memory and storage. The capabilities of SCM have spurred research into *whole-system persistence (WSP)* [79, 170, 174, 182]—that transparently persists all program data, including the OS and libraries, across power failure without requiring source code changes.

To realize WSP on SCM, *LightWSP* [182] and *cWSP* [174] leverage a *non-temporal data path*[1] [74, 137, 145, 174, 182], separated from the regular cache-based path, to bypass the entire cache hierarchy and write all data directly to persistent memory. To ensure correct power failure recovery, they rely on compilers to partition the entire software stack, including all applications, OS, and libraries, into a series of recoverable code regions and enforce so-called *region-level persistence*, where each region boundary serves as a recovery point. Despite not requiring source code changes, these approaches fail to achieve fully transparent WSP due to the mandatory recompilation of the code. To achieve full transparency, *PPA* [170] removes compiler dependence by implementing WSP purely in hardware; however, this comes at the cost of intrusive core microarchitectural changes that are difficult to integrate into commodity processors. Thus, existing WSP mechanisms are technically sound but remain neither lightweight nor practical for real-world deployment.

Even worse, all prior WSP schemes are tightly coupled to SCM DIMMs attached via conventional DDR-style interfaces. On the device side, SCM cell arrays and peripheral circuitry impose power, latency, and endurance constraints that limit per-DIMM capacity [18, 173]. On the interface side, SCM is constrained by the same finite set of DDR memory channels and DIMM slots as DRAM, bounding per-socket SCM capacity by channel and slot counts. Furthermore, SCM devices remain relatively expensive per bit, and the limited per-DIMM capacity forces systems to provision more (costly) DIMMs to build a large persistent memory pool. Hence, scaling the persistent memory pool with SCM inherently requires spending additional DDR resources and is ultimately capped by DDR-interface limitations [172, 173]. As a result, SCM-based WSP inherits restricted capacity growth, limited scalability, and high cost, making SCM a poor foundation for building large, cost-effective persistent memory pools.

Meanwhile, NAND flash emerges as a more scalable and cost-efficient substrate for persistence [34, 55, 58, 115, 116, 177]. It underlies modern SSDs and scales over high-bandwidth I/O fabrics (*e.g.,* PCIe), unlike SCM that competes with DRAM for DDR channels or DIMM slots. Thanks to simple cells, multi-level storage (MLC [20]/TLC [120]/QLC [98]), and 3D die-stacking [14, 61, 110], NAND delivers much higher capacity and lower cost per bit than SCM, allowing enterprise SSDs to scale to tens of terabytes per device with low standby power. Thus, NAND-based SSDs decouple

[1] An existing data path in Intel x86 processors that bypasses caches and connects the store buffer directly to memory.



ICS '26, Belfast, United Kingdom

ACM ISBN 979-8-4007-2522-7/26/07
https://doi.org/10.1145/3797905.3800523

persistent-capacity scaling from DDR channel/slot constraints, rendering large persistent memory pools both cost-efficient and practical without impacting the capacity or bandwidth of the main memory (DRAM). However, these benefits are exposed only through a coarse-grained, block-based storage stack, adding microsecond-scale latency and substantial software overhead. In this traditional form, NAND-based SSDs excel as block devices but cannot serve as a fine-grained, load/store–level persistence substrate like SCM.

Fortunately, *Compute eXpress Link (CXL)* [35, 146] provides a practical way to overcome this mismatch between NAND's device properties and its coarse-grained interface. CXL.mem provides cacheline-granular, load/store access by mapping external devices directly into the processor's physical address space, allowing NAND-backed devices to function as memory-like persistence backends rather than block storage. This eliminates most storage-stack overhead while exposing the internal parallelism and bandwidth of enterprise SSDs. Modern memory-semantic SSDs, such as Samsung's CXL SSD [1], build on this model and integrate CXL.mem with device-side DRAM caching and flash management optimized for fine-grained random access, thereby masking flash latency and absorbing bursty read/write traffic. These devices thus deliver effective latency and throughput approaching SCM-based persistent memory, while retaining NAND flash's capacity and cost advantages [173]. Together, NAND flash's density advantage, CXL's fine-grained load/store access, and the device-side caching and write coalescing of memory-semantic SSDs enable a new class of cost-effective, high-capacity, and high-performance persistent systems that, like prior SCM-based designs, provide persistence directly through regular stores.

Beyond persistence, such CXL-based persistent systems also have the potential to enhance *system reliability*. Traditional memory hierarchies rely on modest ECC schemes, *e.g., single-error correction and double-error detection (SECDED)* [62, 88, 135] in caches and *ChipKill*-class protection in DRAM [21, 37, 56, 57, 121, 176]. While SECDED corrects only single-bit errors, ChipKill extends protection to the failure of an entire DRAM chip. However, both remain limited by their native local correction capability, leaving more complex error patterns uncorrectable. This is increasingly problematic as multi-bit upsets—that can manifest as multi-chip error patterns—and more complex memory faults become more prevalent in both caches and DRAM [23, 52, 87, 112, 165, 166]. Notably, we found out that by persisting all committed store data in the SSD via a separate persistent path, a CXL-based WSP design can leverage the SSD-resident replica as a *ground-truth anchor* for error repair. In this design, the volatile memory hierarchy can repurpose existing ECC for error detection—for example, SECDED can provide deterministic detection of up to 3-bit errors when its check bits are used solely for detection [17]—and then repair corrupted data by refetching the clean replica from the SSD. The insight is that such strong error resilience can be made for free atop the WSP design, reducing the need for stronger on-chip ECC schemes, *e.g.,* double-error correction and triple-error detection (DECTED) ECC.

Based on the insight, this paper proposes ANCHOR, a novel CXL-based memory hierarchy that can achieve both whole-system persistence (WSP) and resilience without recompilation or core modifications. For lightweight yet performant WSP, ANCHOR employs the *dual-path store architecture*: each committed store takes a volatile path through caches to DRAM as usual, while the data being stored is also forwarded to a CXL-attached memory-semantic SSD via the separate persistent path. Because stores retire in program order and the persistent path taps this committed-store stream without introducing additional reordering, the updates written into the SSD respect the same ordering constraints as the underlying memory model, which serves as a basis for ANCHOR's correct power failure recovery. For strong error resilience, caches and DRAM prioritize detection; upon detecting corrupted data, the system discards it and reloads its clean replica from the SSD that offers strong data protection (*e.g.,* LDPC/BCH codes) [84, 91, 153, 155, 180]. In effect, ANCHOR endows the CXL-based WSP with substantially stronger error resilience than conventional SECDED ECC—enabling 3-bit error correction while using the ECC solely for detection—by exploiting persistent replicas at no additional cost.

To mitigate write pressure to the SSD, ANCHOR introduces a *write combining buffer (WCB)* on the persistent path. The WCB coalesces multiple stores to the same address into a single persistent update, preserving per-address program-order semantics, while filtering out intermediate overwrites and significantly reducing SSD write amplification. ANCHOR implements the WCB using a small STT-RAM array that offers non-volatility, low latency, high endurance, and improved resilience. Upon power failure, ANCHOR performs a *just-in-time (JIT) checkpoint* of the register file and any remaining store in the store buffer—whereas the STT-RAM WCB retains its buffered stores. It turns out that most committed stores have already reached either the SSD or the WCB, and thus the amount of data to be JIT checkpointed is small, which helps ANCHOR keep recovery latency low. After power is restored, the processor reconstructs a consistent memory state by draining the WCB (if needed) and restoring data from the SSD back into DRAM, without relying on any crash-consistency mechanisms, *e.g.,* rollback recovery based on undo, redo, or ido loggings [68, 100, 156].

The evaluation using 42 applications from SPEC CPU2006/2017, SPLASH3, STAMP, NPB, and WHISPER shows that ANCHOR incurs only a 4.36% run-time overhead on average. That is, ANCHOR provides a practical and scalable direction for next-generation CXL-based systems to achieve both persistence and resilience at minimal cost. Overall, the contributions of ANCHOR are as follows:

- It is the first CXL-based WSP—that does not require compiler instrumentation or core pipeline modifications.
- It is the first WSP that improves resilience by repurposing existing ECC for detection without costly stronger ECC.
- It devises an STT-RAM WCB that merges data being stored (up to 91.23% merge rate), reducing write amplification a lot.
- It delivers lightweight persistence and strong resilience, providing a deployable path toward WSP in CXL systems.

2 Background and Motivation

2.1 CXL and Memory-Semantic SSDs

Compute eXpress Link (CXL) is an emerging interconnect standard that unifies memory and device communication across CPUs, accelerators, and storage-class devices. CXL defines three sub-protocols: CXL.io, CXL.cache, and CXL.mem. Among them, *CXL.mem* provides a low-latency, byte-addressable interface that allows external

devices to participate directly in the processor's physical address space while preserving ordering and coherence semantics.

Unlike traditional PCIe-attached storage accessed through the block-based I/O stack, CXL.mem devices can appear as extensions of system memory and be accessed via regular load and store instructions. This capability makes CXL particularly attractive for high-performance persistence: a CXL-attached storage device can expose its memory regions—that are reachable without file systems, syscalls, or explicit I/O flushing. Building on this interface, recent *memory-semantic SSDs*, such as Samsung's CXL-based SSD [173], combine high-capacity NAND flash with on-device DRAM and a memory-like CXL.mem front-end. Internally, the controller uses the DRAM as a cache and staging area in front of NAND and—as usual—employs a flash translation layer (FTL) to manage wear leveling, garbage collection, and address mapping. Externally, the device exposes a CXL.mem-mapped region that the host can read and write directly as if it were persistent memory.

This organization is well-suited for whole-system persistence. NAND flash offers terabyte-scale capacity at low cost per bit, while the DRAM-backed controller absorbs bursty, fine-grained writes and converts them into flash-friendly transactions, delivering effective latency and throughput close to that of DRAM. When attached via CXL.mem, such memory-semantic SSDs act as high-capacity persistent backends that are directly reachable through regular load and store instructions. This allows ANCHOR to realize WSP without the drawbacks of SCM-based approaches (Section 1), while also exploiting SSD-resident committed data as a global correction anchor for strong error correction.

2.2 ECC Limitations and Opportunity

Soft errors, caused by radiation, noise, or device wearout, are a major reliability concern in modern memory systems. Commercial processors therefore deploy ECC in both caches and DRAM, most commonly *single-error correction, double-error detection (SECDED)* and ChipKill-class schemes [21, 37, 56, 57, 121, 176]. As technology scales and memory capacity grows, error patterns become more complex: prior work reports an increased likelihood of multi-bit soft-error events in large SRAM arrays [23, 87, 112, 166]. Likewise, field studies of production servers report not only single-bit but also multi-bit and multi-symbol DRAM failures that exceed the correction capability of conventional DRAM ECC schemes, including ChipKill-class schemes [23, 52, 85, 87, 112, 121, 143, 165, 166].

A natural reaction is to strengthen on-chip ECC, *e.g.,* moving beyond SECDED/ChipKill toward multi-bit or multi-symbol correction and stronger detection guarantees. Unfortunately, stronger ECC requires increasingly complex encoder/decoder logic (syndrome computation, error localization, and in-place correction), which increases design and verification complexity and potentially lengthens the critical path, raising both latency and energy consumption [6, 121, 176]. Consistent with this concern, our evaluation of *SPEC CPU 2006/2017* highlights that even a single additional cycle on cache accesses can noticeably degrade end-to-end performance (up to 12.47%). A large body of work therefore seeks to reduce ECC overhead via optimized codes, selective protection, or adaptive mechanisms [5, 6, 9, 39, 40, 47, 48, 80, 90, 157, 166, 175]; however, it turns out that these approaches still require each volatile layer (*i.e.,* caches and DRAM) to independently perform on-the-fly *correction*.

In contrast, SSD controllers already implement much stronger LDPC [51] or BCH [15] codes over multi-kilobyte flash pages (*e.g.,* 4–16 KB), often correcting dozens to hundreds of bit errors per page [84, 91, 153, 155, 180]. This asymmetry suggests a cross-layer opportunity: if the system continuously duplicates committed stores onto a memory-semantic SSD, it can maintain an ECC-protected reference copy of committed state and use it as a *ground-truth anchor* for repair. Our key insight is that both caches and DRAM no longer need to implement strong, low-latency *correction* on every access, and they can instead prioritize *detection* (*i.e.,* repurposing existing SECDED/ChipKill solely for detection) and repair detected corruptions by refetching the corresponding data from the SSD. Based on this insight, ANCHOR can improve existing error resilience on top of its WSP at no additional cost. Recall that SECDED ECC can detect up to 3 bits if its check bits are dedicated entirely to detection [17]; similarly, when ChipKill code is used solely for detection, it can detect multi-chip failure [80, 128].

3 ANCHOR's Approach

The goal of ANCHOR is to provide both *whole-system persistence (WSP)* and enhanced error resilience while keeping the architecture minimally intrusive and commercially practical. To achieve this, ANCHOR combines a CXL memory-semantic SSD with its dual-path persistence architecture that anchors committed store data in the SSD. This SSD-resident anchor not only ensures persistence but also enables stronger error resilience through lightweight error detection in the volatile memory hierarchy and SSD-anchored repair. To preserve practicality, ANCHOR (1) avoids modifications to the core pipeline, (2) requires neither compiler instrumentation nor ISA extensions, and (3) leaves the existing cache/DRAM hierarchy unchanged. Assuming a fail-stop model, a register file, caches, and DRAM lose their states on power failure, whereas the CXL-attached memory-semantic SSD retains the contents of its CXL.mem region. We also assume that soft errors may occur in caches, DRAM, and on-device buffers, with the SSD controller's LDPC/BCH ECC providing standard data protection.

3.1 System Model

ANCHOR targets a conventional multicore system in which each core connects to a cache hierarchy backed by off-chip DRAM. DRAM remains the *sole* main memory: all processor loads are served from the cache/DRAM hierarchy, and the operating system and applications view memory as a standard volatile address space. In addition, the system includes a CXL-attached memory-semantic SSD—featuring NAND flash, an on-device DRAM cache, and strong controller-level ECC—to expose a byte-addressable CXL.mem region. The SSD firmware guarantees that any completed CXL.mem write is rendered persistent across power failure (including data buffered in DRAM cache). Consequently, ANCHOR can treat the entire CXL.mem region as a persistent domain.

Also, ANCHOR adopts the conventional data-race-free (DRF) assumption; programmers use locks, atomics, and fences to establish the required happens-before relationships between threads, and program with data races is considered out of scope. ANCHOR

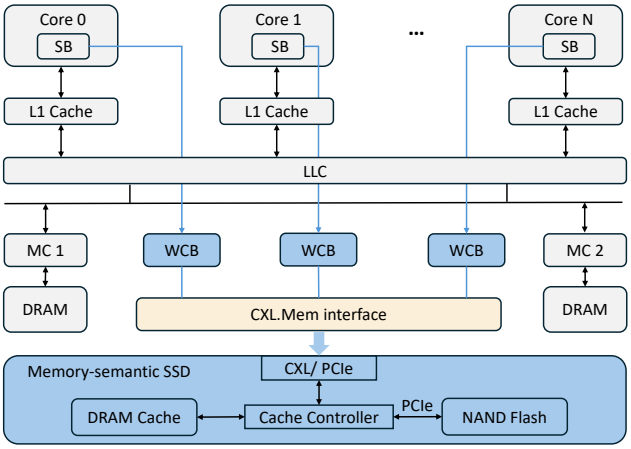


Figure 1: High-level view of ANCHOR; each committed store is sent along two paths: a regular path and a persistent path (blue line); blue boxes signify non-volatile parts.

extends these same synchronization points to persist ordering, *i.e.,* stores ordered by fences or synchronization are likewise ordered on the persistent path. Specifically, a fence does not complete until all older stores have reached persistent domain (Section 4.4).

3.2 Dual-Path Store Architecture

Figure 1 illustrates ANCHOR's dual-path design. On the regular data path, each committed store flows through the existing cache hierarchy toward DRAM exactly as in a conventional system. Thus, DRAM remains the main memory for program execution, and the cache/DRAM subsystem operates unchanged.

Meanwhile, ANCHOR adds a separate *persistent path* (blue line in the figure) that taps into the store buffer. When a store commits, the store buffer forwards its address and data to a small, non-volatile *write combining buffer* (WCB) on the persistent path. The WCB merges multiple stores that target the same address, preserving program-order semantics while filtering out intermediate overwrites, and then streams coalesced writes over CXL.mem into the memory-semantic SSD.

Importantly, the SSD is never placed on the read path: processors do not load from the SSD during normal execution. This memory hierarchy fundamentally differs from NVM-main-memory designs where NVM serves as main memory with DRAM used as the last-level cache (LLC); in those systems, the persistent copy may still be in flight along the persistent path when a LLC load miss consults NVM, leading to subtle stale-load issues [77, 79, 122, 182]. In contrast, ANCHOR lets all normal loads go to DRAM as usual, *i.e.,* the SSD is consulted only for error correction and recovery, eliminating stale-load issues by construction.

3.3 Persistence and Recovery

From a persistence standpoint, ANCHOR maintains a continuously updated persistent domain that includes the CXL-attached memory-semantic SSD and the non-volatile WCB. As stores commit, they first enter the store buffer and are then forwarded into the WCB; upon reaching the WCB, their data enters the persistent domain and can eventually be drained to the SSD. At any time, most committed stores thus reside either in the SSD or the WCB. Only a small tail of recently committed stores may still be buffered in the (volatile) store buffer, *i.e.,* have not yet entered the persistent domain.

In the event of an unexpected power outage, volatile structures such as caches and DRAM lose their contents—whereas both the SSD and the WCB remain intact. To this end, ANCHOR performs a *just-in-time (JIT) checkpoint* just before the outage to save the register file and any committed stores remaining in the store buffer (*i.e.,* those that have not yet reached the WCB) into a small metadata area on the SSD (more details are deferred to Section 4.6). Because the vast majority of committed stores have already drained to the SSD or are buffered in the WCB, the amount of state requiring a checkpoint on the critical path is minimal, keeping both hardware complexity and recovery latency low.

After power is restored, ANCHOR reconstructs memory from the persistent domain. At a high level, recovery finalizes any pending stores buffered in the WCB, replays the store buffer's committed entries recorded during the JIT checkpoint, and restores the SSD-resident data in the CXL.mem region back into DRAM. Note that neither software-based logging/crash-recovery protocol nor compiler instrumentation is required for the recovery process. Conceptually, ANCHOR's recovery reconstructs a prefix of the original program status: all committed stores in that prefix have been made persistent in the persistent domain and are reflected into DRAM before execution resumes (see more in Section 4.6).

3.4 Persistence-Driven Error Resilience

Beyond persistence, ANCHOR rethinks reliability by leveraging the SSD-resident committed state as a *ground-truth anchor* for error repair. Traditional designs equip each volatile layer (*e.g.,* caches and DRAM) with full on-the-fly *correction*, incurring non-trivial encode/decode overhead and possibly increasing access latency and energy, yet limiting protection to the local correction capability of each level. In contrast, SSD controllers already protect data with much stronger LDPC/BCH codes over multi-kilobyte pages.

ANCHOR exploits this reliability asymmetry with a cross-layer approach that shifts the burden of *correction* out of the volatile memory hierarchy. Caches and DRAM instead prioritize *detection* by reusing existing ECC check bits. For caches, operating standard SECDED in a detect-only mode yields *triple-error detection* (TED), *i.e.,* it deterministically detects up to 3-bit errors within a codeword [17]. For DRAM, ANCHOR does not rely on a specific ChipKill instantiation or claim a universal bit-level detection bound. Rather, the key point is that ChipKill-class protection can offer a detection space broader than its native local-correction subset. For intuition, some ChipKill-style organizations (often based on symbol-level protection such as Reed-Solomon [80, 121]) treat the data contributed by one DRAM device per transfer as one symbol; this is 4 bits for ×4 devices and 8 bits for ×8 devices. Under such an organization, detecting two faulty symbols/devices corresponds roughly to detecting two such symbols per transfer. We use this only as an intuitive example rather than a universal bound across all ChipKill implementations.

Compared to a conventional design whose DRAM reliability is strictly bounded by ChipKill's native local-correction capability, ANCHOR expands the effective repairable error space to the broader set of errors that can be detected by the deployed DRAM protection; once such errors are detected, ANCHOR can correct them just by refetching the clean data from the SSD-backed committed image. In the meantime, the WCB is handled separately using lightweight ECC, sufficient to prevent corruption in its entries from propagating into the persistent image on the SSD (see more in Section 4.7).

4 Implementation Details

This section presents the detailed design of ANCHOR, including the key mechanisms that enable whole-system persistence, power failure recovery, and soft error resilience.

4.1 Hardware Organization

ANCHOR targets a conventional multicore system in which each core connects to a cache hierarchy backed by off-chip DRAM; refer to Figure 1. As usual, DRAM remains the only main memory, and the OS and applications continue to view memory as a normal volatile address space.

For ANCHOR to realize whole-system persistence, the system necessitates a CXL-attached memory-semantic SSD that exposes a byte-addressable CXL.mem region. At boot time, firmware reserves a contiguous physical address range and maps it to the SSD's CXL.mem region. ANCHOR also adds a per-core non-volatile write combining buffer (WCB) on the persistent path. Each core's WCB coalesces stores targeting the same (cacheline) address before sending them to the SSD, thereby reducing write amplification. Together, the WCB and the SSD-side CXL.mem region form ANCHOR's persistent domain.

The SSD internally consists of a NAND flash array, a controller with strong LDPC/BCH ECC, and an on-device DRAM cache. The reserved CXL.mem region is used exclusively as a hardware-maintained persistent target for committed memory updates. That is, the SSD never serves regular loads, and it remains invisible to applications. File-system storage (if any) may reside on a separate NVMe namespace—or another device—and is orthogonal to our design.

4.2 Dual-Path Store Microarchitecture

4.2.1 Regular Path. The regular path of ANCHOR is unchanged from a conventional system. That is, stores propagate through the cache hierarchy and update DRAM under the existing coherence and write-back policies, while loads are served from caches/DRAM except during error repair or recovery.

4.2.2 Persistent Path. The persistent path begins at the store buffer, which forwards committed stores to the WCB in the same order they retire. ANCHOR treats the store buffer as the single ordering point, *i.e.,* the WCB observes the same per-core commit order as the regular memory path and does not introduce any additional reordering beyond the underlying memory model. A store leaves the store buffer only after it receives an acknowledgement from the WCB.

While stores are 8 B (word) granularity in the core, the CXL.mem interface operates at a cacheline granularity of 64 B. To this end, ANCHOR uses a **64 B-granular WCB** by default, *i.e.,* the WCB aggregates 8 B stores—forwarded by the store buffer—into full 64 B entries that match the CXL.mem interface. This allows the persistent path to naturally issue 64 B aligned writes without extra padding or read–modify–write traffic on the CXL link. Also, Section 5.4 evaluates an **8 B-granular** variant.

4.3 Write Combining Buffer

4.3.1 Organization and Metadata. ANCHOR organizes each WCB as a set-associative structure whose entry accommodates a full 64 B chunk of data being stored. This organization allows the WCB to merge multiple 8 B stores to the same cacheline granularity before issuing a single 64 B write to the CXL.mem interface, increasing the coalescing rate and reducing downstream SSD write traffic.

The WCB is implemented in STT-RAM rather than SRAM or eDRAM for three reasons. First, STT-RAM is non-volatile, so WCB contents survive power failure and do not need to be included in the just-in-time checkpoint; recovery can simply drain any WCB entries to the SSD in place. Second, since STT-RAM stores information magnetically rather than as charge, its cells are substantially less susceptible to radiation-induced soft errors than conventional SRAM/DRAM arrays [141, 151, 162], which is attractive for a structure that lies directly on the persistent path and therefore must remain reliable to preserve both persistence and resilience guarantees. Third, compared with write-limited NVMs such as PCM and ReRAM, STT-RAM offers much higher write endurance—approximately 10^{12}–10^{15} writes per cell, versus about 10^{8} for PCM and 10^{11} for ReRAM [27, 118, 134, 150]—so the small per-core WCB does not pose lifetime concerns even under heavy write traffic.

Note that the use of STT-RAM for a small, heavily utilized buffer is also consistent with prior work that employs STT-RAM in on-chip memory structures, especially last-level caches, to leverage its non-volatility, high density, low leakage power, and CMOS compatibility [22, 27, 94, 152]. The implication is that the integration of the proposed WCB is technologically feasible without hassle. To support both efficient merging and crash-consistent recovery, each WCB entry contains the following fields:

- **Tag (physical line address).** Identifies which 64 B physical line the entry belongs to, enabling associative lookup and allowing multiple lines to be tracked concurrently in the set-associative structure.
- **Valid bit.** Distinguishes allocated entries from empty slots in each set, so that the WCB can probe only live entries on a lookup and select a true victim on replacement.
- **64 B data field.** Caches the combined data for the 64 B line. As 8 B stores arrive, their values are merged into this field so that the persistent path can eventually emit a single, fully-formed 64 B write without fetching the data from the SSD.
- **Per-word write-valid mask.** Tracks which 8 B words within the 64 B line have been updated by in-flight stores. This mask allows the WCB to safely flush partially-updated lines: on eviction, only the marked words overwrite the corresponding SSD contents, avoiding read–modify–write traffic while preserving the untouched words.
- **Lightweight ECC bits.** Protect the 64 B data field stored in the non-volatile WCB itself. We use SECDED per 8 B word.

The controller checks (and, if needed, corrects) WCB contents before draining them to the SSD and before using them during recovery, preventing silent corruption in buffered the in-flight stores.

4.3.2 Insert and Merge Logic. Upon receiving a store from the store buffer, the WCB computes the 64 B-aligned line address and indexes the WCB sets to search for a matching tag. If a matching entry is found, the following actions are performed in a row:

- The store's data are merged into the entry's data field, respecting the word mask.
- The corresponding bits in the per-word write mask are set.
- The entry's valid bit is set.

If no matching entry exists, the controller first looks for an empty way in the indexed set, in which case one of the two actions occurs:

- If there is at least one invalid entry, the store is inserted into that entry, initializing the tag, data, and write mask, with marking the entry valid.
- If all ways in the set are valid, the WCB selects a victim way using LRU; if the victim is valid, it is marked for drain (see below); once reclaimed, its entry is overwritten with the new tag, data, and write mask.

ANCHOR uses **LRU replacement** rather than FIFO as it achieves a higher merge rate; lines that have been accessed recently are more likely to receive additional stores, and keeping them in the WCB increases the likelihood of coalescing multiple stores into a single SSD write. We compare LRU against a simpler **time-based eviction** policy that evicts entries once they have resided in the WCB longer than a fixed timeout— sweeping timeouts from 1,000 to 5,000 cycles in 1,000-cycle increments. It turns out that the time-based policy yields a 26.69% lower merge rate than ANCHOR.

4.3.3 Drain Policy and Eviction. An entry in the WCB becomes *drainable* when it is valid (*i.e.,* it holds a merged 64 B line)—and not already in flight to the SSD. A drainable entry can be safely sent via the CXL.mem path, and its slot can be reclaimed once the SSD acknowledges that the write has entered its persistent domain.

When a WCB set is full, any store that maps to that set must wait for an entry to be reclaimed, which can backpressure the store buffer and eventually stall the core pipeline. To mitigate this, the WCB controller proactively drains entries before a set becomes completely full. Each set tracks its number of valid entries; when this occupancy exceeds three quarters of the associativity (chosen empirically as a good trade-off between backpressure and merge rate), the controller begins flushing drainable entries from that set; among all drainable entries, the controller selects the LRU entries as drain victims. The draining process continues until the number of valid entries falls back below the three-quarter threshold.

Such proactive flushing reduces the chance of full-set stalls but cannot eliminate them because flushing is not instantaneous. A victim must be sent over CXL.mem and acknowledged by the SSD as having entered its persistent domain before its entry can be reclaimed. As a result, even though flushing begins in a set upon exceeding the three-quarter threshold, those stores targeting the same set may arrive fast enough to fill all its available ways (*i.e.,* the remaining quarter) before any earlier flush completes; each way in the set is either occupied by the in-fight stores or tied up by the flushing, leaving no available room for later stores, in which case they must wait, possibly exerting backpressure on the store buffer and, in turn, stalling the core pipeline.

4.4 Persist Ordering and Fence Handling

ANCHOR does not rely on compiler instrumentation or introduce new ISA primitives for persist ordering. Instead, it reuses existing synchronization operations and extends their ordering guarantees to the persistent path. Specifically, if two stores are ordered by via fences or stronger synchronization under a given memory consistency model, then ANCHOR preserves that ordering for persistence as well. In ANCHOR, a store becomes persistent once it reaches the WCB, so synchronization-induced happens-before relations naturally extend to the WCB—which serves as the persistence point.

To enforce this guarantee, when a core executes a memory fence (*e.g.,* `mfence` or equivalent), the fence is allowed to retire only after all older stores have been inserted into the WCB from the store buffer. Thus, a fence does not complete until all those stores preceding the fence have entered the persistent domain, *i.e.,* reaching WCB. Those stores need not reach the SSD before the fence completes, because our recovery (see more in Section 4.6) always drains valid WCB contents to the SSD before reconstructing memory from the SSD-backed image. This ensures that stores before the fence are preserved as part of the recoverable persistent state before any subsequent memory operation is allowed to proceed. Thus, ANCHOR avoids adding substantial extra fence overhead in that fence completion only requires older stores to reach the WCB, rather than propagate further to the SSD—whose latency is much higher than the local store-buffer-to-WCB path. For stores that are not ordered by additional synchronization primitives, ANCHOR imposes no extra persistence-ordering constraints beyond the the underlying memory consistency, *e.g.,* program-order insertion into the WCB under Intel Total Store Ordering (TSO) [77].

4.5 CXL.mem Interface and SSD Layout

4.5.1 Address Mapping. ANCHOR reserves a contiguous physical address range on the memory-semantic SSD as the CXL.mem region. This region is partitioned into (1) a large *persistent memory region*—sized to be at least as large as DRAM main memory to accommodate stores from the persistent path—and (2) a small *metadata region* which stores JIT checkpoints.

Note that ANCHOR does not maintain a DRAM–SSD mirror; instead, it maps architectural memory into a separate persistent address space on the SSD. Each DRAM physical page P is assigned a corresponding persistent page P' in the SSD region through a static, offset-based mapping known to both the WCB and the recovery controller. For a 64 B data at DRAM physical address A, the WCB computes the persistent address A' by adding the DRAM-to-SSD offset; the page offset and data offset within P are preserved. That is, stores drained through the WCB update the SSD region according to architecturally visible physical addresses; however, it would be a mistake to take this to mean that ANCHOR attempts to shadow cache contents or maintain a strict DRAM mirror. Finally, the underlying SSD controller's FTL maps persistent addresses to NAND

pages and handles wear leveling and garbage collection. Nevertheless, ANCHOR treats the CXL.mem region as a flat byte-addressable space and does not rely on any FTL-specific behavior.

4.5.2 CXL.mem Write Engine. The CXL.mem write engine is a small DMA-like unit that consumes 64 B updates from the WCB and issues them as CXL.mem write transactions. Each transaction carries the persistent address A' together with the ECC-checked 64 B payload. When the controller selects an entry of WCB to drain, the CXL.mem write engine (1) checks the ECC and, if necessary, corrects the 64 B data, and (2) forms and issues a 64 B CXL.mem write to the SSD address. Once the SSD acknowledges that the write has entered its persistent domain, the WCB entry is invalidated and becomes available for reuse.

4.6 Power Failure Recovery Protocol

Recovery assumes the fail-stop model described in Section 3; processor cores, caches, store buffers, and DRAM (main memory) all lose their state on power failure, whereas the WCB remains intact. On the SSD side equipped with memory-semantic storage support such in Samsung CMM-H, we assume that acknowledged CXL.mem writes have entered an SSD-side persistent domain protected by battery-backed power-loss protection (PLP), which includes device-internal buffering such as the internal DRAM cache [172, 173]. Under this assumption, any CXL.mem write acknowledged by the SSD is guaranteed to survive power failure—within the SSD-side persistent domain.

At any failure point, ANCHOR's recoverable state consists of three parts: (1) stores already acknowledged by the SSD, (2) stores buffered in the non-volatile WCB, and (3) any remaining committed stores that may still reside in the volatile store buffer. The first two parts already belong to the persistent domain, while the third part is captured by a just-in-time checkpoint.

4.6.1 JIT Checkpoint. Upon a power-fail signal, ANCHOR performs a just-in-time checkpoint that saves (1) the architectural state for each core—including the program counter (PC) and architectural registers—and (2) those committed stores that remain in the store buffer but have not yet been accepted by the WCB. Again, this checkpoint is written into the SSD's metadata region. It is important to mention that the amount of data captured by the just-in-time checkpoint is quite small in the common case. That is because ANCHOR moves committed stores into the non-volatile WCB as early as possible.

4.6.2 Recovery Sequence. After power is restored, the system boots into a small recovery firmware that runs before the OS. The recovery sequence proceeds as follows:

(1) **Quiesce devices and locate checkpoint.** The recovery firmware first quiesces external devices (*e.g.,* blocks DMA) so that no device-side activity can overwrite recovered memory state. It then reads the JIT checkpoint from a small reserved *metadata region* on the SSD.
(2) **Finalize the persistent image.** The firmware first drains the WCB entries to the SSD's *persistent memory region* using the write masks. It then replays the committed stores saved in the JIT checkpoint, corresponding to the short tail of committed stores that had not yet reached the WCB at power failure time. Together, these two steps reconstruct the persistent state of the latest recoverable committed prefix.
(3) **Rebuild the memory image.** The firmware reconstructs DRAM from the SSD's *persistent memory region* by streaming data from the SSD into DRAM, optionally skipping regions known to be unused (*e.g.,* free physical pages).
(4) **Restore architectural state.** The firmware restores the checkpointed architectural state, including the PC, into the cores.
(5) **Resume execution.** Control is transferred to the OS' resume entry point—rather than the cold-boot path—so that boot-time memory initialization does not overwrite the recovered DRAM (main memory).

Note that this recovery procedure requires no software logs or application modifications. Existing hibernation/resume support can be reused when available; otherwise, a small boot hook is sufficient to bypass cold-boot memory initialization during recovery. From the programmer's perspective, execution resumes from the latest recoverable committed prefix.

4.7 Soft Error Resilience Mechanisms

4.7.1 Lightweight Protection for Caches and DRAM. In ANCHOR, caches and DRAM use existing SECDED/ChipKill check bits in detect-only mode (optionally complemented by CRC on data paths) without performing on-the-fly correction. When corrupted data is detected in a cacheline or in DRAM, the faulty data is invalidated, and the system repairs it by reloading the corresponding clean data from the SSD-backed committed image.

To ensure correct repair, ANCHOR first drains the WCB before reloading data from the SSD, ensuring that it reflects the most up-to-date state. This approach avoids performing repair directly from the WCB, which would otherwise require additional data-read paths from the caches and DRAM, complicating the design. Instead, the system reloads the clean committed value from the SSD and reinstalls the corrected copy into the volatile hierarchy.

As a result, ANCHOR expands the effective *repairable* error space to the broader class of faults by operating the deployed cache and DRAM ECCs in the detection-only mode. This design keeps the volatile hierarchy lightweight on the critical path with the data path performance maintained. Specifically, caches and DRAM are only responsible for fault detection, whereas correction is achieved by restoring the latest committed value from the SSD.

4.7.2 WCB Protection. Although the WCB is small (32 kB per core), it sits directly on the persistent path and temporarily buffers recently committed stores before they reach the SSD. Any silent corruption in the WCB could therefore propagate into the SSD-resident committed state if left unprotected. The potential sources of the faults include both transient soft errors and other bit flips caused by retention failure.

To protect against these faults, ANCHOR implements the WCB in STT-RAM that is more resilient to soft errors (Section 4.3) and possesses higher write endurance compared to other non-volatile technologies. Given the WCB's small capacity and the improved resilience of STT-RAM, a modest per-entry code is sufficient in practice; each 64 B WCB entry is protected with SECDED over its data field. On every read or drain, ANCHOR checks SECDED and, if

necessary, repairs any correctable errors before writing data to the SSD. Rare uncorrectable multi-bit errors in the WCB are treated as fatal faults outside our target model. In the common case, ANCHOR ensures that only correct data is committed to the SSD.

4.7.3 SSD as the Global Correction Anchor. The memory-semantic SSD employs strong LDPC/BCH codes over multi-kilobyte flash pages, enabling robust protection of the SSD-resident committed image. ANCHOR treats this committed image as a *global correction anchor*; after an entry is drained from the WCB into the SSD's persistent region, subsequent repairs are simplified by refetching the trustworthy value from the SSD. Together, these mechanisms establish a simple trust chain. That is, caches and DRAM detect faults, the WCB protects in-flight persistent updates, and the SSD serves as the final correction anchor for all committed data. This allows ANCHOR to repair detected volatile-memory faults without requiring full local correction throughout the volatile hierarchy.

4.8 Software and OS Support

ANCHOR is designed to be transparent to applications and requires only minimal OS and firmware support below:

- **Address-space configuration.** At boot time, firmware configures the CXL.mem region, reserves a contiguous physical address range for the SSD-backed persistent region and metadata, and informs the OS of the DRAM and persistent layout.
- **Page mapping.** The OS ensures that each DRAM physical page has a corresponding location in the SSD persistent region, but this mapping is static and does not require changes to page-fault handling or the virtual memory subsystem.
- **Recovery integration.** A small recovery firmware or kernel module implements the recovery described earlier (Section 4.6); this runs before the normal OS boot sequence and is invoked only after power failure.

The upshot is that no changes are required to application binaries, compilers, or user-level APIs. From the software perspective, ANCHOR preserves the familiar abstraction of DRAM-based main memory while requiring only minimal OS and firmware support.

5 Evaluation

5.1 Methodology

We conduct all experiments using gem5 [13], configured to model an 8-core Intel Skylake-X-like processor (one hardware thread per core) with two memory controllers (MCs). We parameterize the memory-semantic SSD using the median read/write latencies and bandwidth reported in prior CMM-H characterization [173]. To explicitly model device-internal cache behavior, we further model the SSD's internal DRAM cache as a 4 GB, 8-way LRU, write-back cache with 4 KB cachelines [173]; each DRAM-cache miss incurs a 20 µs backend service latency [95]. Our default WCB configuration is a 32 KB, 4-way set-associative buffer (128 sets) with 64 B lines, and its access latency is derived from DESTINY [119]. The detailed system configuration is summarized in Table 1.

This section evaluates ANCHOR using both single-threaded (SPEC CPU 2006 and 2017 [16, 66]) and multi-threaded (SPLASH3 [140], NPB-CPP [109], STAMP [117], and WHISPER [92, 122]) benchmarks. All SPLASH3, NPB, STAMP, and WHISPER benchmarks are run in gem5's full-system mode with Linux kernel 5.4.46. For SPEC CPU2006/2017, we fast-forward 10 billion instructions and then simulate the subsequent 5 billion instructions. On the other hand, for SPLASH3, NPB, STAMP, and WHISPER, we simulate the entire program execution.

Table 1: System configuration.

OS	Ubuntu 18.04 with Linux kernel 5.4.46
Processor	8-core, 4-width OoO, 2 GHz ROB/IQ/SQ/LQ: 224/97/56/72
L1 I-cache	32 KB/core, 8-way, 64 B block, 3 cycles
L1 D-cache	64 KB/core, 8-way, 64 B block, 4 cycles
L2 cache	16 MB shared, 16-way, 64 B block, inclusive, 44 cycles
DRAM	4 GB, DDR4-2400 8×8
Memory controller	2 MCs, 2 channels/MC
Persist path (CXL.mem)	24 GB/s [173]
Write combining buffer (WCB)	32 KB, 4-way, LRU, hit/miss latency: 1.947/1.314 ns [119], write latency: 4.678 ns [119]
Semantic SSD timing/bandwidth	read/write latency: 57.6 ns / 16 ns (DRAM-cache hits) read/write bandwidth: 4 GB/s / 2 GB/s [173]
The SSD's internal DRAM cache	4 GB, 8-way LRU, 4 KB cacheline, write-back
Backend NAND	20 µs backend service latency on DRAM-cache misses [95]

Our baseline is a conventional system with the same configuration, providing full ECC protection for all caches and DRAM but lacking persistence support (*baseline*). We do *not* add any extra ECC latency to *baseline* as the Skylake-X cache and DRAM latencies are already measured with ECC enabled. Separately, we evaluate two conservative strengthened-ECC baselines (sECC) that approximate stronger ECC—such as double-error correction and triple-error detection (DECTED)—by modeling an additional 1-cycle latency on memory accesses. Specifically, *baseline-sECC-All* adds +1 cycle to all cache levels and DRAM, whereas *baseline-sECC-L2M* adds +1 cycle only to L2 and DRAM. Unless otherwise specified, all reported results are normalized to *baseline*.

5.2 Performance Evaluation

Figure 2 reports normalized run time across benchmarks from SPEC CPU 2006, SPEC CPU 2017, STAMP, NPB, SPLASH3, and WHISPER. Averaged over all 42 benchmarks, *baseline-sECC-All* increases run time by 9.75% relative to *baseline*, whereas *baseline-sECC-L2M* and ANCHOR incur only 0.97% and 4.36% overhead, respectively. The takeaway is that ANCHOR not only provides whole-system persistence but also strengthens soft error resilience by repurposing existing ECC check bits for detection-only operation—that yields deterministic detection of 3-bit errors—and offering their correction without adding extra ECC latency; nevertheless, ANCHOR achieves about a 2× lower slowdown than a conventional design that strengthens ECC by uniformly lengthening the critical path throughout the memory hierarchy.

For most SPEC CPU2006/2017, STAMP, and WHISPER applications, ANCHOR incurs less than 5% run-time overhead. This demonstrates that the persistent path and WCB impose minimal performance penalty for common single-threaded and moderate-scale multi-threaded workloads. The remaining overhead is concentrated in a small subset of memory-intensive parallel benchmarks in NPB and SPLASH3, where high write intensity to large shared data structures and frequent synchronization generate a sustained stream of persistent updates. While the WCB still achieves high

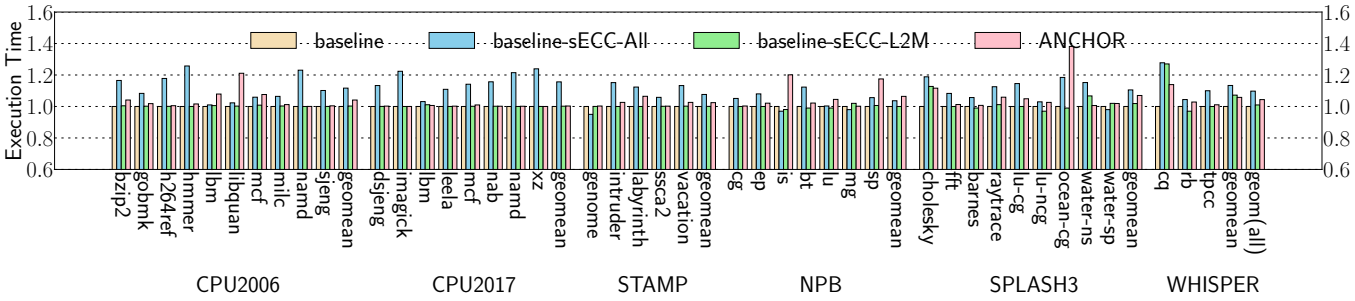


Figure 2: Execution time of baseline, baseline-extra, and ANCHOR.

merge rates for most workloads (Section 5.3), the sheer volume of writes frequently fills the WCB and forces it to drain entries before admitting new stores. This exerts backpressure on the store buffer, introducing short core pipeline stalls that are amplified around synchronization points. That is why these worst-case workloads see higher run-time overhead. Nonetheless, ANCHOR remains competitive relative to both *baseline* and *baseline-sECC-All* while simultaneously providing whole-system persistence and stronger soft error resilience.

5.3 WCB Merge Behavior

This section analyzes the behavior of ANCHOR's write-combining buffer (WCB) along two dimensions: (1) the *merge rate*, *i.e.,* how often incoming persistent stores hit an existing WCB entry instead of allocating a new one, and (2) the *packing efficiency* within each 64 B WCB entry, measured as the number of valid 8 B sub-blocks when the entry is drained.

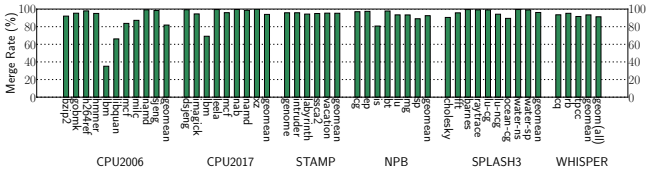


Figure 3: Merge rate of WCB across benchmarks.

We define the WCB *merge rate* as the fraction of WCB accesses that hit an already-allocated entry (WCB hits) rather than allocating a new one. A high merge rate indicates that consecutive persistent stores frequently target the same cacheline, allowing ANCHOR to coalesce them in the WCB and issue a single 64 B persistence write instead of multiple smaller updates. As shown in Figure 3, most benchmarks achieve a high merge rate—91.23% on average—indicating sufficient temporal and/or spatial locality for the WCB to aggregate persistent updates effectively. This high merge rate substantially reduces the number of 64 B lines sent to the SSD and helps explain why ANCHOR incurs only modest run-time overhead for most workloads. Benchmarks exhibiting noticeably lower merge rates tend to create higher pressure on Write Combining Buffer (WCB) capacity. This exerts backpressure on the store buffer, potentially stalling the core pipeline and thus leading to somewhat higher overhead—as observed in *lbm* and *libquantum* in SPEC CPU2006, *is* in NPB, and *ocean-config* in SPLASH3. However, this correlation is not absolute; it is also influenced by other factors such as synchronization intensity and overall memory access patterns.

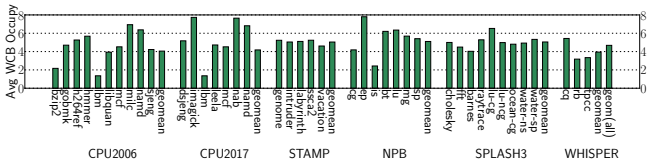


Figure 4: Average WCB occupancy across benchmarks.

To further quantify how effectively each WCB entry is utilized, we also measure the average number of valid 8 B sub-blocks per 64 B entry at eviction time. A 64 B entry can hold up to eight 8 B sub-blocks; if only one sub-block is valid, the write-combining opportunity is essentially lost, whereas multiple valid sub-blocks indicate that several logically separate stores have been coalesced into the same physical write. As shown in Figure 4, the average number of valid sub-blocks per 64 B entry is 4.71 across those benchmarks that experience evictions. In other words, each persistence write typically amortizes about four to five 8 B stores, which substantially reduces the SSD write traffic compared to a naive design that would flush each store individually. The SPEC CPU2017 benchmark *xz* is a special case: it achieves a 100% merge rate within our simulation window and does not trigger any WCB evictions, so its occupancy cannot be measured and is therefore excluded from Figure 4 and from the overall average.

5.4 Sensitivity Analysis

This section studies the sensitivity of ANCHOR to several key parameters of the persistent path. Unless otherwise noted, all results are normalized to the default ANCHOR configuration and averaged across the evaluated benchmarks. Our goal here is to understand which design knobs materially affect performance and persistence traffic, and which ones offer only marginal benefits beyond the baseline design.

5.4.1 WCB Size. The WCB capacity determines how many distinct cachelines can accumulate persistent updates in parallel. A larger WCB increases the probability that temporally related stores hit existing entries while reducing the eviction frequency of partially populated entries. A smaller WCB, by contrast, is more likely to evict partially populated lines, generating extra 64 B persistence

writes. Furthermore, when a smaller WCB exhausts its free entries, it propagates backpressure through the persistent path to the store buffer, potentially inducing pipeline stalls.

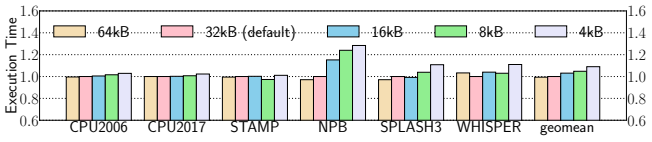


Figure 5: Execution time under different WCB sizes.

Figure 5 reports the performance trend—normalized to the default 32 kB design—when the WCB size varies from 4 kB to 64 kB. On average, shrinking the WCB to 16 kB, 8 kB, and 4 kB increases run time by 3.1%, 4.9%, and 9.0%, respectively, whereas enlarging it to 64 kB reduces run time by 1.3%. The impact is most pronounced for NPB and SPLASH3; a 4 kB WCB slows them down by 28.4% and 10.8%, but a 64 kB WCB turns this into a 2.9% speedup for both suites. For WHISPER, performance is largely stable across 64–8 kB WCBs, with 32 kB performing slightly best and only the 4 kB design showing clear degradation.

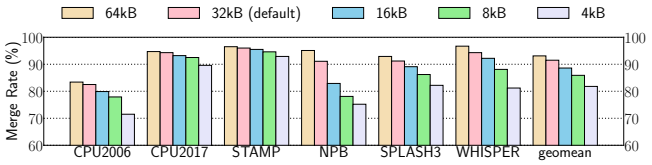


Figure 6: Merge rate under different WCB sizes.

Figure 6 then shows the global merge rate across different WCB sizes. As the WCB size grows from 4 kB to 64 kB, the average merge rate increases from 81.8% to 93.1%, closely mirroring the performance changes. That is, larger WCBs both coalesce more stores into each 64 B persistence write and reduce the frequency of WCB-induced backpressure. It turns out that overall, a 32 kB WCB per core strikes a good balance; making it much smaller amplifies persistent traffic and commit stalls for highly write-intensive parallel workloads, while making it substantially larger offers only modest additional benefit beyond the 32 kB configuration.

5.4.2 WCB Set Associativity. Beyond capacity, the WCB set associativity determines how many distinct cachelines mapping to the same index can accumulate persistent updates without evicting one another. Figure 7 reports the normalized run time varying associativity from 2-way to 16-way, while fixing the WCB capacity at 32 kB and using the 4-way design as the baseline. Overall, all configurations stay within about 5% of the 4-way design with 4-way consistently delivering the lowest run time. Lower associativity (2-way) introduces more conflict-induced evictions, while higher associativity (8- and 16-way) increases WCB lookup latency due to the additional tag-comparison overhead (modeled via DESTINY [119]). Moreover, beyond 4-way, the merging benefits largely saturate, so further increasing associativity mainly changes set-level eviction and drain behavior rather than improving coalescing itself. Our evaluation shows that the 4-way design incurs fewer evictions than the 16-way design, indicating that it relieves set pressure earlier and more effectively. This, in turn, reduces backpressure on the store buffer and leads to lower run time. On the other hand, the slowdowns are most visible in NPB and WHISPER, indicating that additional associativity beyond 4-way does not translate into better end-to-end performance.

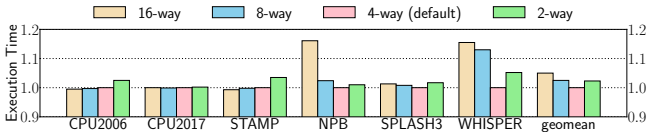


Figure 7: Execution time across WCB associativities.

Figure 8 then reports the global merge rate across different WCB ways. The merge rate is already near-saturated around the 4-way design; 2-way clearly loses merges due to more frequent conflict-induced evictions of partially filled entries, while 8-way and 16-way offer only marginal changes relative to 4-way. Thus, once the merge rate has largely saturated, the remaining performance differences are dominated not by additional coalescing opportunities but by lookup cost and set-level drain dynamics. Overall, a 4-way set-associative WCB strikes a practical balance between implementation complexity and effectiveness, avoiding (1) the conflict penalties of low associativity without incurring the extra lookup overhead and (2) less favorable pressure-relief behavior of very high associativity.

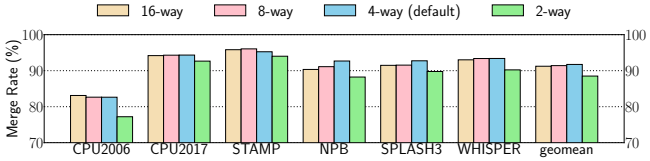


Figure 8: WCB merge rate under different associativities.

5.4.3 WCB Granularity. This section compares the default 64 B-granularity WCB against an 8 B-entry design. The 8 B entry is picked here over 16/32 B because committed stores leave the store buffer at 8 B granularity. In the 8 B design, each entry holds only one 8 B chunk; yet CXL.mem still transfers 64 B data, so each update ultimately becomes a padded 64 B write. As a result, many 64 B writes carry only a single valid 8 B chunk, whereas our 64 B-line WCB packs an average of 4.71 distinct 8 B sub-blocks per entry (Section 5.3), substantially reducing write traffic to the SSD.

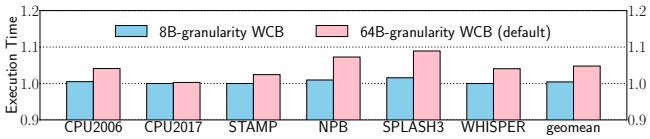


Figure 9: Execution Time under different WCB granularities.

To give the 8 B organization the best possible chance, we sweep its associativity from 2-way up to 256-way, including a fully associative design. We find that a 64-way 8 B-entry WCB achieves both the highest merge rate and the best performance among all

8 B configurations; we therefore use this 64-way 8 B design for comparing against our 4-way 64 B WCB. Figure 9 shows that the 8 B design delivers slightly better performance, because a 32 kB 8 B-granularity WCB contains more entries and thus experiences less capacity pressure than a 32 kB 64 B-line WCB.

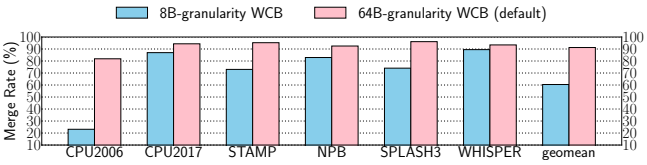


Figure 10: Merge rate under different WCB granularities.

However, as shown in Figure 10, its merge rate is 33.77% lower than that of our default 64 B design. In particular, for SPEC CPU2006, *libquantum* exhibits only a 0.0025% merge rate, while *lbm*, *mcf*, and *milc* also show much lower merge rates. This result is not contradictory; execution time is primarily governed by front-end WCB capacity pressure and the resulting backpressure on store retirement, whereas the lower merge rate of the 8 B design mainly increases back-end SSD write traffic. As a result, the 8 B configuration can generate significantly higher write traffic to the SSD than the 64 B design, so it is not our preferred design point despite its slightly better performance.

Moreover, the 64-way 8 B-entry organization is significantly more expensive to implement. At the same 32 kB capacity, DESTINY [119] shows that it nearly doubles total WCB area, driven primarily by a 2.9× increase in tag-array area. This enlarged tag structure also increases energy and leakage, incurring 5.4× higher hit dynamic energy (0.654 nJ vs. 0.121 nJ) and 1.8× higher leakage (13.275 mW vs. 7.493 mW). Overall, our 4-way 64 B design provides the better design point by balancing performance, SSD write pressure, hardware complexity, and energy consumption.

5.4.4 CPU Microarchitectures. This section tests how ANCHOR behaves across different CPU microarchitectures. Our default configuration models a Skylake-like out-of-order core; we additionally consider a GoldenCove-like core with a wider pipeline and larger structures (*e.g.,* ROB, load/store queues, register file). All memory-system and persistent-path parameters remain unchanged. For each microarchitecture, we normalize the run time to its own baseline configuration without ANCHOR.

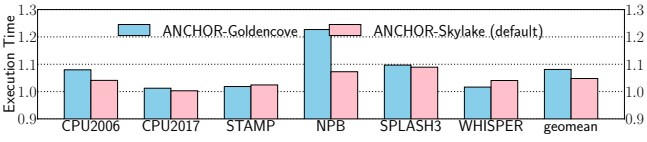


Figure 11: Execution time under different microarchitectures.

As shown in Figure 11, ANCHOR incurs modest overhead on both cores, but the GoldenCove-like core sees a somewhat higher slowdown (about 8.10% versus 4.36% on Skylake). For SPEC CPU2017, STAMP, SPLASH3, and WHISPER, the overhead stays in the low single digits on both microarchitectures, with only small differences between them. The gap is driven mainly by the most memory-intensive workloads in SPEC CPU2006 and NPB, where the stronger core shortens the baseline run time with the same persistent-path configuration that was tuned for Skylake, making the relative cost of persistence more visible. Overall, ANCHOR remains effective across different out-of-order core designs; once the persistent path is provisioned, upgrading the CPU core primarily scales the baseline performance, while the relative overhead of persistence stays moderate even on a more aggressive microarchitecture.

Table 2: DRAM-cache hit rates under stress.

Suite	Hit Rate	Suite	Hit Rate	Suite	Hit Rate
SPEC CPU 2006	79.77%	SPEC CPU 2017	76.58%	STAMP	73.19%
NPB	95.14%	SPLASH-3	69.72%	WHISPER	97.84%

5.4.5 CPU Clock Frequency. This section tests the sensitivity of ANCHOR to CPU frequency by increasing the clock from the default 2 GHz to 3 GHz and 4 GHz with the store buffer size unchanged. Higher clock frequencies can increase the effective cycle cost of WCB access and therefore potentially increase persistence overhead. On the other hand, high-frequency CPU designs often provision deeper buffering structures, such as larger store buffers, which can help alleviate the WCB-induced backpressure on a store buffer—leading to less buffer overflows and the resulting pipeline stalls.

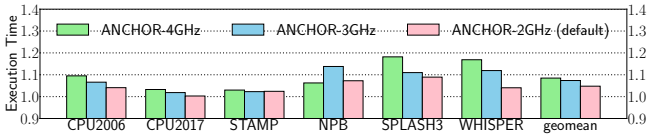


Figure 12: Execution time under different clock frequencies.

Nevertheless, as shown in Figure 12 in which each frequency is normalized to its corresponding baseline, even with an unchanged store buffer size, ANCHOR shows limited sensitivity to CPU frequency; the average overhead is 7.37% at 3 GHz and 8.48% at 4 GHz.

5.4.6 DRAM-Cache Pressure Sensitivity. To examine the sensitivity of ANCHOR to increased DRAM-cache pressure, this study substantially reduces the SSD's internal DRAM-cache capacity from the default 4 GB setting so that it becomes smaller than the typical memory footprints of the target workloads. Specifically, the cache is reduced to 128 MB for SPEC CPU 2006/2017 workloads (memory footprint ≈ 90–800 MB), 16 MB for NPB and WHISPER (memory footprint ≈ 50 MB), 10 MB for SPLASH-3 (memory footprint ≈ 10–40 MB), and 4 MB for STAMP (memory footprint ≈ 4–8 MB).

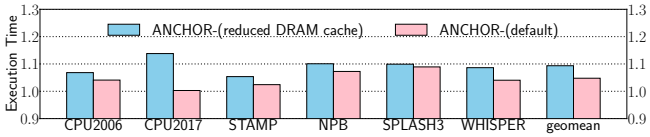


Figure 13: Execution time under reduced DRAM-cache size.

Figure 13 shows that ANCHOR remains robust even under such intentionally stressed cache settings, incurring only about 5% additional performance overhead compared with the default setting, *i.e.,* 9.35% vs. 4.36%. Overall, this result indicates that ANCHOR is not inherently sensitive to moderate increases in DRAM-cache pressure by its size reduction.

There are 3 reasons for this behavior. First, ANCHOR decouples the core-side critical path from backend SSD persistence through the WCB; the store buffer—off the critical path most of the time in modern processors—releases committed stores as soon as they are accepted by the WCB, while the SSD write proceeds asynchronously. Moreover, WCB coalescing substantially reduces downstream SSD traffic as shown in Figure 3. Second, DRAM-cache hit rates remain substantial even under the stressed settings, ranging from 69.72%o 97.84% across all benchmark suites, as shown in Table 2. Although some suite-level averages fall below 90%, these are mainly driven by one or two outlier benchmarks with hit rates below 10%; most benchmarks in the same suite still exceed 90%. Third, DRAM-cache misses do not directly translate to core-visible performance loss. Miss-induced pressure must first accumulate at the device-side PLP-protected acceptance path, then backpropagate to WCB occupancy; only after WCB pressure becomes sufficiently high enough to have the store buffer clogged, the core pipeline starts stall. As a result, even under reduced DRAM-cache capacity, the backend pressure caused by additional DRAM-cache misses does not directly translate to substantial performance degradation. If future systems experience substantially higher backend pressure, a larger WCB would provide a straightforward way to absorb more transient pressure before it propagates back to the store buffer of the core.

5.5 Error Repair Coverage

This section compares the error-repair coverage of the baseline and ANCHOR across fault classes. The key distinction is whether a fault is (1) locally correctable, (2) detectable but not locally correctable, or (3) undetected. In the baseline, repairability is bounded by the native local-correction capability of each volatile memory hierarchy level. In ANCHOR, faults in the second class become repairable because the corrupted volatile data can be discarded and restored from the SSD-backed committed image.

For caches, ANCHOR reuses the existing SECDED check bits in detect-only mode, in which case it offers deterministic detection of up to 3-bit faults in a protected codeword—though its native local-correction capability remains limited to single-bit faults; see Table 3 for representative cache-side examples. In the baseline, SECDED offers local correction for 1-bit faults and detection for 2-bit faults, but faults beyond its local-correction capability cannot be repaired once they corrupt the cache data. In ANCHOR, the same check bits serve as the front-end detector only, while the SSD-backed committed image serves as the repair anchor. As a result, the detected cache faults—covering a broader space than when the check bits are used for both detection and correction—become repairable via restoration rather than local ECC correction, thereby improving the resilience. For example, assuming a uniform distribution over bit-flip patterns within a 64-bit protected word, ANCHOR expands the guaranteed repairable cache-side pattern space from $C(64, 1)$ in the baseline to $C(64, 1) + C(64, 2) + C(64, 3)$, achieving a 683.5× increase.

For DRAM, prior work classifies failures into modes such as single-bit, single-word, single-column, single-row, single-bank, multiple-bank, and multiple-rank faults [149]. In the baseline, repair is limited to the subset of these modes that the deployed ChipKill organization can correct locally. Detectable fault modes outside that envelope remain unrecoverable once they appear in DRAM. In ANCHOR, any DRAM fault mode that remains detectable by the front-end protection becomes repairable, because the corrupted volatile data can be discarded and reconstructed from the SSD-backed committed image. Thus, ANCHOR extends DRAM-side repair coverage from locally correctable ChipKill fault modes to the broader set of detectable DRAM fault modes.

Table 3: Repair coverage under different fault classes.

Fault class	Baseline	ANCHOR
Cache: 1-bit fault	Local correction	Repairable
Cache: 2-bit fault	Detect only	Repairable
Cache: 3-bit fault	Unrecoverable	Repairable
Cache: undetected fault	Unrecoverable	Unrecoverable
DRAM: locally correctable ChipKill fault modes	Local correction	Repairable
DRAM: detectable but locally uncorrectable fault modes	Unrecoverable	Repairable
DRAM: undetected fault modes	Unrecoverable	Unrecoverable

The key implication is that ANCHOR does not improve soft error resilience by strengthening local correction at every volatile memory level. Instead, it changes the repair criterion itself. In the baseline, repairability is bounded by local correction capability. In ANCHOR, repairability is bounded by front-end detectability together with the availability of the SSD-backed committed image. As a result, ANCHOR converts detectable but locally uncorrectable faults from unrecoverable to repairable.

5.6 Energy Consumption

This section estimates ANCHOR's energy overhead based on a component-based model that separates conventional system energy from the additional energy introduced by the persistence path. Baseline energy consists of processor-core energy and memory energy, where the latter includes both the on-chip cache hierarchy and DRAM. Processor-core and on-chip-cache energy are estimated with McPAT [97], while DRAM energy is obtained from gem5. Baseline energy is therefore computed as:

$$E_{\text{baseline}} = E_{\text{core}} + E_{\text{memory}}. \tag{1}$$

For ANCHOR, we additionally include both the energy of the WCB—which is modeled using DESTINY [119]—and that of the memory-semantic SSD:

$$E_{\text{ANCHOR}} = E_{\text{core}} + E_{\text{memory}} + E_{\text{WCB}} + E_{\text{SSD}}. \tag{2}$$

Each incoming persistent store performs one lookup. On a hit, the corresponding entry is updated; on a miss, a new entry is allocated. Accordingly, the WCB energy is computed as:

$$E_{\text{WCB}} = N_{\text{lookup}} \cdot E_{\text{lookup}} + (N_{\text{hit}} + N_{\text{insert}}) \cdot E_{\text{write}} + P_{\text{leak}} \cdot T_{\text{exec}}. \tag{3}$$

We model the memory-semantic SSD energy using a first-order event-based formulation that accounts for internal DRAM-cache accesses, backend SSD read/write operations, and SSD standby power. Internal DRAM-cache access energy is derived from CACTI [148], while backend SSD read/write energy and standby power are parameterized from an enterprise SSD datasheet [142]; see Table 4 for the parameters. Finally, the SSD energy is computed as:

$$E_{\text{SSD}} = E_{\text{DRAM Cache}} + E_{\text{SSD R/W}} + P_{\text{standby}} \cdot T_{\text{exec}}. \tag{4}$$

Figure 14 shows the energy breakdown of the baseline (B) and ANCHOR (A). On average, ANCHOR increases total energy by

Table 4: Energy parameters.

Component / Event	Value
WCB lookup energy	0.121 nJ/access [119]
WCB write energy	0.048 nJ/access [119]
WCB leakage power	7.493 mW [119]
SSD DRAM-cache read energy	9.30 nJ/access [148]
SSD DRAM-cache write energy	9.34 nJ/access [148]
SSD backend read energy	16.38 μJ/access [142]
SSD backend write energy	114.29 μJ/access [142]
SSD standby power	3.5 W [142]

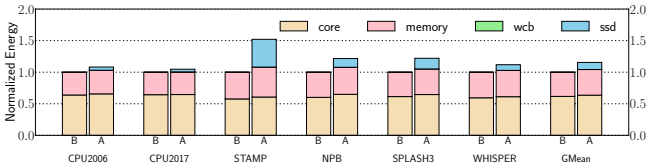


Figure 14: Energy breakdown of the baseline and ANCHOR. B denotes the baseline, and A denotes ANCHOR.

15.46%, mostly due to the SSD, whereas the WCB contributes negligibly. Note that the result is conservative (over-estimated), since we do not model the ECC-related energy savings enabled by ANCHOR.

5.7 Hardware Cost Analysis

ANCHOR adds a small per-core structure on the persistent path, *i.e.,* a STT-RAM write-combining buffer (WCB). In our default configuration, the WCB is organized as a 4-way set-associative array with 512 entries of 64 B data each (32 kB), plus per-entry metadata including a physical-line tag, a valid bit, an 8 B-word write-valid mask, and lightweight ECC bits for the 64 B line. This metadata contributes roughly another 6 kB, so the WCB occupies about 38 kB per core, which is small compared to the existing L1/L2 capacity. We estimate the area of this structure using CACTI 7.0 [10] at 22 nm. The WCB occupies about 0.084 mm^2 per core. Compared to an Intel Xeon Skylake server core (11.85 mm^2 excluding the shared L2 cache), as obtained from McPAT [97], this corresponds to roughly 0.7% per-core area overhead.

6 Related Work

Many schemes have been proposed to provide data persistence and crash consistency. A large body of the work focuses on *partial-system persistence (PSP)*, where only user-defined regions of code or data structures are made persistent [2, 7, 12, 19, 38, 50, 60, 67, 75, 77, 82, 100, 106–108, 113, 114, 124, 125, 131, 132, 139, 145, 160, 161, 163, 164, 179]. These approaches expose programming interfaces such as transactions or region annotations; while flexible, they are error-prone and often rely on write-ahead logging (WAL) plus flushes and fences on the commit critical path, which can stall CPUs and limit scalability. To reduce this overhead, prior work proposes hardware-assisted WAL that overlaps logging with subsequent execution [53, 78, 81, 126, 147]. Battery-backed schemes such as eADR and related designs [2, 7, 41, 60, 72, 79, 82, 123, 161] avoid logging by flushing volatile state on power loss, but typically require substantial energy storage or intrusive changes to cache/coherence mechanisms, thus significantly complicating deployment.

Motivated by these limitations, many recent efforts have targeted *whole-system persistence (WSP)* [3, 28–33, 43–46, 63, 64, 76, 79, 101, 123, 159, 167, 168, 170, 174, 181, 182], transparently persisting all software state—including the OS and libraries. For example, Flush-on-Fail [123] and LightPC [101] preserve volatile state through power-failure-triggered JIT checkpointing, but both are fundamentally designed so that persistent memory itself acts as main memory. Thus, unlike ANCHOR, they cannot be easily adopted in real-world systems where DRAM remains the main memory—much faster than any commodity persistent memory. Capri [79] avoids programming burden and the need to preserve large volatile states through software/hardware co-design, but is also built around an SCM-oriented persistent-memory substrate and incurs considerable performance overhead due to its complex redo + undo logging mechanism.

Several PSP and WSP schemes also employ a dedicated persist path to deliver data to persistent memory [2, 60, 77, 79, 161, 174, 182]. In many designs, the path originates at private caches (*e.g.,* L1) and propagates 64 B cache lines, so each 8 B store can inflate into a 64 B transfer, increasing bandwidth demand. In contrast, ANCHOR taps the committed-store stream at the store buffer and forwards only 8 B updates into a compact per-core WCB, which merges multiple stores to the same line into a single 64 B write before flushing it over CXL. This organization preserves a narrow on-chip path while avoiding cache-line write amplification.

Beyond crash consistency, prior work extensively studies reliability under soft errors and silent data corruption (SDC) [54, 70, 71, 86, 102–105, 111, 133, 167, 169, 171, 178], spanning system-level characterization, detection, and recovery mechanisms [8, 11, 24, 26, 36, 42, 49, 59, 65, 71, 96, 99, 129, 130, 138, 158]. In a complementary security setting, Soteria [183] hardens integrity-protected and encrypted NVMs by protecting security metadata (*e.g.,* integrity structures) against memory faults. These efforts are orthogonal to ANCHOR; Soteria focuses on encrypted-NVM metadata protection, whereas ANCHOR targets WSP over CXL-attached memory-semantic SSDs and leverages an SSD-resident anchor to enhance resilience without core pipeline change or heavy software instrumentation.

7 Conclusion

This paper presents ANCHOR, a CXL-based memory hierarchy that provides both whole-system persistence and soft error resilience without compiler instrumentation or core microarchitectural changes. Through the CXL.mem interface, ANCHOR sends all committed stores to a memory-semantic SSD, while coalescing them with its STT-RAM-based write combining buffer to reduce the traffic down to the SSD. More importantly, ANCHOR leverages the SSD-resident committed state not only for persistence, but also as a correction anchor for resilience. This allows caches and DRAM to focus on error detection, while correction is shifted to the persistent domain via SSD-assisted repair. Taken together, these techniques enable both whole-system persistence and enhanced caches/DRAM error resilience at low cost, demonstrating that persistence and resilience can be effectively unified in future CXL-enabled systems.

Acknowledgments

We thank the anonymous reviewers for their valuable comments. This work was supported in part by NSF Grants 2333645, 2314681, and 2153749, and by a gift from Samsung.

References

[1] 2022. *Samsung Memory-Semantic CXL SSD at FMS 2022 Powered by AMD-Xilinx*. https://www.servethehome.com/samsung-memory-semantic-cxl-ssd-at-fms-2022-powered-by-amd-xilinx/ Accessed: 2025-12-09.

[2] Ahmed Abulila, Izzat El Hajj, Myoungsoo Jung, and Nam Sung Kim. 2022. ASAP: architecture support for asynchronous persistence. In *Proceedings of the 49th Annual International Symposium on Computer Architecture*.

[3] Khakim Akhunov, Kasim Sinan Yildirim, Jongouk Choi, and Changhee Jung. 2025. Adaptive Computing in Memory Meets Conventional Batteryless Platforms. *ACM Transactions on Embedded Computing Systems* (2025).

[4] Hiroyuki Akinaga and Hisashi Shima. 2010. Resistive random access memory (ReRAM) based on metal oxides. *Proc. IEEE* 98, 12 (2010).

[5] Irina Alam, Clayton Schoeny, Lara Dolecek, and Puneet Gupta. 2018. Parity++: Lightweight error correction for last level caches. In *2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*. IEEE.

[6] Alaa R Alameldeen, Ilya Wagner, Zeshan Chishti, Wei Wu, Chris Wilkerson, and Shih-Lien Lu. 2011. Energy-efficient cache design using variable-strength error-correcting codes. *ACM SIGARCH Computer Architecture News* 39, 3 (2011).

[7] Mohammad Alshboul, Prakash Ramrakhyani, William Wang, James Tuck, and Yan Solihin. 2021. Bbb: Simplifying persistent programming using battery-backed buffers. In *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE.

[8] Abdul Rehman Anwer, Guanpeng Li, Karthik Pattabiraman, Michael Sullivan, Timothy Tsai, and Siva Kumar Sastry Hari. 2020. Gpu-trident: efficient modeling of error propagation in gpu programs. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE.

[9] Zahra Azad, Hamed Farbeh, Amir Mahdi Hosseini Monazzah, and Seyed Ghassem Miremadi. 2016. An efficient protection technique for last level STT-RAM caches in multi-core processors. *IEEE transactions on parallel and distributed systems* 28, 6 (2016).

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

[11] Leonardo Bautista Gomez and Franck Cappello. 2014. Detecting silent data corruption through data dynamic monitoring for scientific applications. *ACM SIGPLAN Notices* 49, 8 (2014).

[12] Abhishek Bhattacharyya, Abhijith Somashekhar, and Joshua San Miguel. 2022. NvMR: non-volatile memory renaming for intermittent computing. In *Proceedings of the 49th Annual International Symposium on Computer Architecture*.

[13] Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R Hower, Tushar Krishna, Somayeh Sardashti, et al. 2011. The gem5 simulator. *ACM SIGARCH computer architecture news* 39, 2 (2011).

[14] Bryan Black, Murali Annavaram, Ned Brekelbaum, John DeVale, Lei Jiang, Gabriel H Loh, Don McCaule, Pat Morrow, Donald W Nelson, Daniel Pantuso, et al. 2006. Die stacking (3D) microarchitecture. In *2006 39th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'06)*. IEEE.

[15] Raj Chandra Bose and Dwijendra K Ray-Chaudhuri. 1960. On a class of error correcting binary group codes. *Information and control* 3, 1 (1960).

[16] James Bucek, Klaus-Dieter Lange, and Jóakim v. Kistowski. 2018. SPEC CPU2017: Next-generation compute benchmark. In *Companion of the 2018 ACM/SPEC International Conference on Performance Engineering*.

[17] Steffen Buch. 2023. Error detecting and correcting codes for dram functional safety. In *Proceedings of the International Symposium on Memory Systems*.

[18] Geoffrey W Burr and Paul Franzon. 2014. Storage class memory. *Emerging Nanoelectronic Devices* (2014).

[19] Miao Cai, Chance C Coats, and Jian Huang. 2020. Hoop: efficient hardware-assisted out-of-place update for non-volatile memory. In *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. IEEE.

[20] Yu Cai, Yixin Luo, Erich F Haratsch, Ken Mai, and Onur Mutlu. 2015. Data retention in MLC NAND flash memory: Characterization, optimization, and recovery. In *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*. IEEE.

[21] Sanguhn Cha, O Seongil, Hyunsung Shin, Sangjoon Hwang, Kwangil Park, Seong Jin Jang, Joo Sun Choi, Gyo Young Jin, Young Hoon Son, Hyunyoon Cho, et al. 2017. Defect analysis and cost-effective resilience architecture for future DRAM devices. In *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE.

[22] Mu-Tien Chang, Paul Rosenfeld, Shih-Lien Lu, and Bruce Jacob. 2013. Technology comparison for large last-level caches (L 3 Cs): Low-leakage SRAM, low write-energy STT-RAM, and refresh-optimized eDRAM. In *2013 IEEE 19th international symposium on high performance computer architecture (HPCA)*.

[23] Athanasios Chatzidimitriou, George Papadimitriou, Christos Gavanas, George Katsoridas, and Dimitris Gizopoulos. 2019. Multi-bit upsets vulnerability analysis of modern microprocessors. In *2019 IEEE International Symposium on Workload Characterization (IISWC)*. IEEE.

[24] Odysseas Chatzopoulos, Nikos Karystinos, George Papadimitriou, Dimitris Gizopoulos, Harish D Dixit, and Sriram Sankar. 2025. Veritas–Demystifying silent data corruptions: μArch-level modeling and fleet data of modern x86 CPUs. In *2025 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE.

[25] Yangyin Chen. 2020. ReRAM: History, status, and future. *IEEE Transactions on Electron Devices* 67, 4 (2020).

[26] Zizhong Chen. 2013. Online-ABFT: An online algorithm based fault tolerance scheme for soft error detection in iterative methods. *ACM SIGPLAN Notices* 48, 8 (2013).

[27] Ping Chi, Shuangchen Li, Yuanqing Cheng, Yu Lu, Seung H Kang, and Yuan Xie. 2016. Architecture design with STT-RAM: Opportunities and challenges. In *2016 21st Asia and South Pacific design automation conference (ASP-DAC)*. IEEE.

[28] Jongouk Choi, Jaeseok Choi, Hyunwoo Joe, and Changhee Jung. 2024. Caphammer: Exploiting capacitor vulnerability of energy harvesting systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* (2024).

[29] Jongouk Choi, Hyunwoo Joe, and Changhee Jung. 2022. CapOS: Capacitor Error Resilience for Energy Harvesting Systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 41, 11 (2022).

[30] Jongouk Choi, Hyunwoo Joe, Yongjoo Kim, and Changhee Jung. 2019. Achieving stagnation-free intermittent computation with boundary-free adaptive execution. In *2019 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE.

[31] Jongouk Choi, Larry Kittinger, Qingrui Liu, and Changhee Jung. 2022. Compiler-directed high-performance intermittent computation with power failure immunity. In *2022 IEEE 28th Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE.

[32] Jongouk Choi, Qingrui Liu, and Changhee Jung. 2019. CoSpec: Compiler directed speculative intermittent computation. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*.

[33] Jongouk Choi, Jianping Zeng, Dongyoon Lee, Changwoo Min, and Changhee Jung. 2023. Write-light cache for energy harvesting systems. In *Proceedings of the 50th Annual International Symposium on Computer Architecture*.

[34] Christian Monzio Compagnoni, Akira Goda, Alessandro S Spinelli, Peter Feeley, Andrea L Lacaita, and Angelo Visconti. 2017. Reviewing the evolution of the NAND flash technology. *Proc. IEEE* 105, 9 (2017).

[35] Debendra Das Sharma, Robert Blankenship, and Daniel Berger. 2024. An introduction to the compute express link (cxl) interconnect. *Comput. Surveys* 56, 11 (2024).

[36] Teresa Davies and Zizhong Chen. 2013. Correcting soft errors online in LU factorization. In *Proceedings of the 22nd international symposium on High-performance parallel and distributed computing*.

[37] Timothy J Dell. 1997. A white paper on the benefits of chipkill-correct ECC for PC server main memory. *IBM Microelectronics division* 11, 1-23 (1997).

[38] Bang Di, Jiawen Liu, Hao Chen, and Dong Li. 2021. Fast, flexible, and comprehensive bug detection for persistent memory programs. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*.

[39] Chunfeng Du, Suzhen Wu, Jiapeng Wu, Bo Mao, and Shengzhe Wang. 2023. ESD: An ECC-assisted and Selective Deduplication for Encrypted Non-Volatile Main Memory. In *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE.

[40] Henry Duwe, Xun Jian, and Rakesh Kumar. 2015. Correction prediction: Reducing error correction latency for on-chip memories. In *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*. IEEE.

[41] Per Ekemark, Yuan Yao, Alberto Ros, Konstantinos Sagonas, and Stefanos Kaxiras. 2021. TSOPER: Efficient coherence-based strict persistency. In *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE.

[42] James Elliott, Kishor Kharbas, David Fiala, Frank Mueller, Kurt Ferreira, and Christian Engelmann. 2012. Combining partial redundancy and checkpointing for HPC. In *2012 IEEE 32nd International Conference on Distributed Computing Systems*. IEEE.

[43] Gan Fang, Jongouk Choi, and Changhee Jung. 2024. Hybrid Power Failure Recovery for Intermittent Computing. In *ACM/IEEE International Conference on Computer-Aided Design (ICCAD)*.

[44] Gan Fang and Changhee Jung. 2025. Rethinking dead block prediction for intermittent computing. In *2025 IEEE International Symposium on High Performance Computer Architecture (HPCA)*.

[45] Gan Fang, Jianping Zeng, Aditya Gupta, and Changhee Jung. 2025. Rethinking prefetching for intermittent computing. In *Proceedings of the 52nd Annual International Symposium on Computer Architecture*.

[46] Gan Fang, Jianping Zeng, Yuchen Zhou, and Changhee Jung. 2026. Intermittence-Aware Cache Compression. In *2026 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE.

[47] Hamed Farbeh, Leila Delshadtehrani, Hyeonggyu Kim, and Soontae Kim. 2020. ECC-United cache: Maximizing efficiency of error detection/correction codes in associative cache memories. *IEEE Trans. Comput.* 70, 4 (2020).

[48] Hamed Farbeh, Hyeonggyu Kim, Seyed Ghassem Miremadi, and Soontae Kim. 2016. Floating-ECC: Dynamic repositioning of error correcting code bits for extending the lifetime of STT-RAM caches. *IEEE Trans. Comput.* 65, 12 (2016).

[49] David Fiala, Frank Mueller, Christian Engelmann, Rolf Riesen, Kurt Ferreira, and Ron Brightwell. 2012. Detection and correction of silent data corruption for large-scale high-performance computing. In *SC'12: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. IEEE.

[50] Alexander Freij, Huiyang Zhou, and Yan Solihin. 2023. SecPB: Architectures for Secure Non-Volatile Memory with Battery-Backed Persist Buffers. In *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE.

[51] Robert Gallager. 2003. Low-density parity-check codes. *IRE Transactions on information theory* 8, 1 (2003).

[52] Alex Gendler, Arkady Bramnik, Ariel Szapiro, and Yiannakis Sazeides. 2018. Don't correct the tags in a cache, just check their hamming distance from the lookup tag. In *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE.

[53] Ellis Giles, Kshitij Doshi, and Peter Varman. 2013. Bridging the programming gap between persistent and volatile memory using WrAP. In *Proceedings of the ACM International Conference on Computing Frontiers*.

[54] Dimitris Gizopoulos. 2025. The Dark Side of Computing: Silent Data Corruptions. *Computer* 58, 6 (2025).

[55] Akira Goda. 2021. Recent progress on 3D NAND flash technologies. *Electronics* 10, 24 (2021).

[56] Seong-Lyong Gong, Jungrae Kim, Sangkug Lym, Michael Sullivan, Howard David, and Mattan Erez. 2018. Duo: Exposing on-chip redundancy to rank-level ecc for high reliability. In *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE.

[57] Seong-Lyong Gong, Minsoo Rhu, Jungrae Kim, Jinsuk Chung, and Mattan Erez. 2015. Clean-ecc: High reliability ecc for adaptive granularity memory system. In *Proceedings of the 48th International Symposium on Microarchitecture*.

[58] Laura M Grupp, John D Davis, and Steven Swanson. 2012. The bleak future of NAND flash memory.. In *FAST*, Vol. 7.

[59] Luanzheng Guo, Dong Li, Ignacio Laguna, and Martin Schulz. 2018. Fliptracker: Understanding natural error resilience in hpc applications. In *SC18: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE.

[60] Siddharth Gupta, Alexandros Daglis, and Babak Falsafi. 2019. Distributed logless atomic durability with persistent memory. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*.

[61] Ramyad Hadidi, Bahar Asgari, Burhan Ahmad Mudassar, Saibal Mukhopadhyay, Sudhakar Yalamanchili, and Hyesoon Kim. 2017. Demystifying the characteristics of 3D-stacked memories: A case study for hybrid memory cube. In *2017 IEEE international symposium on Workload characterization (IISWC)*. IEEE.

[62] Richard W Hamming. 1950. Error detecting and error correcting codes. *The Bell system technical journal* 29, 2 (1950).

[63] Youngkwang Han, Zhenyu Hu, Jongouk Choi, Kazi Abu Zubair, Amro Awad, Changhee Jung, and Brent Byunghoon Kang. 2024. A novel efficient crash consistency solution enabling rollback recovery for secure nvm in low-power energy harvesting systems. *IEEE Transactions on Dependable and Secure Computing* (2024).

[64] Noureldin Hassan, Byounguk Min, Changhee Jung, Yan Solihin, and Jongouk Choi. 2025. WarmCache: Exploiting STT-RAM Cache for Low-Power Intermittent Systems. In *Proceedings of the 52nd Annual International Symposium on Computer Architecture*.

[65] Zhengyang He, Yafan Huang, Hui Xu, Dingwen Tao, and Guanpeng Li. 2023. Demystifying and mitigating cross-layer deficiencies of soft error protection in instruction duplication. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*.

[66] John L Henning. 2006. SPEC CPU2006 benchmark descriptions. *ACM SIGARCH Computer Architecture News* 34, 4 (2006).

[67] George Hodgkins, Yi Xu, Steven Swanson, and Joseph Izraelevitz. 2023. Zhuque: Failure is Not an Option,it's an Exception. In *2023 USENIX Annual Technical Conference (USENIX ATC 23)*.

[68] Terry Ching-Hsiang Hsu, Helge Brügner, Indrajit Roy, Kimberly Keeton, and Patrick Eugster. 2017. NVthreads: Practical persistence for multi-threaded applications. In *Proceedings of the Twelfth European Conference on Computer Systems*.

[69] Yiming Huai et al. 2008. Spin-transfer torque MRAM (STT-MRAM): Challenges and prospects. *AAPPS bulletin* 18, 6 (2008).

[70] Shao-Yu Huang, Jianping Zeng, Xuanliang Deng, Sen Wang, Ashrarul Sifat, Burhanuddin Bharmal, Jia-Bin Huang, Ryan Williams, Haibo Zeng, and Changhee Jung. 2023. RTailor: Parameterizing Soft Error Resilience for Mixed-Criticality Real-Time Systems. In *2023 IEEE Real-Time Systems Symposium (RTSS)*. IEEE.

[71] Yafan Huang, Shengjian Guo, Sheng Di, Guanpeng Li, and Franck Cappello. 2022. Mitigating silent data corruptions in hpc applications across multiple program inputs. In *SC22: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE.

[72] Intel. [n. d.]. eADR: New Opportunities for Persistent Memory Applications. https://www.intel.com/content/www/us/en/developer/articles/technical/eadr-new-opportunities-for-persistent-memory-applications.html.

[73] Intel. 2020. Intel Optane DC Persistent Memory Quick Start Guide. https://www.intel.com/content/dam/support/us/en/documents/memory-and-storage/data-center-persistent-mem/Intel-Optane-DC-Persistent-Memory-Quick-Start-Guide.pdf.

[74] Intel. 2024. Intel 64 and IA-32 architectures software developer's manual. *Volume 3A: System Programming Guide, Part* 1 (2024).

[75] Joseph Izraelevitz, Terence Kelly, and Aasheesh Kolli. 2016. Failure-atomic persistent memory updates via JUSTDO logging. *ACM SIGARCH Computer Architecture News* 44, 2 (2016).

[76] Changhee Jung Jaeseok Choi, Hyunwoo Joe and Jongouk Choi. 2024. Defending Against EMI Attacks on Just-In-Time Checkpoint for Resilient Intermittent Systems. In *IEEE/ACM International Symposium on Microarchitecture (MICRO)*.

[77] Jungi Jeong and Changhee Jung. 2021. PMEM-spec: persistent memory speculation (strict persistency can trump relaxed persistency). In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*.

[78] Jungi Jeong, Chang Hyun Park, Jaehyuk Huh, and Seungryoul Maeng. 2018. Efficient hardware-assisted logging with asynchronous and direct-update for persistent memory. In *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE.

[79] Jungi Jeong, Jianping Zeng, and Changhee Jung. 2022. Capri: Compiler and architecture support for whole-system persistence. In *Proceedings of the 31st International Symposium on High-Performance Parallel and Distributed Computing*.

[80] Xun Jian, Henry Duwe, John Sartori, Vilas Sridharan, and Rakesh Kumar. 2013. Low-power, low-storage-overhead chipkill correct via multi-line error correction. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*.

[81] Arpit Joshi, Vijay Nagarajan, Stratis Viglas, and Marcelo Cintra. 2017. Atom: Atomic durability in non-volatile memory through hardware logging. In *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE.

[82] Rajat Kateja, Anirudh Badam, Sriram Govindan, Bikash Sharma, and Greg Ganger. 2017. Viyojit: Decoupling battery and DRAM capacities for battery-backed DRAM. *ACM SIGARCH Computer Architecture News* 45, 2 (2017).

[83] AV Khvalkovskiy, Dmytro Apalkov, Steve Watts, Roman Chepulskii, Robert S Beach, Adrian Ong, Xueti Tang, Alexander Driskill-Smith, William H Butler, Pieter B Visscher, et al. 2013. Basic principles of STT-MRAM cell operation in memory arrays. *Journal of Physics D: Applied Physics* 46, 7 (2013).

[84] Beomjun Kim and Myungsuk Kim. 2025. REO: Revisiting Erase Operation for Improving Lifetime and Performance of Modern NAND Flash-Based SSDs. *Electronics* 14, 4 (2025).

[85] Dongwhee Kim, Jaeyoon Lee, Wonyeong Jung, Michael Sullivan, and Jungrae Kim. 2023. Unity ECC: Unified memory protection against bit and chip errors. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*.

[86] Hongjune Kim, Jianping Zeng, Qingrui Liu, Mohammad Abdel-Majeed, Jaejin Lee, and Changhee Jung. 2020. Compiler-directed soft error resilience for lightweight GPU register file protection. In *Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation*.

[87] Jangwoo Kim, Nikos Hardavellas, Ken Mai, Babak Falsafi, and James Hoe. 2007. Multi-bit error tolerant caches using two-dimensional error coding. In *40th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO 2007)*. IEEE.

[88] Jungrae Kim, Michael Sullivan, and Mattan Erez. 2015. Bamboo ECC: Strong, safe, and flexible codes for reliable computer memory. In *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*. IEEE.

[89] Nam Sung Kim, Choungki Song, Woo Young Cho, Jian Huang, and Myoungsoo Jung. 2019. LL-PCM: Low-latency phase change memory architecture. In *Proceedings of the 56th Annual Design Automation Conference 2019*.

[90] Seongwoo Kim and Arun K Somani. 1999. Area efficient architectures for information integrity in cache memories. *ACM SIGARCH Computer Architecture News* 27, 2 (1999).

[91] Kingston Technology. 2020. *eMMC Life Cycle Estimating*. Technical Report. Kingston. https://connectivity-staging.s3.us-east-2.amazonaws.com/2024-03/eMMC-Life-Cycle-Estimating-by-Kingston.pdf Technical document stating 40 bits/1KB BCH and 120 bits/1KB LDPC ECC for NAND flash.

[92] Aasheesh Kolli, Jeff Rosen, Stephan Diestelhorst, Ali Saidi, Steven Pelley, Sihang Liu, Peter M Chen, and Thomas F Wenisch. 2016. Delegated persist ordering. In *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE.

[93] Kunal Korgaonkar, Ishwar Bhati, Huichu Liu, Jayesh Gaur, Sasikanth Manipatruni, Sreenivas Subramoney, Tanay Karnik, Steven Swanson, Ian Young, and

Hong Wang. 2018. Density tradeoffs of non-volatile memory as a replacement for SRAM based last level cache. In *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*. IEEE.

[94] Kyle Kuan and Tosiron Adegbija. 2019. HALLS: An energy-efficient highly adaptable last level STT-RAM cache for multicore systems. *IEEE Trans. Comput.* 68, 11 (2019).

[95] Lenovo Press 2024. *ThinkSystem PM9A3 Read Intensive NVMe PCIe 4.0 SSDs*. Lenovo Press. https://lenovopress.lenovo.com/lp1557-thinksystem-pm9a3-read-intensive-nvme-pcie-40-ssds Product guide. Accessed: 2026-03-19.

[96] Guanpeng Li, Siva Kumar Sastry Hari, Michael Sullivan, Timothy Tsai, Karthik Pattabiraman, Joel Emer, and Stephen W Keckler. 2017. Understanding error propagation in deep learning neural network (DNN) accelerators and applications. In *Proceedings of the international conference for high performance computing, networking, storage and analysis*.

[97] Sheng Li, Jung Ho Ahn, Richard D Strong, Jay B Brockman, Dean M Tullsen, and Norman P Jouppi. 2009. McPAT: An integrated power, area, and timing modeling framework for multicore and manycore architectures. In *Proceedings of the 42nd annual ieee/acm international symposium on microarchitecture*.

[98] Shuwen Liang, Zhi Qiao, Sihai Tang, Jacob Hochstetler, Song Fu, Weisong Shi, and Hsing-Bung Chen. 2019. An empirical study of quad-level cell (QLC) NAND flash SSDs for big data applications. In *2019 IEEE International Conference on Big Data (Big Data)*. IEEE.

[99] Xin Liang, Jieyang Chen, Dingwen Tao, Sihuan Li, Panruo Wu, Hongbo Li, Kaiming Ouyang, Yuanlai Liu, Fengguang Song, and Zizhong Chen. 2017. Correcting soft errors online in fast fourier transform. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*.

[100] Qingrui Liu, Joseph Izraelevitz, Se Kwon Lee, Michael L Scott, Sam H Noh, and Changhee Jung. 2018. iDO: Compiler-directed failure atomicity for nonvolatile memory. In *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE.

[101] Qingrui Liu and Changhee Jung. 2016. Lightweight hardware support for transparent consistency-aware checkpointing in intermittent energy-harvesting systems. In *2016 5th Non-Volatile Memory Systems and Applications Symposium (NVMSA)*. IEEE.

[102] Qingrui Liu, Changhee Jung, Dongyoon Lee, and Devesh Tiwari. 2015. Clover: Compiler directed lightweight soft error resilience. *ACM Sigplan Notices* 50, 5 (2015).

[103] Qingrui Liu, Changhee Jung, Dongyoon Lee, and Devesh Tiwari. 2016. Compiler-directed lightweight checkpointing for fine-grained guaranteed soft error recovery. In *SC'16: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE.

[104] Qingrui Liu, Changhee Jung, Dongyoon Lee, and Devesh Tiwari. 2016. Compiler-directed soft error detection and recovery to avoid DUE and SDC via Tail-DMR. *ACM Transactions on Embedded Computing Systems (TECS)* 16, 2 (2016).

[105] Qingrui Liu, Changhee Jung, Dongyoon Lee, and Devesh Tiwarit. 2016. Low-cost soft error resilience with unified data verification and fine-grained recovery for acoustic sensor based detection. In *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE.

[106] Sihang Liu, Suyash Mahar, Baishakhi Ray, and Samira Khan. 2021. PMFuzz: Test case generation for persistent memory programs. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*.

[107] Sihang Liu, Korakit Seemakhupt, Yizhou Wei, Thomas Wenisch, Aasheesh Kolli, and Samira Khan. 2020. Cross-failure bug detection in persistent memory programs. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*.

[108] Sihang Liu, Yizhou Wei, Jishen Zhao, Aasheesh Kolli, and Samira Khan. 2019. PMTest: A fast and flexible testing framework for persistent memory programs. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*.

[109] Júnior Löff, Dalvan Griebler, Gabriele Mencagli, Gabriell Araujo, Massimo Torquati, Marco Danelutto, and Luiz Gustavo Fernandes. 2021. The NAS parallel benchmarks for evaluating C++ parallel programming frameworks on shared-memory architectures. *Future Generation Computer Systems* 125 (2021).

[110] Gabriel H Loh, Yuan Xie, and Bryan Black. 2007. Processor design in 3D die-stacking technologies. *Ieee Micro* 27, 3 (2007).

[111] Thiago Macieira, Sankar Gurumurthy, Sudhanva Gurumurthi, Amr Haggag, George Papadimitriou, and Dimitris Gizopoulos. 2024. Silent data corruptions in computing: Understand and quantify. In *2024 IEEE 30th International Symposium on On-Line Testing and Robust System Design (IOLTS)*. IEEE.

[112] Jose Maiz, Scott Hareland, Kevin Zhang, and Patrick Armstrong. 2003. Characterization of multi-bit soft error events in advanced SRAMs. In *IEEE International Electron Devices Meeting 2003*. IEEE.

[113] Virendra J Marathe, Margo I Seltzer, Steve Byan, and Tim Harris. 2017. Persistent Memcached: Bringing Legacy Code to Byte-Addressable Persistent Memory.. In *HotStorage*.

[114] Amirsaman Memaripour and Steven Swanson. 2018. Breeze: User-level access to non-volatile main memories for legacy software. In *2018 IEEE 36th International Conference on Computer Design (ICCD)*. IEEE.

[115] Rino Micheloni, Luca Crippa, and Alessia Marelli. 2010. *Inside NAND flash memories*. Springer Science & Business Media.

[116] Neal R Mielke, Robert E Frickey, Ivan Kalastirsky, Minyan Quan, Dmitry Ustinov, and Venkatesh J Vasudevan. 2017. Reliability of solid-state drives based on NAND flash memory. *Proc. IEEE* 105, 9 (2017).

[117] Chi Cao Minh, JaeWoong Chung, Christos Kozyrakis, and Kunle Olukotun. 2008. STAMP: Stanford transactional applications for multi-processing. In *2008 IEEE International Symposium on Workload Characterization*. IEEE.

[118] Sparsh Mittal, Jeffrey S Vetter, and Dong Li. 2014. A survey of architectural approaches for managing embedded DRAM and non-volatile on-chip caches. *IEEE Transactions on Parallel and Distributed Systems* 26, 6 (2014).

[119] Sparsh Mittal, Rujia Wang, and Jeffrey Vetter. 2017. DESTINY: A comprehensive tool with 3D and multi-level cell memory modeling capability. *Journal of Low Power Electronics and Applications* 7, 3 (2017).

[120] Kyoji Mizoguchi, Tomonori Takahashi, Seiichi Aritome, and Ken Takeuchi. 2017. Data-retention characteristics comparison of 2D and 3D TLC NAND flash memories. In *2017 IEEE International Memory Workshop (IMW)*. IEEE.

[121] Prashant J Nair, Vilas Sridharan, and Moinuddin K Qureshi. 2016. XED: Exposing on-die error detection information for strong memory reliability. *ACM SIGARCH Computer Architecture News* 44, 3 (2016).

[122] Sanketh Nalli, Swapnil Haria, Mark D Hill, Michael M Swift, Haris Volos, and Kimberly Keeton. 2017. An analysis of persistent memory use with WHISPER. *ACM SIGPLAN Notices* 52, 4 (2017).

[123] Dushyanth Narayanan and Orion Hodson. 2012. Whole-system persistence. In *Proceedings of the seventeenth international conference on Architectural Support for Programming Languages and Operating Systems*.

[124] Ian Neal, Andrew Quinn, and Baris Kasikci. 2021. Hippocrates: Healing persistent memory bugs without doing any harm. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*.

[125] Yuanjiang Ni, Jishen Zhao, Heiner Litz, Daniel Bittman, and Ethan L Miller. 2019. SSP: Eliminating redundant writes in failure-atomic NVRAMs via shadow subpaging. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*.

[126] Matheus Almeida Ogleari, Ethan L Miller, and Jishen Zhao. 2018. Steal but no force: Efficient hardware undo+ redo logging for persistent memory systems. In *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE.

[127] Byoungchan Oh, Nilmini Abeyratne, Nam Sung Kim, Jeongseob Ahn, Ronald G Dreslinski, and Trevor Mudge. 2022. Rethinking DRAM's page mode with STT-MRAM. *IEEE Trans. Comput.* 72, 5 (2022).

[128] Sompong P Olarig. 2006. Technique for implementing chipkill in a memory system. US Patent 7,096,407.

[129] Hamza Omar and Omer Khan. 2021. Prism: Strong hardware isolation-based soft-error resilient multicore architecture with high performance and availability at low hardware overheads. *ACM Transactions on Architecture and Code Optimization (TACO)* 18, 3 (2021).

[130] Hamza Omar, Qingchuan Shi, Masab Ahmad, Halit Dogan, and Omer Khan. 2018. Declarative resilience: A holistic soft-error resilient multicore architecture that trades off program accuracy for efficiency. *ACM Transactions on Embedded Computing Systems (TECS)* 17, 4 (2018).

[131] Shweta Pandey, Aditya K Kamath, and Arkaprava Basu. 2022. GPM: leveraging persistent memory from a GPU. In *Proceedings of ACM International Conference on Architectural Support for Programming Languages and Operating Systems*.

[132] Shweta Pandey, Aditya K Kamath, and Arkaprava Basu. 2023. Scoped Buffered Persistency Model for GPUs. In *Proceedings of ACM International Conference on Architectural Support for Programming Languages and Operating Systems*.

[133] George Papadimitriou and Dimitris Gizopoulos. 2023. Silent data corruptions: Microarchitectural perspectives. *IEEE Trans. Comput.* 72, 11 (2023).

[134] Bhukya Krishna Priya. 2023. Enhancing the lifetime of STT-RAM using compression based wear leveling technique. *Microelectronics Reliability* 143 (2023).

[135] Moinuddin Qureshi. 2021. Rethinking ECC in the era of row-hammer. In *DRAMSec*.

[136] Moinuddin K Qureshi, John Karidis, Michele Franceschini, Vijayalakshmi Srinivasan, Luis Lastras, and Bulent Abali. 2009. Enhancing lifetime and security of PCM-based main memory with start-gap wear leveling. In *Proceedings of the 42nd annual IEEE/ACM international symposium on microarchitecture*.

[137] Azalea Raad, Luc Maranget, and Viktor Vafeiadis. 2022. Extending Intel-X86 consistency and persistency: Formalising the semantics of Intel-X86 memory types and non-temporal stores. *Proceedings of the ACM on Programming Languages* 6, POPL (2022).

[138] Md Hasanur Rahman, Aabid Shamji, Shengjian Guo, and Guanpeng Li. 2021. Peppa-x: finding program test inputs to bound silent data corruption vulnerability in hpc applications. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*.

[139] Jinglei Ren, Jishen Zhao, Samira Khan, Jongmoo Choi, Yongwei Wu, and Onur Mutlu. 2015. ThyNVM: Enabling software-transparent crash consistency in

persistent memory systems. In *Proceedings of the 48th International Symposium on Microarchitecture*.

[140] Christos Sakalis, Carl Leonardsson, Stefanos Kaxiras, and Alberto Ros. 2016. Splash-3: A properly synchronized benchmark suite for contemporary research. In *2016 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE.

[141] Soheil Salehi, Deliang Fan, and Ronald F Demara. 2017. Survey of STT-MRAM cell design strategies: Taxonomy and sense amplifier tradeoffs for resiliency. *ACM Journal on Emerging Technologies in Computing Systems (JETC)* 13, 3 (2017).

[142] Samsung Electronics Co., Ltd. 2022. Samsung V-NAND SSD PM9A3 2022 Data Sheet. *Samsung Semiconductor* (Aug. 2022). https://image.semiconductor.samsung.com/resources/data-sheet/Samsung_SSD_PM9A3_Data_Sheet_Rev1.0.pdf Revision 1.0.

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

[144] Dipanjan Sengupta, Qi Wang, Haris Volos, Ludmila Cherkasova, Jun Li, Guilherme Magalhaes, and Karsten Schwan. 2015. A framework for emulating non-volatile memory systemswith different performance characteristics. In *Proceedings of the 6th ACM/SPEC International Conference on Performance Engineering*.

[145] Sara Mahdizadeh Shahri, Seyed Armin Vakil Ghahani, and Aasheesh Kolli. 2020. (almost) Fence-less persist ordering. In *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE.

[146] Debendra Das Sharma. 2022. Compute express link (cxl): Enabling heterogeneous data-centric computing with heterogeneous memory hierarchy. *IEEE Micro* 43, 2 (2022).

[147] Seunghee Shin, Satish Kumar Tirukkovalluri, James Tuck, and Yan Solihin. 2017. Proteus: A flexible and fast software supported hardware logging approach for nvm. In *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*.

[148] Premkishore Shivakumar and Norman P Jouppi. 2001. Cacti 3.0: An integrated cache timing, power, and area model. (2001).

[149] Vilas Sridharan, Nathan DeBardeleben, Sean Blanchard, Kurt B Ferreira, Jon Stearley, John Shalf, and Sudhanva Gurumurthi. 2015. Memory errors in modern systems: The good, the bad, and the ugly. *ACM SIGARCH Computer Architecture News* 43, 1 (2015), 297–310.

[150] Nikolaos Strikos, Vasileios Kontorinis, Xiangyu Dong, Houman Homayoun, and Dean Tullsen. 2013. Low-current probabilistic writes for power-efficient STT-RAM caches. In *2013 IEEE 31st International Conference on Computer Design (ICCD)*. IEEE.

[151] Guangyu Sun, Eren Kursun, Jude A Rivers, and Yuan Xie. 2013. Exploring the vulnerability of CMPs to soft errors with 3D stacked nonvolatile memory. *ACM Journal on Emerging Technologies in Computing Systems (JETC)* 9, 3 (2013).

[152] Zhenyu Sun, Xiuyuan Bi, Hai Li, Weng-Fai Wong, Zhong-Liang Ong, Xiaochun Zhu, and Wenqing Wu. 2011. Multi retention level STT-RAM cache designs with a dynamic refresh scheme. In *proceedings of the 44th annual IEEE/ACM international symposium on microarchitecture*.

[153] Swissbit AG. 2024. *Error Correction Coding*. Technical Report. Swissbit. https://www.astutegroup.com/wp-content/uploads/2024/07/WP_Swissbit_Error_Correction_Coding_EN.pdf White paper on ECC for NAND flash; discusses BCH codes correcting 20–120 bit errors per 2 KB block.

[154] Vineet Veer Tyagi and DPCM Buddhi. 2007. PCM thermal storage in buildings: A state of art. *Renewable and sustainable energy reviews* 11, 6 (2007).

[155] Viking Technology. 2017. *SATA 6Gb/s 2.5" SSD Manual*. Technical Report. Viking Technology. https://www.vikingtechnology.com/wp-content/uploads/2021/03/2.5inch_PhisonS10.pdf Product manual specifying "ECC up to 120bit/2KB ECC circuit (BCH)".

[156] Haris Volos, Andres Jaan Tack, and Michael M Swift. 2011. Mnemosyne: Lightweight persistent memory. *ACM SIGARCH Computer Architecture News* 39, 1 (2011).

[157] Wujie Wen, Mengjie Mao, Xiaochun Zhu, Seung H Kang, Danghui Wang, and Yiran Chen. 2013. CD-ECC: Content-dependent error correction codes for combating asymmetric nonvolatile memory operation errors. In *2013 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE.

[158] Panruo Wu and Zizhong Chen. 2014. FT-ScaLAPACK: Correcting soft errors online for ScaLAPACK Cholesky, QR, and LU factorization routines. In *Proceedings of the 23rd international symposium on High-performance parallel and distributed computing*.

[159] Yilun Wu, Byounguk Min, Mohannad Ismail, Wenjie Xiong, Changhee Jung, and Dongyoon Lee. 2024. {IntOS}: Persistent Embedded Operating System and Language Support for Multi-threaded Intermittent Computing. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*.

[160] Yi Xu, Joseph Izraelevitz, and Steven Swanson. 2021. Clobber-NVM: log less, re-execute more. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*.

[161] Sujay Yadalam, Nisarg Shah, Xiangyao Yu, and Michael Swift. 2022. ASAP: A speculative approach to persistence. In *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE.

[162] Jianlei Yang, Peiyuan Wang, Yaojun Zhang, Yuanqing Cheng, Weisheng Zhao, Yiran Chen, and Hai Helen Li. 2015. Radiation-induced soft error analysis of STT-MRAM: A device to circuit approach. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 35, 3 (2015).

[163] Chencheng Ye, Yuanchao Xu, Xipeng Shen, Hai Jin, Xiaofei Liao, and Yan Solihin. 2022. Preserving addressability upon GC-triggered data movements on non-volatile memory. *ACM Transactions on Architecture and Code Optimization (TACO)* 19, 2 (2022).

[164] Chencheng Ye, Yuanchao Xu, Xipeng Shen, Yan Sha, Xiaofei Liao, Hai Jin, and Yan Solihin. 2023. SpecPMT: Speculative Logging for Resolving Crash Consistency Overhead of Persistent Memory. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*.

[165] Ravikiran Yeleswarapu and Arun K Somani. 2021. Addressing multiple bit/symbol errors in DRAM subsystem. *PeerJ Computer Science* 7 (2021).

[166] Doe Hyun Yoon and Mattan Erez. 2009. Memory mapped ECC: Low-cost error protection for last level caches. In *Proceedings of the 36th annual international symposium on Computer architecture*.

[167] Jianping Zeng. 2024. Compiler and Architecture Co-design for Reliable Computing. (7 2024). https://doi.org/10.25394/PGS.26356696.v1

[168] Jianping Zeng, Jongouk Choi, Xinwei Fu, Ajay Paddayuru Shreepathi, Dongyoon Lee, Changwoo Min, and Changhee Jung. 2021. Replaycache: Enabling volatile cachesfor energy harvesting systems. In *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*.

[169] Jianping Zeng, Shao-Yu Huang, Jiuyang Liu, and Changhee Jung. 2024. Soft Error Resilience at Near-Zero Cost. In *Proceedings of the 38th ACM International Conference on Supercomputing*.

[170] Jianping Zeng, Jungi Jeong, and Changhee Jung. 2023. Persistent processor architecture. In *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture*.

[171] Jianping Zeng, Hongjune Kim, Jaejin Lee, and Changhee Jung. 2021. Turnpike: Lightweight Soft Error Resilience for In-Order Cores. In *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*.

[172] Jianping Zeng, Shuyi Pei, Da Zhang, Yuchen Zhou, Amir Beygi, Xuebin Yao, Ramdas Kachare, Tong Zhang, Zongwang Li, Marie Nguyen, et al. 2025. Performance Characterizations and Usage Guidelines of Samsung CMM-H. *IEEE Micro* (2025).

[173] Jianping Zeng, Shuyi Pei, Da Zhang, Yuchen Zhou, Amir Beygi, Xuebin Yao, Ramdas Kachare, Tong Zhang, Zongwang Li, Marie Nguyen, et al. 2025. Performance Characterizations and Usage Guidelines of Samsung CXL Memory Module Hybrid Prototype. *arXiv preprint arXiv:2503.22017* (2025).

[174] Jianping Zeng, Tong Zhang, and Changhee Jung. 2024. Compiler-Directed Whole-System Persistence. In *Proceedings of the 51th Annual International Symposium on Computer Architecture*.

[175] Chao Zhang, Khaled Abdelaal, Angel Chen, Xinhui Zhao, Wujie Wen, and Xiaochen Guo. 2020. ECC cache: A lightweight error detection for phase-change memory stuck-at faults. In *Proceedings of the 39th International Conference on Computer-Aided Design*.

[176] Da Zhang, Vilas Sridharan, and Xun Jian. 2018. Exploring and optimizing chipkill-correct for persistent memory based on high-density nvrams. In *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE.

[177] Xiangqun Zhang, Shuyi Pei, Jongmoo Choi, and Bryan S Kim. 2023. Excessive SSD-internal parallelism considered harmful. In *Proceedings of the 15th ACM Workshop on Hot Topics in Storage and File Systems*.

[178] Yida Zhang and Changhee Jung. 2022. Featherweight soft error resilience for GPUs. In *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE.

[179] Jishen Zhao, Sheng Li, Doe Hyun Yoon, Yuan Xie, and Norman P Jouppi. 2013. Kiln: Closing the performance gap between systems with and without persistence support. In *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture*.

[180] Kai Zhao, Wenzhe Zhao, Hongbin Sun, Xiaodong Zhang, Nanning Zheng, and Tong Zhang. 2013. {LDPC-in-SSD}: Making advanced error correction codes work effectively in solid state drives. In *11th USENIX Conference on File and Storage Technologies (FAST 13)*.

[181] Yuchen Zhou, Jianping Zeng, Jungi Jeong, Jongouk Choi, and Changhee Jung. 2023. SweepCache: Intermittence-Aware Cache on the Cheap. In *MICRO-56: 56th Annual IEEE/ACM International Symposium on Microarchitecture*.

[182] Yuchen Zhou, Jianping Zeng, and Changhee Jung. 2024. Lightwsp: Whole-system persistence on the cheap. In *2024 57th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE.

[183] Kazi Abu Zubair, Sudhanva Gurumurthi, Vilas Sridharan, and Amro Awad. 2021. Soteria: Towards resilient integrity-protected and encrypted non-volatile memories. In *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*.