Compression-Aware and Performance-Efficient Insertion Policies for Long-Lasting Hybrid LLCs

Carlos Escuin*, Asif Ali Khan†, Pablo Ibáñez*, Teresa Monreal‡, Jeronimo Castrillon† and Víctor Viñals*
*Dept. Informática e Ingeniería de Sistemas - I3A, Universidad de Zaragoza, Zaragoza, Spain
†Chair for Compiler Construction, TU Dresden, Dresden, Germany
‡Universitat Politècnica de Catalunya · BarcelonaTech, Barcelona, Spain

***Abstract*—Emerging non-volatile memory (NVM) technologies can potentially replace large SRAM memories such as the last-level cache (LLC). However, despite recent advances, NVMs suffer from higher write latency and limited write endurance. Recently, NVM-SRAM hybrid LLCs are proposed to combine the best of both worlds. Several policies have been proposed to improve the performance and lifetime of hybrid LLCs by intelligently steering the incoming LLC blocks into either the SRAM or NVM part, regarding the cache behavior of the LLC blocks and the SRAM/NVM device properties. However, these policies neither consider compressing the contents of the cache block nor using partially worn-out NVM cache blocks.**

This paper proposes new insertion policies for byte-level fault-tolerant hybrid LLCs that collaboratively optimize for lifetime and performance. Specifically, we leverage data compression to utilize partially defective NVM cache entries, thereby improving the LLC hit rate. The key to our approach is to guide the insertion policy by both the reuse properties of the block and the size resulting from its compression. A block is inserted in NVM only if it is a read-reuse block or its compressed size is lower than a threshold. It will be inserted in SRAM if the block is a write-reuse or its compressed size is greater than the threshold. We use set-dueling to tune the compression threshold at runtime. This compression threshold provides a knob to control the NVM write rate and, together with a rule-based mechanism, allows balancing performance and lifetime.

Overall, our evaluation shows that, with affordable hardware overheads, the proposed schemes can nearly reach the performance of an SRAM cache with the same associativity while improving lifetime by 17× compared to a hybrid NVM-unaware LLC. Our proposed scheme outperforms the state-of-the-art insertion policies by 9% while achieving a comparative lifetime. The rule-based mechanism shows that by compromising, for instance, 1.1% and 1.9% performance, the NVM lifetime can be further increased by 28% and 44%, respectively.

I. Introduction

The ever-growing working set sizes of emerging application domains such as machine learning and artificial intelligence require larger on-chip *last-level caches* (LLCs). Increasing the LLC capacity is also imperative as the number of cores sharing it grows, because it is the last line of defense of the processor against costly off-chip memory accesses. However, with the deceleration of Moore's law, the increase in the LLC capacity has stagnated [22]. The scaling of conventional SRAM-based LLCs significantly increases the leakage power consumption and is becoming prohibitive in terms of both capacity and area [29]. Therefore, recent research advocates employing emerging *non-volatile memory* (NVM) technologies to increase the LLC capacity.

Emerging NVM technologies such as spin-transfer and spin-orbit torque (STT and SOT) magnetic RAM (MRAM), phase change memory, resistive memory, and racetrack memory have shown great promise to replace or augment conventional SRAM and DRAM technologies. In the last decade, some NVM technologies have matured greatly and have made their way into the memory hierarchy [23], [39]. Compared to conventional SRAM technologies, NVMs, particularly MRAMs are attractive alternatives for large size LLCs because they are extremely energy efficient, offer larger densities, and SRAM-competitive read latencies [4], [9], [27], [29]. However, without proper buffering the slow write operation on NVMs can degrade performance by throttling subsequent critical reads, potentially leading to core stalls. In addition to the read/write asymmetry, most NVMs also have a limited endurance, i.e., the number of writes that each bitcell supports, until it deteriorates and loses its retention capacity is limited and can be approximated by a normal distribution with a mean that can vary between 10^6 and 10^{12} [10], [17], [42]–[44]. Many device, circuit, and architectural optimizations have been proposed to mitigate the impact of the write operations on the NVM-LLC performance and lifetime [20], [29], [33]. However, these solutions increase the overall power consumption and reduce the NVM capacity, thereby offsetting the NVMs benefits.

Recent proposals combine the best of both worlds, i.e., performance and endurance of SRAM/DRAM with the energy efficiency and density of STT-MRAM to implement *hybrid* LLCs [4], [19], [27], [32]. MRAMs, compared to other NVM technologies, offer better endurance and SRAM comparable read latencies with higher density. However, it still suffers from higher write latencies and limited endurance compared to conventional SRAM/DRAM technologies. Therefore, the performance, energy, and lifetime improvements of these hybrid proposals are associated with the reduction in number of write requests to the NVM[1]. Thus, various techniques have been proposed to identify and steer write-intensive blocks towards the SRAM part and read-intensive blocks towards the NVM part [9], [32]. The identification of read- and write-intensive blocks is either performed with address-based predictors that sample LLC accesses [4] or with predictors using counters and threshold values for LLC block accesses [28].

The asymmetric read-write operations in NVMs have also

[1] Unless otherwise mentioned, NVM hereafter refers to STT-MRAM.

motivated novel insertion policies. For instance, Luo et al. propose TAP which classifies LLC write requests into demand-writes, prefetch-writes, and clean/dirty thrashing-writes [32]. Thrashing requests are routed to the SRAM part to reduce the LLC energy consumption and improve lifetime. Compared to the *least recently used* (LRU) replacement policy, their proposal reduces the energy consumption by 25%. Similarly, Cheng et al. propose LHybrid [9], a loop-block aware policy to insert only clean blocks that are frequently reused (loop-blocks) in the NVM part, protecting them from non-loop-blocks when a victim is selected for replacement. LHybrid significantly reduces write traffic and improves LLC lifetime. However, in these previous proposals, the LLC lifetime improvement is only achieved by conservative insertion in the NVM part, which limits LLC performance.

In addition to specific insertion and replacement policies, a different class of optimization techniques improves the NVM lifetime and performance by decreasing the average number of bits written in each write request [11], [34], [35]. In particular, compression can increase effective main memory capacity and reduce bandwidth utilization by $2-4\times$ [1]. Unfortunately, compression has received only little attention in the context of hybrid LLCs. In particular, the behavior of the state-of-the-art insertion policies for hybrid-LLCs enhanced with compression are yet to be investigated.

Figure 1 shows a forecast of the performance evolution of a hybrid LLC over time, until the capacity of the NVM drops to 50%. Twelve and four ways have been devoted to NVM and SRAM storage, respectively. Further details on the methodology and workload are discussed in Section V-A. The baseline hybrid LLC configuration (BH) manages a single LRU list for all ways in a cache set. The insertion policy does not distinguish between NVM and SRAM parts and incoming blocks are written to the LRU way, regardless of its technology. BH initial performance is excellent, but the write wear on the NVM part leads to 50% of its capacity being exhausted in less than three months.

Compared to BH, LHybrid [9], thanks to its selective insertion policy, improves LLC lifetime by more than 19×, but at the cost of significant performance degradation ($> 11\%$). TAP [32] sacrifices even more performance in exchange for a lifetime improvement of 39×.

This paper bridges the performance and lifetime disparities between BH and LHybrid approaches by proposing CP_SD, a hybrid insertion policy that combines data compression and block reuse information. Besides, the NVM part tolerates byte-level faults and is provided with a block rearrangement circuitry. As can be seen in Figure 1, CP_SD maintains for almost two years 97% BH performance, reaching 50% capacity exhaustion in about three years and nine months. Our solution strikes a good balance in the performance vs. lifetime trade-off, prioritizing performance without neglecting lifetime. Moreover, this paper also proposes a rule-based mechanism to further tune this trade-off: CP_SD_Th4 and CP_SD_Th8 in Figure 1 trade 1.1% and 1.9% performance in exchange for 28% and 44% NVM lifetime improvement, respectively.

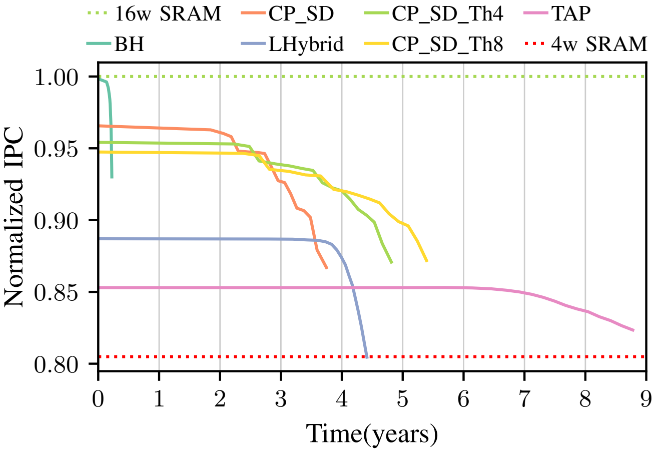


Fig. 1: Performance vs. time for various hybrid LLCs until the NVM part loses 50% capacity. The write endurance of NVM bitcells follows a normal distribution of $\mu = 10^{10}$ and $cv = 0.2$. Bounds of SRAM-only LLCs are also plotted.

Specifically, this work makes the following contributions to shared hybrid NVM-SRAM LLCs whose NVM part tolerates byte-level faults and leverages data compression:

- A novel insertion scheme that places cache blocks into either the SRAM or NVM part of the LLC, considering the read-reuse, write-reuse and compression features of cache blocks. In the NVM part, the replacement algorithm considers an NVM fault-map and the compressed size of the incoming LLC block, replacing the LRU block from the frames the incoming block can fit in.
- A threshold-based mechanism tunes the write-traffic to the NVM part, thereby allowing to explore the trade-off between performance and lifetime. We propose to use Set Dueling [37] to capture the runtime behaviour of the workload and allow more (or less) compressed blocks to be inserted in the NVM part. The sample cache sets collect the number of writes and the number of hits. Based on these counters, a rule-based decision mechanism balances lifetime and performance.
- The forecasting procedure introduced in [15] is adapted to the hybrid LLC scenario. This procedure tracks NVM aging, providing the temporal evolution of performance and capacity. It allows to analyze all dimensions of the hybrid LLC design.
- For a fair comparison, it is necessary to test existing insertion policies on NVM caches that lose capacity due to aging. Therefore, the state-of-the-art LHybrid and TAP policies [9], [32] are implemented in a fault-aware environment, extended with *frame-disabling* to tolerate hard-errors [7], [46].
- For evaluation, we consider multi-programmed workloads of memory-intensive applications from the SPEC 2006 and SPEC 2017 suites. The evaluation environment combines three elements: a fast architectural simulator [16], a detailed cycle-level simulator [31] and the forecast procedure mentioned above [15]. We present a comprehensive

analysis and evaluation of our novel schemes and their comparison to the state-of-the-art. We show that our proposals consistently and significantly outperform the state-of-the-art in all performance metrics.

The rest of the paper is organized as follows. Section II discusses the background and motivation of this study. Section III describes the microarchitecture of the hybrid LLC. Section IV describes the proposed insertion policies together with the Set Dueling mechanism. Section V presents the results of our insertion policies against the state-of-the-art. Finally, Section VI concludes the paper.

II. Background and motivation

This section introduces the NVM endurance problem, provides background on data compression, explains state-of-the-art insertion policies, and makes a quantitative analysis of these insertion policies.

A. NVM endurance

Most NVM technologies have limited write endurance, i.e., NVM bitcells can only be written a maximum of 10^n times where n varies significantly depending on the technology, optimizations, and target market from 10^6 - 10^{12} [10], [17], [42]–[44]. The endurance is usually approximated by a normal distribution of mean $\mu = 10^n$ and coefficient of variation $cv = \frac{\sigma}{\mu}$, usually between 0.2 and 0.3. The cv reflects the variability in the manufacturing process [12], [17].

In the context of NVM-based LLCs, lifetime is strictly determined by the LLC write traffic and the write distribution across cache lines. Streaming or thrashing workloads, for instance, can reduce the NVM lifetime from a few years to a few months. NVM structures should be provided with wear-leveling mechanisms able to evenly distribute the write wear throughout all the cache dimensions: sets, frames within sets, and bytes within frames; we call frame to the physical arrays of bitcells storing a cache block. Wear-leveling mechanisms have been thoroughly studied in the literature to prevent early wear-out of memory regions [2], [17], [25], [42]. Our proposal is independent of the wear-leveling mechanism used. In our case, an intra-frame wear-leveling is used by means of a global counter, like in [24]; but any other mechanism could be used.

B. Data compression

The lifetime of an NVM can increase if the information it contains has been previously compressed, because the average number of bitcells written decreases. In the case of NVM caches, if a byte-level fault-aware mechanism is also available, see Section III-B, compression will allow frames with faulty bytes to hold compressed blocks [15], [18].

Since compression directly influences access latency and write bandwidth, the compression mechanism adopted on the cache hierarchy must satisfy some properties. First, the decompression latency must be as low as possible because it is on the critical path of block service. Second, the compression ratio should be as high as possible as it determines the amount of bytes written to the NVM. Besides, the compression mechanism must have a wide coverage and low hardware complexity to maximize utilization of partially disabled cache frames with with little overhead.

Our proposed policies are orthogonal to the compression mechanism and can be used with any compression scheme that satisfies the above properties. For this work we use a slightly modified version of the *Base-Delta Immediate (BDI)* compression mechanism that provides fast decompression latency (1 cycle), wide coverage, little hardware overhead, and an acceptable compression ratio [36]. The algorithm is based on value locality; it assumes that a 64-byte block can be split into either thirty-two 2-byte, sixteen 4-byte, or eight 8-byte values and can be compacted using a *Base* value and a series of differences with respect to the Base (*Deltas*). A particular combination of Base and Delta is called *Compression Encoding* (CE), see Table I. They are known a priori and are computed in parallel.

TABLE I: BDI compression encodings and their sizes in bytes.

CE	**Base**	**Delta**	**Size**	CE	**Base**	**Delta**	**Size**
Zeros	0	0	0	B2Δ1	2	1	37
RV(8)	8	0	8	B8Δ4	8	4	37
B8Δ1	8	1	16	B8Δ5*	8	5	44
B4Δ1	4	1	21	B4Δ3*	4	3	51
B8Δ2	8	2	23	B8Δ6*	8	6	51
B8Δ3	8	3	30	B8Δ7*	8	7	58
B4Δ2	4	2	36	Unc.	-	-	64

The compression encodings that do not achieve a significant compression ratio are typically discarded in the original BDI proposal in order to increase the average compression ratio of the mechanism. These blocks correspond in our design to the ones whose compressed size is greater than 37, we refer to them as *low-compression ratio* (LCR) blocks, marked with a star in Table I. Similarly, we refer as *high-compression ratio* (HCR) blocks to the ones having compressed sizes lower than or equal to 37 bytes. For our design, LCR blocks are equally important as HCR ones since they allow frames with only a few faulty bytes to allocate blocks that cannot be compressed as much as HCR ones.

Figure 2 shows the compression ratios of the most memory demanding applications of SPEC 2006 and SPEC 2017 benchmark suites. On average, 78% of the total cache blocks are compressible, either HCR blocks (49%) or LCR blocks (29%). In an NVM cache, since not all bitcells wear out simultaneously, frames gradually become partially defective, so the remaining functional bitcells can still be utilized. Together with compression, these defective (partially disabled) frames can be used to preserve the effective capacity unimpaired.

C. State-of-the-art hybrid LLC insertion policies

This section outlines LHybrid [9] and TAP [32], two state-of-the-art insertion policies for hybrid LLCs.

LHybrid [9] classifies LLC blocks into *loop-blocks* and *non-loop-blocks*. Cache blocks that are not modified during their round trips between L1/L2 and LLC, i.e., read-only blocks that show reuse in the LLC are referred to as loop-blocks. They are

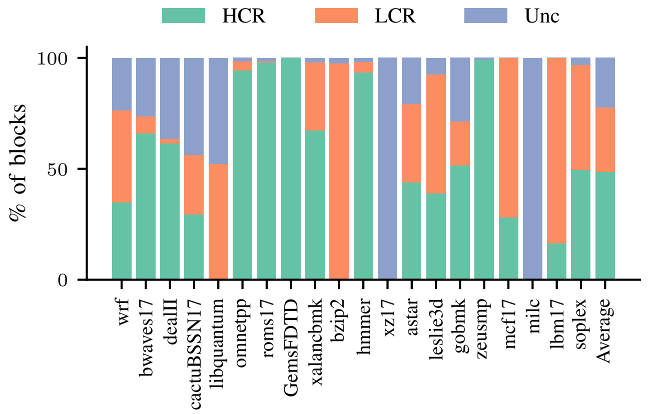


Fig. 2: Block classification regarding its compression ratio for the selected SPEC CPU 2006 and 2017 applications.

of utmost importance in the context of hybrid LLCs because they are ideal residents of the NVM part. LHybrid strives to keep as many loop-blocks (LBs) in the NVM part as possible and steers non-loop-blocks (NLBs) into the SRAM part.

The LHybrid insertion scheme works as follows. All blocks in LLC and L2 are tagged as LB or NLB and this tag is supplied along with the block in both directions. Initially, all blocks entering L2 from the main memory are marked as NLB. A block evicted from L2, marked as NLB and not present in the LLC, is inserted into the SRAM part. Conversely, a block evicted from L2 and tagged as LB, if not in the LLC, is inserted into the NVM part. A read request from L2 that hits LLC implies a previous eviction from L2, and thus a reuse. This read request will tag the block as LB if, and only if, the block is clean. In the LHybrid replacement scheme, for the NVM part, the LRU block is simply evicted (local replacement). In SRAM, the replacement policy first searches for LB blocks, and if found, the most recent LB, in LRU order, is migrated to the NVM part; otherwise, the LRU block is evicted.

Similar to LHybrid, TAP [32] defines *thrashing-blocks* as blocks that have hit in the LLC more than TH_{thrash} times. TAP only inserts clean thrashing-blocks to the NVM part because they are expected to stay longer in the LLC, preventing energy-hungry NVM write operations from other blocks. In terms of NVM insertions, TAP is more conservative than LHybrid, see Figure 1, because a block needs to show reuse more than once (unlike the LHybrid loop-block) to be inserted in the NVM part. We thus use LHybrid as the state-of-the-art reference policy so that results are more comparable in terms of performance.

D. Motivation: quantitative analysis of hybrid LLC insertion policies

As described in the previous section, state-of-the-art insertion policies conservatively target the NVM part, which extends lifetime but sacrifices performance. To demonstrate this, we evaluate ten multi-programmed workloads from the SPEC 2006 and SPEC 2017 benchmarks on a hybrid LLC having 12 NVM-ways and 4 SRAM-ways (see Section V-A and Table III for more details) and show the impact of different configurations on the LLC performance and lifetime in Figure 1. For comparison, we use SRAM-only LLC configurations with 16-ways (best-case) and 4-ways (worst-case, as if the 12-NVM ways were faulty), to determine the upper and lower bounds on the hybrid LLC performance. Both configurations employ the LRU replacement scheme and are compared to the following.

BH, that is NVM-unaware and naively fills data into NVM and SRAM ways implementing a global LRU replacement policy and frame-disabling, achieves performance similar to that of a 16-way SRAM cache, the expected upper limit. The small performance loss compared to an SRAM cache is exclusively due to the increased latency in the STT-RAM ways, since the contents of both caches are exactly the same. However, for a mean endurance of 10^{10}, it takes less than three months for the NVM part to lose 50% of its effective capacity.

LHybrid, which conservatively inserts into the NVM by steering only the loop-blocks into it and employs a local loop-block-aware replacement scheme. As a result, it improves the NVM lifetime by 19.7× compared to BH but at the cost of a significant 11% performance decrease.

TAP is more conservative than LHybrid. Blocks must show higher level of reuse (clean thrashing-blocks) to be inserted in the NVM. It improves the hybrid cache lifetime by 39× compared to BH in exchange for 15% performance drop.

To reduce these wide disparities between performance and lifetime of different configurations, this work investigates hybrid LLC designs that achieve near-BH performance and near-LHybrid lifetime by jointly optimizing for both metrics.

III. Hybrid LLC Architecture

Hybrid LLC designs require intelligent insertion policies supported by the underlying microarchitecutre. This section presents our microarchitectural design decisions carefully adopted to get the most out of the insertion policies.

A. NVM-friendly non-inclusive LLCs

Non-inclusive LLC designs increase caching capacity by only partially duplicating data between the private and shared levels. The non-inclusive relationship allows replacing a block in LLC without having to invalidate copies in the private levels [49]. In an NVM-LLC friendly implementation of this model, a miss in all cache levels involves a main memory access that takes the block directly to the private L1/L2 levels. In turn, the victim block replaced in L2, clean or dirty, is sent to LLC and written if it was not there [13]. Most NVM and hybrid LLCs follow this mostly-exclusive implementation because it reduces the write traffic in the LLC [8], [9], [32], [38].

Figure 3 shows a high-level overview of the proposed hybrid LLC design. Similar to [9], we use a non-inclusive hierarchy and complement it with a fault map, a compression mechanism and an insertion mechanism. The blocks movement in the cache hierarchy follows the above rules, with the exception of block requests with write permission (GetX) coming from L2 that hit in LLC. In this case, LLC returns the block to the private levels and invalidates it in the LLC. This immediate invalidation improves LLC performance because it leaves room for the replacement algorithm to reuse it as needed. The obsolete copy of the invalidated block in LLC will be written anyway when the dirty block is evicted from L1/L2. These coherency features are already implemented in the MOESI_CMP_directory Ruby protocol in gem5 [31].

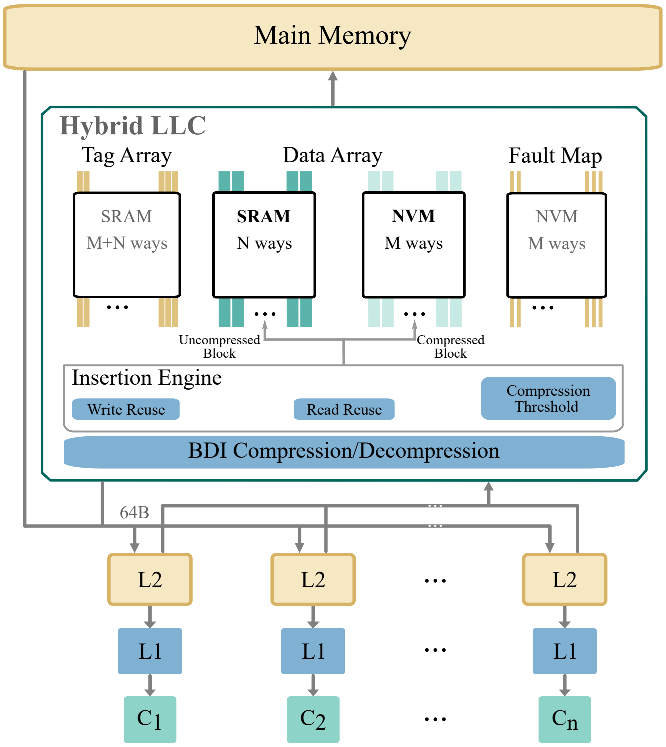


Fig. 3: High-level overview of the hybrid LLC organization.

B. *Fault-tolerant microarchitecture*

NVM bitcells become defective due to repeated write operations. From the architectural perspective, these memory structures must be provided with error correcting codes (ECCs) to detect and correct hard faults. We assume Hamming SECDED protection in all arrays. In particular, we use code (527, 516) for the NVM data array: it can correct one fault and detect up to two faults. Besides, the SECDED can trigger an OS exception that confirms the NVM hard-fault, notifying the identity of the faulty cell so that the corresponding region can be disabled [48], preventing the occurrence of a second uncorrectable error in the same region. Note that this ECC protection does not bring any additional overhead as they already exists in SRAM LLCs to cope with soft and transient faults [3], [5], [17], [26], [40], [47]; for instance, AMD Zen's SRAM LLC employs DECTED protection [40].

Different disabling granularities in the LLC have different performance implications. For example, disabling at frame granularity incurs little overhead but severe degradation of capacity and, thus, performance. Conversely, disabling at a finer granularity, such as at byte level, requires more metadata (overhead) but allows live bytes within a frame to be used [15], [18], [41], [45]. By leveraging compression, these partially disabled frames can be used as functional frames, and the impact of bitcell failures on performance can be effectively mitigated. For instance, if a byte in an NVM cache frame (64B) is disabled, it can still be used to store all cache blocks of compression encodings B8Δ7 and above (≤ 58B), see Table I.

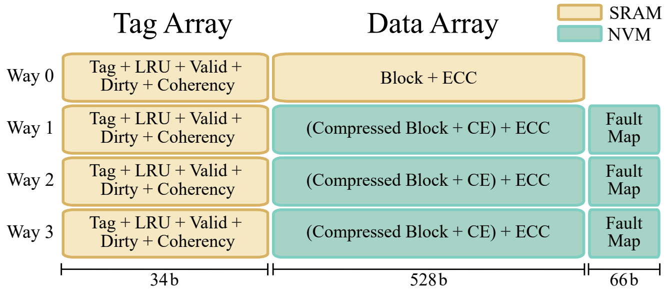
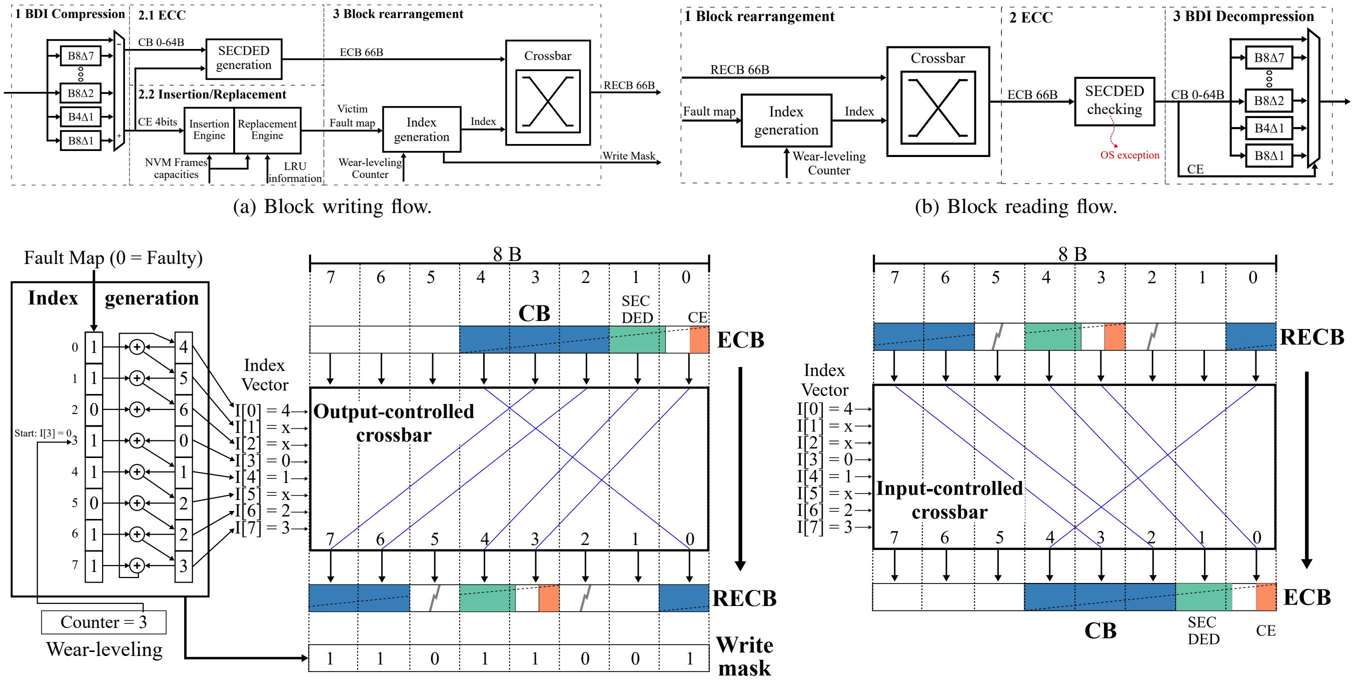

Tag Array
Data Array
SRAM
NVM
Way 0
Way 1
Way 2
Way 3
Tag + LRU + Valid + Dirty + Coherency
Tag + LRU + Valid + Dirty + Coherency
Tag + LRU + Valid + Dirty + Coherency
Tag + LRU + Valid + Dirty + Coherency
Block + ECC
(Compressed Block + CE) + ECC
(Compressed Block + CE) + ECC
(Compressed Block + CE) + ECC
Fault Map
Fault Map
Fault Map
34b
528b
66b

Fig. 4: Example of a four-way cache set split into three NVM ways and one SRAM way, showing fields and their sizes.

Our hybrid LLC employs data compression together with byte-disabling and an intra-frame wear-leveling mechanism to mitigate the effective capacity drop due to bitcell failures. Similar to [15], we maintain a *fault map* of the NVM frames, storing the faultiness information of each byte. Every fault map entry consisting of 66 bits, see Figure 4, is updated every time a byte becomes faulty, i.e., at most 66 times (until the frame is completely dead). This low amount of write accesses leads to no wear problems, and thus the fault map can be implemented in NVM technology. Unlike the fault map, the tag array is written more often as it needs to keep the coherence and replacement information up to date. A hard fault on a tag array bitcell means disabling the whole cache frame. Therefore, we assume that the tag array is realized using SRAM technology that is not subject to wear. The data array, see Figure 4, is split in NVM ways and SRAM ways, typically, with a factor of three NVM ways for every SRAM way [9], [32].

1) *Block writing:* Figure 5a shows the block writing flow in the LLC. For every incoming LLC block, the compressed block (CB, 0-64 bytes) and the chosen compression encoding (CE, 4-bit) information are obtained from the compressor. The extended compressed block (ECB) is then formed by combining the CB with the 4-bit CE and the 11-bit SECDED code The SECDED code is calculated from 516-bit, i.e., the combined CE (4-bit) and 512-bit vector (the CB bit vector plus the required number of zeros to make 512-bit). In parallel to SECDED generation, the insertion engine decides whether to insert it either in an NVM or in an SRAM frame according to our proposed insertion policy, see Section IV; and the replacement algorithm (LRU) selects the target frame. In the case of NVM, the replacement algorithm will look for the target frame among those with an effective capacity greater than or equal to the incoming compressed block (Fit-LRU) [18]. In the case of SRAM, a conventional LRU is employed for replacement, and the block is stored uncompressed.

The NVM part of the hybrid LLC is also provided with a *block rearrangement circuitry*, adopted from the proposal in [15], that scatters the ECBs among the non-faulty bytes of the target frame, generating the sparse block (RECB) and a write



(a) Block writing flow.

(b) Block reading flow.

(c) Example of a 5-byte block rearrangement for writing.

(d) Example of a 5-byte block rearrangement for reading.

Fig. 5: Block writing (a) and reading (b). Example of a 5-byte block rearrangement for writing (c) and reading (d).

mask for selective writing. This block reordering synergistically works with an intra-frame wear-leveling mechanism to evenly distribute write operations' wear among all non-faulty bytes in the target frame. To do this, the block rearrangement circuitry maintains a counter that indicates the byte at which the write operation is performed. This counter is global, shared among all sets, and increments after long periods of time (a few hours or even days) so that the writing region of the frames gets shifted over time [24].

The block rearrangement circuitry consists of two modules: *index generator* and *crossbar*. Figure 5c shows an example of rearranging a 5-byte ECB for scattering into an 8-byte frame with faulty bytes (2 and 5). On the left side of the figure, the index generator computes an index vector I[i] from the fault map and the wear-leveling counter. The optimized implementation of this circuit uses a parallel tree adder [15]. Each index indicates which byte of the ECB is to be placed in each RECB byte (x stands for don't care). For example, I[6]=2 indicates that byte 2 of the ECB is placed in RECB byte 6. Thus, this index vector controls the output ports of the crossbar used to obtain the RECB. In the example, crossbar output 6 selects input 2. For the write mask, the first n positions starting from the wear-leveling counter value, and corresponding to non-faulty bytes, are set to 1s; n being the ECB size.

2) Block reading: A data block access to the NVM data array is depicted in Figure 5b. First, the RECB is read. In parallel, the index vector I[i] is computed again as in block writing. This index vector is now used to obtain the ECB from the RECB. Figure 5d shows a rearrangement example of the same 5-byte RECB of Figure 5c, for delivery to L2. In this case, each index indicates the target output crossbar for each input, e.g., I[6]=2 means that input 6 is forwarded to output 2. Within the already aligned block (ECB), the CE field indicates the length of the compressed block. This value is employed to fill the bytes not used to store the compressed block (CB) with zeros in order to match the SECDED previously generated during the writing. Besides, it selects the corresponding BDI decompressor.

3) Latency overhead: Using VLSI synthesis for 16 nm, the block rearrangement circuitry has proven to be feasible in terms of latency (incurring 0.33 and 0.38 ns for writing and reading, respectively) as well as area and power consumption [15]. The BDI decompressor, that is a SIMD-style vector adder [30], [36], incurs a 2-cycle latency overhead. Note that all the competing mechanisms need SECDED for hard-fault detection. For this reason, the latency incurred by SECDED is not accounted for in the overheads.

IV. Compression-aware insertion policies

Data compression reduces the size of the incoming blocks to the LLC and thereby reduces the number of bytes written to cache frames. Together with byte-disabling, compression can be used to allocate reduced size blocks to partially defective NVM frames, always taking care to level the write wear among the remaining non-faulty bitcells. This section describes how block features such as compressed size, read-reuse and write-reuse information can be leveraged to develop performance-efficient and lifetime-aware content management policies.

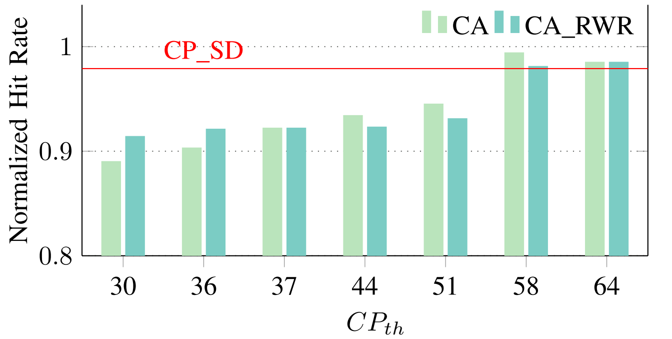


Fig. 6: Hybrid LLC hit rate with different CP_{th} normalized to BH.

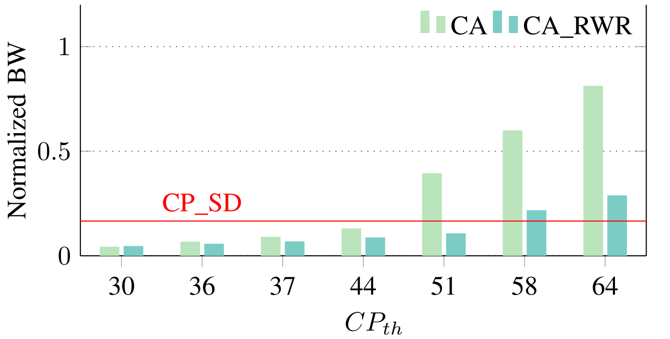


Fig. 7: Normalized BW: average number of bytes written per frame in the NVM part, normalized to BH, varying CP_{th}.

A. Naive compression-aware insertion

We denote blocks whose compressed size is lower than or equal to a given compression threshold (CP_{th}) as *small blocks* and blocks that are either incompressible or whose compressed size is greater than the threshold as *big blocks*. Intuitively, the greater the compression ratio a block achieves, the less harmful it is to write it on the NVM part. A naive compression-aware insertion policy (*CA*), therefore, directs small blocks to NVM ways and big blocks to SRAM ways. Both NVM and SRAM ways follow a local LRU replacement policy.

CA may lead to performance degradation. The performance loss occurs when there is an imbalance between the number of references to blocks allocated in NVM and SRAM ways.

For instance, 100% of the cache blocks are incompressible in the benchmarks `xz17` and `milc`, see Figure 2, and as a result *CA* will direct all blocks to the SRAM ways. On the contrary, in benchmarks such as `GemsFDTD` and `zeusmp`, where almost all cache blocks are highly compressible (HCR), the *CA* policy will only insert blocks into NVM ways. In both cases, one part of the LLC is over-referenced, experiencing many misses and leading to performance degradation.

CA evaluation. The light green bars (CA) in Figure 6 show the LLC hit rate for different CP_{th} values, normalized to BH hit rate. Unless otherwise mentioned, all results in this and the following sections are averaged across ten multiprogrammed mixes of four randomly selected applications from the employed subset of SPEC 2006 and SPEC 2017, see Table V. The normalized hit rate varies between 0.89 and 0.99, with the highest figure being obtained for a CP_{th} of 58. With this CP_{th} value, only uncompressed blocks are inserted into SRAM and the rest into NVM. The attained hit rate is very close to BH, which indicates that this CP_{th} value achieves a block distribution with only little conflict misses compared to BH and, therefore, is well balanced.

The light green bars (CA) in Figure 7 show the number of bytes written to the NVM part for different CP_{th} values normalized to the values obtained in BH. As can be seen, the CP_{th} has a considerable impact on the number of bytes written in the NVM part, varying between 5% and 80% of the writes for BH. With a CP_{th} of 58, the best in terms of hit rate from Figure 6, the number of bytes written is still 40% lower than the BH cache.

B. Read and write reuse aware insertion

As discussed in Section II-C, several works have shown that content management based on block reuse properties can reduce the number of writes in NVM caches [9], [32]. We now discuss how to incorporate read and write reuse into together with the compression-aware insertion policy (CA_RWR).

We classify blocks into three categories based on their reuse properties: blocks that have not yet demonstrated any reuse, blocks with read reuse, and blocks with write reuse. Initially, all blocks are classified as non-reused when copied from the main memory to the cache hierarchy. A hit in the LLC classifies a block into either a read-reused block if it has not been modified or a write-reused block if it has been written at least once. Notice that our read-reuse class corresponds to the loop-blocks in LHybrid while write-reuse and non-reuse blocks correspond to non-loop-blocks.

Table II summarizes CA_RWR, our proposed insertion policy that places blocks in either SRAM or NVM depending on the size of the compressed block and its reuse type:

- Blocks that show *read reuse* are candidates to stay longer in the LLC. Inserting them in the NVM part is beneficial, regardless of the compressed size, because they will stay longer in the LLC, preventing other writes in the frame.
- Blocks that show *write reuse* are candidates to stay for a short time in the LLC due to the invalidate-on-hit coherence policy of LLC requests with write permission (GetX), see Section III-A. Such dirty blocks will be inserted back into LLC when they are evicted from L2. Therefore, they should be placed in SRAM since they are candidates for multiple LLC writes.
- Blocks *without reuse* are inserted either into SRAM or NVM, depending on their compressed size, i.e., small blocks are inserted in NVM and big blocks in SRAM.

Note that NVM frames render partially defective due to write operations. Therefore, a block directed to NVM that does not fit in any NVM frame, because its compressed size is bigger than any of their effective capacities, will be placed in SRAM.

As mentioned above, blocks are initially inserted into SRAM or NVM depending only on their compressed size, since they have not yet shown any reuse. In fact, many of them will be evicted from the LLC without being reused. But for those that do show reuse, the final destination will depend on the type

TABLE II: CA_RWR insertion policy

		Compressed size	
		Small	Big
Reuse	no	NVM	SRAM
	R	NVM	NVM
	W	SRAM	SRAM

of reuse, read or write. Therefore, it is sometimes necessary to migrate blocks between SRAM and NVM arrays: i) blocks initially stored in NVM that show reuse on write, and ii) blocks initially stored in SRAM that show reuse on read. On the one hand, reuse on write is detected when a GetX request hits in LLC and the block is invalidated. Later, when the block is evicted from L2, it will be inserted into SRAM as a write-reused block (regardless of its compressed size). On the other hand, a block initially stored in SRAM that is reused on read remains in SRAM until it is evicted (due to a replacement). At that time, the block is migrated to NVM.

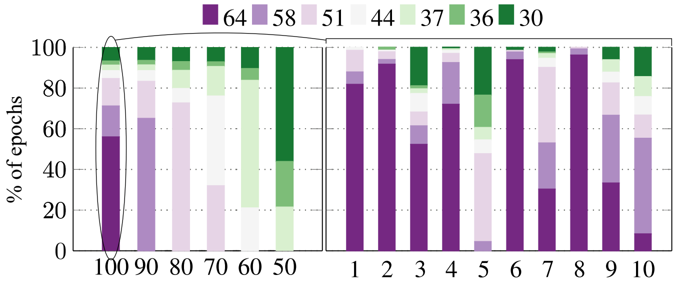
CA_RWR evaluation. The dark green bars (CA_RWR) in Figures 6 and 7 show the normalized LLC hit rate and BW on a CA_RWR cache, varying CP_{th}. Compared to the cache with CA policy, the reuse information in the block insertion policy has a relatively small impact on the hit rate but a noticeable impact on BW, especially for high CP_{th} values. The hit rate is better than the CA cache for small values of CP_{th} and marginally worse for high values of CP_{th}; specifically, the CA_RWR hit rate only increases by 1.9% for CP_{th} values between 30 and 51 and increasing to 5.4% for CP_{th} 58. Compared to the CA, the relative decrease in BW for CA_RWR is significant, reaching 73% for CP_{th} 51. Even though BW varies between 4.4% and 28.6% as CP_{th} varies between 30 and 64.

C. CP_SD insertion: Set Dueling for performance

Figures 6 and 7 illustrate in simplified form the influence of CP_{th} on the normalized hit rates and BW, averaging the results for the entire workload and assuming full capacity in the NVM part. However, for the best CP_{th} selection, one must delve deeper into the impact of two key factors. First, applications may exhibit different behaviors throughout their execution, and second, as the NVM part ages, its capacity decreases.

To analyze the impact of workload time variability and NVM capacity loss we will divide the workload execution time into epochs of fixed duration and calculate the hit rate achieved in each epoch with each CP_{th} value. The different bar colors in Figure 8 show the percentage of epochs for which each CP_{th} value achieves the highest number of cache hits. Specifically, Figure 8a presents the distributions of optimal CP_{th} varying the NVM part capacity from 100 to 50%, and Figure 8b presents the distributions of optimal CP_{th} for each workload in an LLC with 100% capacity in the NVM part.

Let us consider the bar that represents the NVM cache with 100% capacity in Figure 8a. This is the same cache that was used in Figure 6, where we observed that the maximum hit rate is achieved with CP_{th} values 58 or 64. However, Figure 8 reveals that these CP_{th} values are not optimal throughout the entire workload execution. In 30% of the epochs, the optimal hit rate is attained with CP_{th} values less than 58. Furthermore, this percentage varies greatly depending on the workload, reaching 96% in mix 5, see Figure 8b. In Figure 7, we demonstrated that these smaller CP_{th} values are beneficial as they reduce the number of bytes written to the NVM. This implies that a fixed CP_{th} value may easily lead to both sub-optimal overall system performance and sub-optimal NVM lifetime. The impact of varying optimal CP_{th} values becomes even more prominent as the cache loses effective capacity, see Figure 8a, because frames with higher capacities become more scarce.



(a) % of NVM effective capacity (b) By workload

Fig. 8: CA_RWR insertion policy: distribution of CP_{th} achieving the best hit rates across execution epochs, vs. NVM part capacity (a). Uniquely for 100% NVM capacity, the same distribution, but for each of the 10 workloads (b).

For an adaptive CP_{th} value selection mechanism, we propose CP_SD, a new insertion policy using Set Dueling [37] that reacts to both the changing workload behavior and the decreasing capacity of the NVM part. We propose to specialize some sets to use a fixed value of CP_{th}, from 30 to 64. Every value is tested on a group of N/32 sets, where N is the number of sets in the LLC. The rest of the sets follow the group of sets whose CP_{th} brings optimal performance, the group of sets with the maximum number of hits in the previous epoch.

CP_SD evaluation. The red horizontal lines in Figures 6 and 7 indicate hit rate and BW of CP_SD, respectively. CP_SD achieves a hit rate equivalent to the best-case CA_RWR (with values of CP_{th} 58 and 64), which is also comparable to the hit rate of the reference system BH. However, in terms of bytes written, CP_SD reduces the number of writes by a significant 83.4% compared to the BH cache and by 22.9% and 42% compared to CA_RWR for CP_{th} 58 and 64, respectively. Besides, we perform these experiments varying the epoch size and our evaluation shows that 2M cycles achieves the best Set Dueling performance. This value is used for the all evaluations in the following sections.

D. CP_SD_Th: CP_SD for both performance and lifetime

By using the CP_SD insertion policy it may happen that a very small difference in performance in an epoch determines the selection of a CP_{th} value that produces a much larger number of bytes written to the NVM part. We thereby introduce CP_SD_Th: a variation that seeks a better tradeoff between performance and lifetime. It is based on selecting CP_{th} considering not only the number of hits in LLC but also *the number of bytes written to the NVM part.*

We have not found a simple arithmetic function that combines both metrics to compute the Set Dueling winner, largely because their ranges of variability are very different and highly dependent on workload and NVM cache capacity. Alternatively, we will make a rule-based decision with two thresholds: i) *Th*, the maximum percentage of cache hits we are willing to sacrifice, and ii) *Tw*, the minimum percentage of NVM bytes written decrement we require to admit a performance loss. As usual, the rule for choosing CP_{th} is applied at the beginning of each epoch by first looking for the value i of CP_{th} that achieved the maximum number of hits in the previous epoch. Then, the smallest value j of CP_{th} that satisfies the following inequalities is selected:

$$H(j) > H(i) * (1 - \frac{Th}{100}) \ \& \ W(j) < W(i) * (1 - \frac{Tw}{100}) \quad (1)$$

Where $H(x)$ and $W(x)$ are the number of hits and bytes written to NVM, respectively, in the sampler sets whose CP_{th} was x in the previous epoch.

CP_SD evaluation by varying *Th* and *Tw*. Our evaluation of Set Dueling with different values for the parameters *Th* and *Tw* shows that the sensitivity of the number of hits and the number of bytes written to NVM to the parameter *Tw* is very low; therefore, Figure 9 only shows data for values of *Th* 0, 2, 4, 6, and 8% (different colors) keeping *Tw* = 5%. The different shapes represent different NVM capacities: the circles, triangles and squares correspond to NVM part capacities of 100, 90 and 80%, respectively. Both the number of hits and number of bytes written to NVM are normalized to BH with 100% NVM capacity.

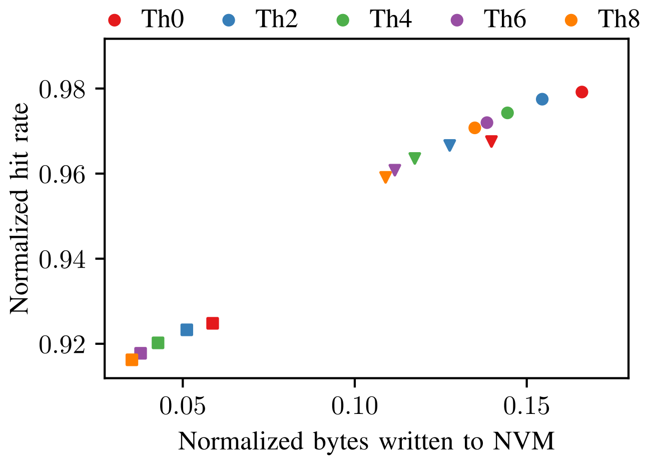


Fig. 9: Hit rate and number of bytes written to NVM normalized to BH, for different *Th* [0-8%] and different NVM capacities: 100-90-80% (circles, triangles, and squares). *Tw* set at 5%.

The observed trend is similar for the three NVM capacities analyzed. Increasing the value of *Th* always produces a decrease in both the number of hits and number of bytes written to NVM. However, the relative decrease is much larger in the number of bytes written, especially when the cache capacity decreases. For example, going from *Th* = 0 to *Th* = 8% for 80% NVM capacity, the number of cache hits decreases from 0.925 to 0.916 (1.0% reduction) while the number of writes decreases from 0.059 to 0.035 (40.7% reduction). However, the same *Th* variation for a 100% NVM capacity results in a 18.7% relative decrease in the bytes written to NVM.

Note that the 8% limit on *Th* only produces a real loss of 1% in hits, since a) in many epochs no change in CP_{th} is applied to decrease hits because in return there is not enough reduction in bytes written, and b) when the change is applied the decrease in hits will be between 0 and 8%. In addition, by construction, the decrease in bytes written can be much greater than the decrease in hits because the CP_{th} is only changed in those epochs in which the decrease in bytes written is large and the decrease in hits is small.

V. Results and Analysis

This section evaluates the impact of compression-aware insertion policies on the lifetime and performance of a hybrid LLC, comparing them to the state-of-the-art proposals described in section II-C, see Table III: CP_SD and CP_SD_Th refer to insertion policies introduced in Sections IV-C and IV-D, respectively.

TABLE III: Summary of tested insertion policies.

Name	Disabling granularity	Data Comp.	NVM aware
BH	Frame	No	No
BH_CP	Byte	Yes	No
LHybrid	Frame	No	Yes
CP_SD	Byte	Yes	Yes
CP_SD_Th	Byte	Yes	Yes

Section V-A introduces the experimental setup, including system specification, simulation infrastructure, and benchmark suites. Section V-B compares CP_SD insertion policies with state-of-the-art in terms of performance and NVM lifetime, showing how the rule-based mechanism effectively trades performance in exchange for lifetime by tuning CP_SD Th. Sections V-C, V-D, V-E, and V-F present sensitivity studies concerning to the ratio of NVM size vs. SRAM size, the impact of the coefficient of variation on system performance and NVM lifetime, the size of L2 (x2), and the increase in NVM access latency (x1.5); respectively. Finally, in Section V-G the cost of fine-grain disabling, i.e. the overhead of the byte fault map, is discussed, evaluating its impact by reducing the number of ways in the NVM part by an amount equivalent to such overhead.

A. Experimental setup

We use a 4-core system with private L1 (instructions and data) and L2 caches and a shared hybrid LLC, as detailed in Table IV. The hybrid LLC is non-inclusive and partitioned into four banks. The system uses a directory-based MOESI coherence protocol, and a crossbar that connects the private levels (L2) with the LLC banks and the directory.

As for the simulation, we use two different infrastructures: a trace-driven simulator called HyCSim [16] for design space

TABLE IV: System specification.

Cores	4, ARMv8, out-of-order (up to 8 inst/cycle), 3.5 GHz
Coherence Protocol	MOESI, directory distributed among LLC banks 64 B data block in all levels
L1	Private, 32 KB D, 32 KB I, 4 ways, LRU 3-cycles load-use delay. Fetch on write miss
L2	Private, 128 KB D, 128 KB I, 16 ways, LRU 11-cycle load-use delay. Fetch on write miss
Hybrid LLC	Shared, non-inclusive, 4 banks, 4MB/bank
	4 SRAM ways, 28-cycle load-use delay (4-cycle D-array)
	12 NVM ways, 32-cycle load-use delay (8-cycle D-array) +2 cycles for decompression and block rearrangement 20-cycle data array write latency Endurance: mean = 10^{10} writes, cv = 0.2
Main Memory	1 memory controller, DDR4 1 channel, 8GB/channel (1200 MHz)
NoC	Crossbar between NV-LLC banks and L2s. 32 B flits

TABLE V: SPEC CPU 2006 and 2017 mixes.

mix 1	zeusmp06 gobmk06 dealII06 bzip206
mix 2	hmmer06 bzip206 wrf06 roms17
mix 3	zeusmp06 cactuBSSN17 hmmer06 soplex06
mix 4	omnetpp06 astar06 milc06 libquantum06
mix 5	xalancbmk06 leslie3d06 bwaves17 mcf17
mix 6	lbm17 xz17 GemsFDTD06 wrf06
mix 7	cactuBSSN17 dealII06 libquantum06 xalancbmk06
mix 8	gobmk06 milc06 mcf17 lbm17
mix 9	xz17 astar06 bwaves17 soplex06
mix 10	GemsFDTD06 omnetpp06 roms17 leslie3d06

exploration and for figures in Section IV, and gem5 [31] for the detailed cycle-accurate full-system simulation presented in this Section. To estimate hybrid LLC latencies we use NVSim [14].

We adapted the forecasting procedure introduced in [15] to the hybrid LLC scenario. It allows to accurately measure the impact of different insertion policies on the evolution over time of performance and capacity of the NVM part, taking into account the disabling of frames or bytes and the use of compression. BH and LHybrid are provided with frame-disabling to tolerate hard faults while BH_CP and CP_SD employ byte-disabling together with data compression. The forecasting procedure alternates between simulation and prediction phases. The simulation phase starts reading the NVM LLC state; for instance, in BH_CP and CP_SD such state is the fault map of every NVM frame, then it performs a full system simulation reporting several indexes of interest, e.g., the write rate on NVM frames, system IPC, and LLC hit rate. The prediction phase receives such write rates, computes the next k NVM bitcells to become faulty, and update the fault map for the next simulation. In this work, the forecasting procedure advances in time until the NVM part loses 50% of its capacity, but there is no problem in reaching full depletion. The IPC evolution depicted in Section V figures is obtained at each simulation phase, computing the arithmetic mean of the IPCs of the mixes conforming the workload.

The experimental evaluation is made on the ten multi-programmed workloads shown in Table V. Mixes are formed by randomly selecting applications from the SPEC CPU 2006 and 2017 suites, leaving out applications that do not show substantial memory activity [6], [21]. Fast forward is performed for 2 billion instructions, warm-up for 60 million cycles, and then 200 million cycles are simulated to collect statistics.

B. Performance vs. Lifetime

The solid lines in Figure 10a show the performance evolution over time of the proposed insertion policies for hybrid LLCs (CP_SD, CP_SD_Th), comparing them with BH, BH_CP and LHybrid. Dashed lines mark the upper and lower performance bounds. The upper bound corresponds to a a 16-way SRAM LLC, while the lower bound corresponds to a hybrid LLC whose NVM part is fully impaired (4w SRAM). Finally, we also show performance of a base hybrid LLC with compression (BH_CP). BH_CP uses compression and byte-disabling, but it is oblivious to the NVM wear due to writing and uses a global fit-LRU replacement policy. In BH_CP, the victim frame is the one containing the LRU block from among those occupying frames with a size greater than or equal to the size of the block to be inserted, either in the SRAM or NVM.

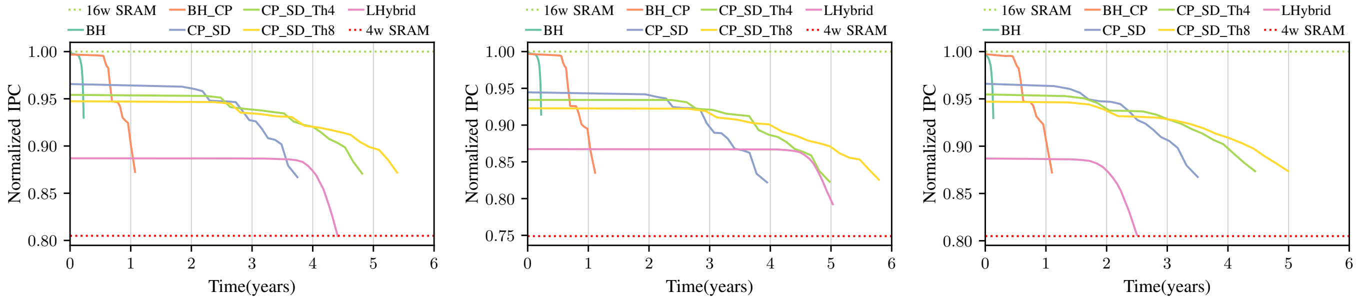
Initially, in the first few months, the performance of BH_CP is similar to that of BH because both use a global, unconstrained LRU replacement algorithm. However, compression and byte disabling, even without compression-aware insertion and replacement policies, reduce the number of writes in the NVM part and manage to extend the lifetime of BH_CP by 4.8× with respect to BH. Still, the effective capacity of 50% is reached in 13 months, far short of the LHybrid 53 months.

CP_SD, the performance-optimized configuration, manages to delay the 50% capacity loss to 45 months, multiplying BH lifetime by 16.8×. This increase in lifetime is achieved at the cost of a performance loss of only 3.3% at the beginning of the cache lifetime. This performance level remains almost unchanged beyond two years. From this point on, the NVM part starts to gradually lose capacity, which translates into a gradual drop in performance. Compared to LHybrid, CP_SD reaches the 50% capacity 8 months earlier but always maintains a significant difference in performance, especially in the long initial stage where it reduces the performance loss compared to the upper limit (dotted green line) from 11.2% of LHybrid to only 3.3%.

As introduced in Section IV-D, CP_SD can be further tuned to trade performance in exchange for lifetime and vice versa. Figure 10a shows the results for Th 4 and 8%, keeping Tw = 5. CP_SD_Th4 and CP_SD_Th8 achieve 28% and 44% increase in lifetime compared to CP_SD, in exchange for 1.1% and 1.9% performance degradation, respectively. Compared to LHybrid, CP_SD_Th4 and CP_SD_Th8 achieve 9% and 22% more lifetime while keeping 7.6% and 6.8% higher performance, respectively.

C. SRAM-NVM proportion variation

We analyze the behavior of the hybrid LLC by increasing asymmetry between the sizes of the SRAM and NVM. Specifically, Figure 10b shows hybrid LLCs with a 3-way SRAM and a 13-way NVM part. The decrease in the number of SRAM ways has little impact on BH and BH_CP because block insertion and replacement do not depend on this parameter. The original 12-way NVM wears similarly in this new 3/13-way



(a) Performance evaluation

(b) NVM-SRAM proportion variation

(c) cv = 0.25

Fig. 10: Performance evolution until the NVM part reaches 50% effective capacity for default parameters and different CP_SD Th (a), NVM-SRAM proportion (b), and coefficient of variation of the NVM endurance distribution, $cv = 0.25$ (c).

configuration. The only additional performance degradation occurs due to the loss of capacity of the new NVM way.

In LHybrid, the SRAM part acts as a read-reuse detector for the NVM part. By decreasing the number of SRAM ways, less read reuse is detected, thereby resulting in fewer block insertions to the NVM part. This translates into a 14% longer lifetime and a 2.2% lower performance compared to the 4/12 configuration. For CP_SD-based policies, increasing the SRAM/NVM asymmetry also means a slight increase in lifetime (5.5%, 3.4%, and 7.4%) and a slight drop in performance (2.2%, 2.1%, and 2.6%), for CP_SD, CP_SD_Th4, and CP_SD_Th8, respectively.

D. Impact of cv on performance and lifetime

We model the endurance of NVM memory bitcells using a normal distribution, see Section II-A. The coefficient of variation cv of this distribution reflects the memory manufacturing variability. This parameter has a significant impact on the evolution of the LLC capacity. A higher coefficient of variation implies a larger dispersion in the number of writes supported by each cell. Consequently, the first faults occur earlier, thereby impacting frame- and byte-disabling techniques.

In Figure 10c we have repeated the experimentation assuming a higher manufacturing variability, changing the coefficient of variation cv from 0.20 to 0.25 and keeping the mean $\mu = 10^{10}$ constant. The lifetime of frame-disabling caches is drastically reduced as the coefficient of variation increases. The time to reach 50% capacity goes from 2.7 months to 1.6 months for BH and from 53 to 30 months for LHybrid. However, the impact on the lifetime of the models with byte-disabling is much smaller: for BH_CP, it remains the same, for CP_SD it drops from 45 to 42 months, for CP_SD_Th4, it drops from 58 to 53 months, and for CP_SD_Th8 it drops from 65 to 60 months. Consequently, CP_SD-based policies manage to significantly improve both performance and lifetime over LHybrid as cv grows: CP_SD, CP_SD_Th4, and CP_SD_Th8 achieve 1.4×, 1.8×, and 2× greater lifetime while maintaining 8.9%, 7.6%, and 6.8% greater performance, respectively.

E. L2 size sensitivity

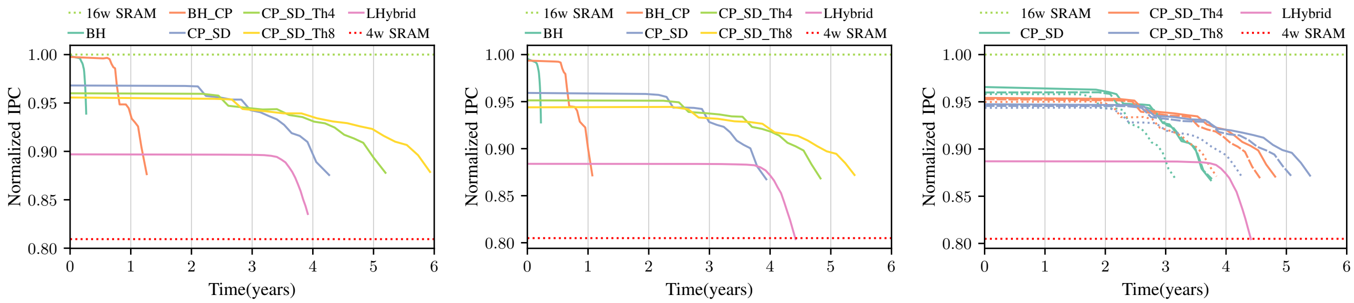
Figure 11a shows performance evolution when L2 size is increased from 128 to 256 KB. Increasing the L2 size means increasing the overall system performance. Besides, it also means that the L2 can filter more write operations from the hybrid LLC, which translates into a slight increase in lifetime. Compared to the systems in Figure 10a, lifetime increases 19%, 18%, 14%, 8%, and 10% for BH, BH_CP, CP_SD, CP_SD_Th4, and CP_SD_Th8, respectively. On the contrary, the lifetime of LHybrid decreases by 11%. As already mentioned, in LHybrid, a block must experience a hit in the SRAM ways before being inserted in the NVM ones. When the L2 capacity is increased, the SRAM activity and the number of block-fills decrease, so the blocks spend more time in the SRAM ways. The more time a block is present in the LLC, the higher the probability of it being hit and detected as a loop-block. Detecting more loop-blocks results in an overall increase in the write rate to the NVM part and, thereby, a lifetime reduction.

F. NVM latency sensitivity

NVM technology and system integration may have various optimization targets. As a result, the NVM latency might vary significantly. Figure 11b shows results for an NVM latency equal to 1.5× the original one, i.e. the NVM data array read latency is increased from 8 to 12 cycles. As expected, policies that insert more aggressively on the NVM part are more affected by increased latency than those that insert more conservatively. For instance, compared to Figure 10a, the performance at the beginning of CP_SD, CP_SD_Th4, CP_SD_Th8, and LHybrid decreases by 0.7, 0.3, 0.4, and 0.4%, respectively. This small drop in performance translates into a slight reduction in the NVM write rate, and these configurations experience a slight increase in lifetime. However, overall, there is no drastic change in the hybrid LLC performance and lifetime.

G. Overhead analysis & Equalizing costs

In the previous sections it has been shown that insertion policies tailored to compression and byte disabling improve the state of the art in both performance and lifetime, and achieve this even with a higher read latency due to rearrangement and decompression, see Section III-B3. But of course, this is at the cost of a non-negligible storage overhead. It is therefore necessary to re-evaluate the comparison, using the same total storage in the systems without and with compression.



(a) L2 size = 256KB

(b) 1.5× NVM latency

(c) Equalizing costs

Fig. 11: Performance evolution until the NVM part reaches 50% of effective capacity, increasing L2 size to 256 KB (a), increasing 50% the NVM data array latency (b), and equalizing costs of CP_SD systems with LHybrid (c).

All evaluated configurations employ SECDED protection, able to point out the faulty bitcell and disable the corresponding region, see Section III-B. Regarding the metadata overhead, the baseline configuration (BH) and the state-of-the-art (LHybrid, TAP) are provided with frame-disabling and hence require one bit per NVM frame. BH_CP and CP_SD, similar to [15], need a fault map to disable at byte granularity: one bit per NVM byte. Compared to LHybrid, CP_SD incurs a storage overhead of 8.6%, i.e., 12.3% of the NVM data array.

Hence, we now analyze the performance and lifetime of CP_SD, CP_SD_Th4 and CP_SD_Th8 with a similar storage cost to LHybrid. We thereby reduce the number of NVM ways of these caches from 12 to 11 and 10, which results in 1.8% higher and 5.2% lower storage cost than LHybrid, respectively.

The solid, dashed, and dotted pattern lines in Figure 11c show the data from caches with 12, 11, and 10 NVM ways, respectively. All CP_SD configurations decrease their performance and lifetime when the number of ways is reduced. Nonetheless, the normalized IPC in the initial phase of the cache lifetime is in all configurations significantly higher than that of LHybrid. The CP_SD_Th8 cache with 10 NVM ways, with a 5.2% lower storage cost than LHybrid, manages to increase the normalized IPC of LHybrid by 6.4% during the first two years and maintains a higher IPC throughout the whole life of the cache.

VI. Conclusions

Hybrid LLCs bridge the performance and capacity gap between the high-performance SRAM and high-capacity NVM LLC designs. Existing hybrid LLC proposals particularly optimize for LLC lifetime by only conservatively inserting cache blocks into the NVM ways. These lifetime-focused optimizations significantly reduce the LLC performance.

In this paper, we leverage that 78% of the total LLC blocks are compressible to some extent and thus propose fault-aware policies to smartly steer cache blocks into the NVM or SRAM ways by analyzing both the cache block read-/write-reuse behavior and its compressed size. We use Set Dueling to identify the best-performing compression threshold, CP_{th}, depending on the workload behavior and the NVM capacity. Our proposed insertion policy can be further tuned to trade performance in exchange for NVM lifetime by adjusting the NVM write rate with a rule-based mechanism.

Our evaluations show that our insertion policies with Set Dueling nearly achieve the performance of a SRAM cache with the same associativity while improving lifetime by 17× compared to a hybrid NVM-unaware LLC. For a fair comparison, we adapt state-of-the-art hybrid LLC insertion policies to a fault-aware environment. Our design outperforms the state-of-the-art by 9% while attaining a comparable lifetime. Besides, the rule-based mechanism can achieve, for instance, 9% and 22% more lifetime than LHybrid while still outperforming it by 7.6% and 6.8%, respectively.

Acknowledgements

We would like to thank the reviewers for their constructive feedback. This work was partially funded by the HiPEAC collaboration grant 2020, the Center for Advancing Electronics Dresden (cfaed), the German Research Council (DFG) through the HetCIM project (502388442) under the Priority Program on ‘Disruptive Memory Technologies’ (SPP 2377), and from grants (1) PID2019-105660RB-C21 and PID2019-107255GB-C22/AEI/10.13039/501100011033 from Agencia Estatal de Investigación (AEI), and (2) gaZ: T5820R research group from Dept. of Science, University and Knowledge Society, Government of Aragon.

References

[1] B. Abali, B. Blaner, J. Reilly, M. Klein, A. Mishra, C. B. Agricola, B. Sendir, A. Buyuktosunoglu, C. Jacobi, W. J. Starke, H. Myneni, and C. Wang, “Data compression accelerator on ibm power9 and z15 processors : Industrial product,” in *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, 2020, pp. 1–14.

[2] S. Agarwal, “Linovo: Longevity enhancement of non-volatile caches by placement, write-restriction & victim caching in chip multi-processors,” Ph.D. dissertation, 2020.

[3] J. Ahn, S. Yoo, and K. Choi, “Selectively protecting error-correcting code for area-efficient and reliable stt-ram caches,” in *2013 18th Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 2013, pp. 285–290.

[4] J. Ahn, S. Yoo, and K. Choi, “Prediction hybrid cache: An energy-efficient stt-ram cache architecture,” *IEEE Transactions on Computers*, vol. 65, no. 3, pp. 940–951, 2015.

[5] K. Bhattacharya, N. Ranganathan, and S. Kim, "A framework for correction of multi-bit soft errors in l2 caches based on redundancy," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 17, no. 2, pp. 194–206, 2008.

[6] J. Bucek, K.-D. Lange, and J. v. Kistowski, "Spec cpu2017: Next-generation compute benchmark," in *Companion of the 2018 ACM/SPEC International Conference on Performance Engineering*, ser. ICPE '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 41–42. [Online]. Available: https://doi.org/10.1145/3185768.3185771

[7] J. Chang, M. Huang, J. Shoemaker, J. Benoit, S.-L. Chen, W. Chen, S. Chiu, R. Ganesan, G. Leong, V. Lukka, S. Rusu, and D. Srivastava, "The 65-nm 16-mb shared on-die l3 cache for the dual-core intel xeon processor 7100 series," *IEEE Journal of Solid-State Circuits*, vol. 42, no. 4, pp. 846–852, 2007.

[8] M.-T. Chang, S.-L. Lu, and B. Jacob, "Impact of cache coherence protocols on the power consumption of stt-ram-based llc," in *The Memory Forum Workshop*, 2014.

[9] H.-Y. Cheng, J. Zhao, J. Sampson, M. J. Irwin, A. Jaleel, Y. Lu, and Y. Xie, "Lap: Loop-block aware inclusion properties for energy-efficient asymmetric last level caches," in *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, 2016, pp. 103–114.

[10] Y.-D. Chih, Y.-C. Shih, C.-F. Lee, Y.-A. Chang, P.-H. Lee, H.-J. Lin, Y.-L. Chen, C.-P. Lo, M.-C. Shih, K.-H. Shen, H. Chuang, and T.-Y. J. Chang, "13.3 a 22nm 32mb embedded stt-mram with 10ns read speed, 1m cycle write endurance, 10 years retention at 150° c and high immunity to magnetic field interference," in *2020 IEEE International Solid-State Circuits Conference-(ISSCC)*. IEEE, 2020, pp. 222–224.

[11] S. Cho and H. Lee, "Flip-n-write: A simple deterministic technique to improve pram write performance, energy and endurance," in *2009 42nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2009, pp. 347–357.

[12] M. Cintra and N. Linkewitsch, "Characterizing the impact of process variation on write endurance enhancing techniques for non-volatile memory systems," in *Proceedings of the ACM SIGMETRICS/international conference on Measurement and modeling of computer systems*, 2013, pp. 217–228.

[13] J. Cook, J. Cook, and W. Alkohlani, "A statistical performance model of the opteron processor," *SIGMETRICS Perform. Eval. Rev.*, vol. 38, no. 4, p. 75–80, mar 2011. [Online]. Available: https://doi.org/10.1145/1964218.1964231

[14] X. Dong, C. Xu, Y. Xie, and N. P. Jouppi, "Nvsim: A circuit-level performance, energy, and area model for emerging nonvolatile memory," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 31, no. 7, pp. 994–1007, 2012.

[15] C. Escuin, P. Ibañez, D. Navarro, T. Monreal, J. M. Llaberia, and V. Viñals, "L2c2: Last-level compressed-cache nvm and a procedure to forecast performance and lifetime," *PLOS ONE*, 2023.

[16] C. Escuin, A. A. Khan, P. Ibañez, T. Monreal, V. Viñals, and J. Castrillon, "Hycsim: A rapid design space exploration tool for emerging hybrid last-level caches," in *System Engineering for constrained embedded systems (DroneSE and RAPIDO '22)*. New York, NY, USA: ACM, 2022, pp. 1–6.

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

[18] A. Ferreron, D. Suarez-Gracia, J. Alastruey-Benede, T. Monreal-Arnal, and P. Ibanez, "Concertina: Squeezing in cache content to operate at near-threshold voltage," *IEEE Transactions on Computers*, vol. 65, no. 3, pp. 755–769, 2015.

[19] F. Hameed and J. Castrillon, "A novel hybrid dram/stt-ram last-level-cache architecture for performance, energy, and endurance enhancement," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 27, no. 10, pp. 2375–2386, 2019.

[20] A. Hankin, T. Shapira, K. Sangaiah, M. Lui, and M. Hempstead, "Evaluation of non-volatile memory based last level cache given modern use case behavior," in *2019 IEEE International Symposium on Workload Characterization (IISWC)*. IEEE, 2019, pp. 143–154.

[21] J. L. Henning, "Spec cpu2006 benchmark descriptions," *ACM SIGARCH Computer Architecture News*, vol. 34, no. 4, pp. 1–17, 2006.

[22] S. Hong, B. Abali, A. Buyuktosunoglu, M. B. Healy, and P. J. Nair, "Touché: Towards ideal and efficient cache compression by mitigating tag area overheads," in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, 2019, pp. 453–465.

[23] J. Izraelevitz, J. Yang, L. Zhang, J. Kim, X. Liu, A. Memaripour, Y. J. Soh, Z. Wang, Y. Xu, S. R. Dulloor *et al.*, "Basic performance measurements of the intel optane dc persistent memory module," *arXiv preprint arXiv:1903.05714*, 2019.

[24] A. Jadidi, M. Arjomand, M. K. Tavana, D. R. Kaeli, M. T. Kandemir, and C. R. Das, "Exploring the potential for collaborative data compression and hard-error tolerance in pcm memories," in *2017 47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 2017, pp. 85–96.

[25] M. R. Jokar, M. Arjomand, and H. Sarbazi-Azad, "Sequoia: A high-endurance nvm-based cache architecture," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 24, no. 3, pp. 954–967, 2016.

[26] W. Kang, W. Zhao, Z. Wang, Y. Zhang, J.-O. Klein, Y. Zhang, C. Chappert, and D. Ravelosona, "A low-cost built-in error correction circuit design for stt-mram reliability improvement," *Microelectronics Reliability*, vol. 53, no. 9-11, pp. 1224–1229, 2013.

[27] B. Kim, P. J. Nair, and S. Hong, "Adam: Adaptive block placement with metadata embedding for hybrid caches," in *2020 IEEE 38th International Conference on Computer Design (ICCD)*. IEEE, 2020, pp. 421–424.

[28] N. Kim, J. Ahn, W. Seo, and K. Choi, "Energy-efficient exclusive last-level hybrid caches consisting of sram and stt-ram," in *2015 IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC)*. IEEE, 2015, pp. 183–188.

[29] K. Korgaonkar, I. Bhati, H. Liu, J. Gaur, S. Manipatruni, S. Subramoney, T. Karnik, S. Swanson, I. Young, and H. Wang, "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)*, 2018, pp. 315–327.

[30] A. S. Kozhin and A. V. Surchenko, "Evaluation of cache compression for elbrus processors," in *2018 Engineering and Telecommunication (EnT-MIPT)*, 2018, pp. 135–139.

[31] J. Lowe-Power, A. M. Ahmad, A. Akram, M. Alian, R. Amslinger, M. Andreozzi, A. Armejach, N. Asmussen, B. Beckmann, S. Bharadwaj, G. Black, G. Bloom, B. R. Bruce, D. R. Carvalho, J. Castrillon, L. Chen, N. Derumigny, S. Diestelhorst, W. Elsasser, C. Escuin, M. Fariborz, A. Farmahini-Farahani, P. Fotouhi, R. Gambord, J. Gandhi, D. Gope, T. Grass, A. Gutierrez, B. Hanindhito, A. Hansson, S. Haria, A. Harris, T. Hayes, A. Herrera, M. Horsnell, S. A. R. Jafri, R. Jagtap, H. Jang, R. Jeyapaul, T. M. Jones, M. Jung, S. Kannoth, H. Khaleghzadeh, Y. Kodama, T. Krishna, T. Marinelli, C. Menard, A. Mondelli, M. Moreto, T. Mück, O. Naji, K. Nathella, H. Nguyen, N. Nikoleris, L. E. Olson, M. Orr, B. Pham, P. Prieto, T. Reddy, A. Roelke, M. Samani, A. Sandberg, J. Setoain, B. Shingarov, M. D. Sinclair, T. Ta, R. Thakur, G. Travaglini, M. Upton, N. Vaish, I. Vougioukas, W. Wang, Z. Wang, N. Wehn, C. Weis, D. A. Wood, H. Yoon, and Éder F. Zulian, "The gem5 simulator: Version 20.0+," *arXiv preprint arXiv:2007.03152*, 2020.

[32] J.-Y. Luo, H.-Y. Cheng, I.-C. Lin, and D.-W. Chang, "Tap: Reducing the energy of asymmetric hybrid last-level cache via thrashing aware placement and migration," *IEEE Transactions on Computers*, vol. 68, no. 12, pp. 1704–1719, 2019.

[33] H. Noguchi, K. Ikegami, S. Takaya, E. Arima, K. Kushida, A. Kawasumi, H. Hara, K. Abe, N. Shimomura, J. Ito, S. Fujita, T. Nakada, and H. Nakamura, "7.2 4mb stt-mram-based cache with memory-access-aware power optimization and write-verify-write / read-modify-write scheme," in *2016 IEEE International Solid-State Circuits Conference (ISSCC)*, 2016, pp. 132–133.

[34] P. M. Palangappa and K. Mohanram, "Compex: Compression-expansion coding for energy, latency, and lifetime improvements in mlc/tlc nvm," in *2016 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2016, pp. 90–101.

[35] P. M. Palangappa and K. Mohanram, "Castle: Compression architecture for secure low latency, low energy, high endurance nvms," in *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*, 2018, pp. 1–6.

[36] G. Pekhimenko, V. Seshadri, O. Mutlu, M. A. Kozuch, P. B. Gibbons, and T. C. Mowry, "Base-delta-immediate compression: Practical data compression for on-chip caches," in *2012 21st International Conference on Parallel Architectures and Compilation Techniques (PACT)*. IEEE, 2012, pp. 377–388.

[37] M. K. Qureshi, A. Jaleel, Y. N. Patt, S. C. Steely, and J. Emer, "Set-dueling-controlled adaptive insertion for high-performance caching," *IEEE micro*, vol. 28, no. 1, pp. 91–98, 2008.

[38] R. Rodríguez-Rodríguez, J. Díaz, F. Castro, P. Ibáñez, D. Chaver, V. Viñals, J. C. Saez, M. Prieto-Matias, L. Piñuel, T. Monreal, and J. M. Llabería, "Reuse Detector: Improving the Management of STT-RAM SLLCs," *The Computer Journal*, vol. 61, no. 6, pp. 856–880, 10 2017. [Online]. Available: https://doi.org/10.1093/comjnl/bxx099

[39] Y. J. Song, J. H. Lee, S. H. Han, H. C. Shin, K. H. Lee, K. Suh, D. E. Jeong, G. H. Koh, S. C. Oh, J. H. Park, S. O. Park, B. J. Bae, O. I. Kwon, K. H. Hwang, B. Seo, Y. Lee, S. H. Hwang, D. S. Lee, Y. Ji, K. Park, G. T. Jeong, H. S. Hong, K. P. Lee, H. K. Kang, and E. S. Jung, "Demonstration of highly manufacturable stt-mram embedded in 28nm logic," in *2018 IEEE International Electron Devices Meeting (IEDM)*, 2018, pp. 18.2.1–18.2.4.

[40] D. Suggs, M. Subramony, and D. Bouvier, "The amd "zen 2" processor," *IEEE Micro*, vol. 40, no. 2, pp. 45–52, 2020.

[41] J. Wang, X. Dong, and Y. Xie, "Point and discard: a hard-error-tolerant architecture for non-volatile last level caches," in *Proceedings of the 49th Annual Design Automation Conference*, 2012, pp. 253–258.

[42] J. Wang, X. Dong, Y. Xie, and N. P. Jouppi, "i 2 wap: Improving non-volatile cache lifetime by reducing inter-and intra-set write variations," in *2013 IEEE 19th International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2013, pp. 234–245.

[43] Z. Wang, D. A. Jiménez, C. Xu, G. Sun, and Y. Xie, "Adaptive placement and migration policy for an stt-ram-based hybrid cache," in *2014 IEEE 20th International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2014, pp. 13–24.

[44] L. Wei, J. G. Alzate, U. Arslan, J. Brockman, N. Das, K. Fischer, T. Ghani, O. Golonzka, P. Hentges, R. Jahan, P. Jain, B. Lin, M. Meterelliyoz, J. O'Donnell, C. Puls, P. Quintero, T. Sahu, M. Sekhar, A. Vangapaty, C. Wiegand, and F. Hamzaoglu, "13.3 a 7mb stt-mram in 22ffl finfet technology with 4ns read sensing time at 0.9 v using write-verify-write scheme and offset-cancellation sensing technique," in *2019 IEEE International Solid-State Circuits Conference-(ISSCC)*. IEEE, 2019, pp. 214–216.

[45] C. Wilkerson, H. Gao, A. R. Alameldeen, Z. Chishti, M. Khellah, and S.-L. Lu, "Trading off cache capacity for reliability to enable low voltage operation," *ACM SIGARCH computer architecture news*, vol. 36, no. 3, pp. 203–214, 2008.

[46] J. Wuu, D. Weiss, C. Morganti, and M. Dreesen, "The asynchronous 24mb on-chip level-3 cache for a dual-core itanium/sup /spl reg//-family processor," in *ISSCC. 2005 IEEE Int. Digest of Technical Papers. Solid-State Circuits Conf., 2005.*, 2005, pp. 488–612 Vol. 1.

[47] D. H. Yoon and M. Erez, "Memory mapped ecc: Low-cost error protection for last level caches," in *Proceedings of the 36th annual international symposium on Computer architecture*, 2009, pp. 116–127.

[48] D. H. Yoon, N. Muralimanohar, J. Chang, P. Ranganathan, N. P. Jouppi, and M. Erez, "Free-p: Protecting non-volatile memory against both hard and soft errors," in *2011 IEEE 17th International Symposium on High Performance Computer Architecture*. IEEE, 2011, pp. 466–477.

[49] M. Zahran, K. Albayraktaroglu, and M. Franklin, "Non-inclusion property in multi-level caches revisited," *International Journal of Computers and Their Applications*, vol. 14, no. 2, p. 99, 2007.